TEXAS INSTRUMENTS

Application Report
SPRA512

Implementing the TMS320C6201/C6701/C6211 HPI Boot Process

Eric Biscondi *Digital Signal Processing Solutions*

Abstract

The Texas Instruments (TI™) TMS320C62x and TMS320C67x digital signal processors (DSPs) provide a variety of boot configurations that determine which actions the DSP should perform to prepare for initialization after device reset. The boot process is determined by latching the boot configuration settings at reset.

The various boot processes load code from an external ROM memory space and from an external host processor through the host port interface (HPI).

This document describes the following:

❒ How to connect a host with the C6x HPI

❒ The host port interface boot process

❒ An example of C source code for the host processor

❒ How to create a boot code to be downloaded though the HPI

Contents

Figures

Tables

Overview

The TMS320C6000 uses various types of boot configurations, including the following:

- The CPU starts direct execution at address 0.
- A 16K x 32-bit word memory block is automatically copied from a ROM memory space to memory located at address 0.
- A host processor (connected to the DSP through the host port interface) maintains the DSP core in reset while initializing the DSP memory spaces, including external memory spaces.

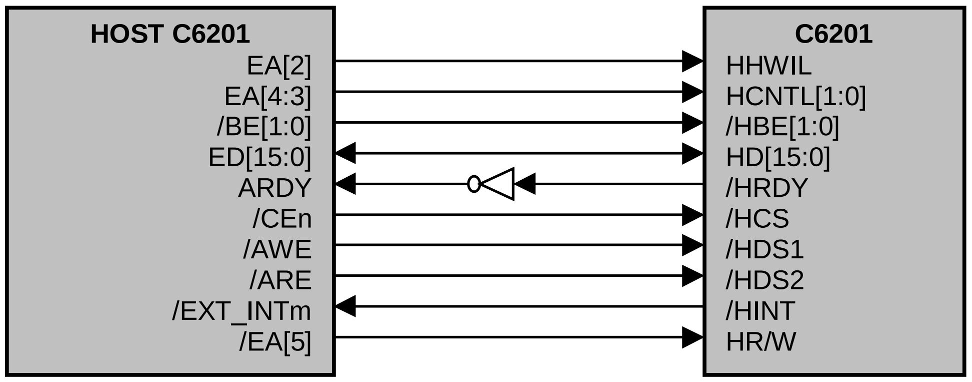
When the HPI boot process is selected, the DSP is held in reset while the remainder of the device is awakened from reset. This means that a host processor connected to the TMS320C6201/C6701/C6211 through the HPI may access and initialize the entire DSP memory space as well as all on-chip peripheral control registers. Once the host has initialized the entire DSP environment, it writes a 1 to the DSPHINT bit in the HPI control register.

This document describes the HPI boot process using an example in which one C6201 (host) talks to another C6201 (slave).

Connecting the C6201/C6701/C6211 to a Host Processor

Some systems require the use of a host processor that communicates with the DSP. The HPI is a dedicated port available on the TMS320C6201/C6701/C6211. The HPI is a 16-bit-wide parallel port through which a host processor can access all of the DSP memory space.

Figure 1. Connecting Two TMS320C6201 DSPs Through the HPI



Using a Host to Boot the TMS320C6201/C6701/C6211

HPI Boot Process

A host processor can directly access the C6201/C6701/C6211 address space through the HPI. This peripheral allows a host processor to exchange information with the DSP.

The HPI can also by used by the host to initialize and load a boot code in the DSP. The HPI boot configuration is selected by the external pins BOOTMODE[4:0] on the C6201/C67001 and HD[4:0] on the C6211.

Table 1. HPI Boot Configuration for the TMS320C6201/C6701

BOOTMODE[4:0]	Memory Map	Memory at Address 0	Boot
00110	MAP 0	External; default values	HPI
00111	MAP 1	Internal	HPI

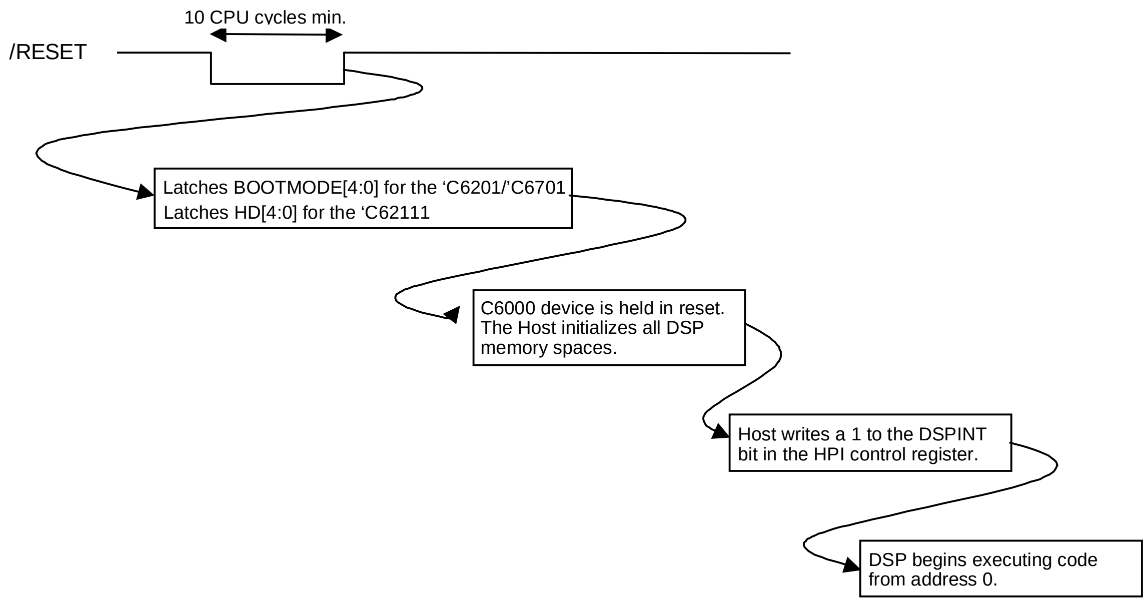
Driving the RESET pin on the processor low and then high resets the device. When the HPI boot process is selected, the DSP core is held in reset while the remainder of the device awakens from reset. At that time, a host processor (connected to the C6x through the HPI) can access all of the C6x memory space, including internal, external, and on-chip peripheral registers.

To release the DSP from its reset state, the host writes a 1 to the DSPINT bit in the HPI control register (HPIC). The CPU then starts the program execution from address 0.

On the C6201/C6701, the HPI boot process can operate in memory MAP1 (the CPU starts from internal program memory) or memory MAP0. In this case, the CPU starts from CE0 with the default values, i.e., 32-bit asynchronous memory with the maximum read/write setup, strobe, and hold time. Therefore, the host can write to memory mapped at 0 without initializing the EMIF.

On the C6211, external pull-up and pull-down resistors connected to the HD[4:0] during reset configure the device. The C6211 operates in one memory MAP only, with internal memory mapped at address 0.

Figure 2. HPI Boot Process Description


10 CPU cycles min.
/RESET
Latches BOOTMODE[4:0] for the ‘C6201/’C6701
Latches HD[4:0] for the ‘C62111
C6000 device is held in reset.
The Host initializes all DSP memory spaces.
Host writes a 1 to the DSPINT bit in the HPI control register.
DSP begins executing code from address 0.

Accessing HPI Registers From a Host

Depending on the connection used between the host and the C6000 DSP, the method used to access HPI registers from the host may differ.

Typically, HPI registers are mapped in the host memory map. HCNTRL[1:0] and HHWIL are connected to address lines of the host processors as shown in Table 2 to select which register is accessed.

Table 2. HPI Control Signals Function Selection Description

HCNTL1	HCNTL0	HHWIL	HPI Register Accessed
0	0	0	HPIC 1st half-word
0	0	1	HPIC 2nd half-word
0	1	0	HPIA 1st half-word
0	1	1	HPIA 2nd half-word
1	0	0	HPID 1st half-word, HPIA is post-incremented.
1	0	1	HPID 2nd half-word, HPIA is post-incremented.
1	1	0	HPID 1st half-word, HPIA not affected.
1	1	1	HPID 2nd half-word, HPIA not affected.

Even if the HPI is a 16-bit external interface, it provides 32 bits to the CPU by combining successive 16-bit transfers. HHWIL identifies the first or second halfword of transfer and the bit HWOB determines the halfword ordering.

Example

Consider the example in Figure 1, in which one C6201 (host) is connected to the HPI of an another C6201 (slave). The HPI is mapped into the asynchronous memory space CE1. The address lines EA[4:2] are used to control the HPI control lines HCNTL[1:0] and HHWIL.

To access the HPI registers, the host performs a memory access to CE1 space as shown in Table 3.

Using C language, a pointer can be used as shown below:

```
#define C6201_HPI 0x01400000  /* Host address on which C6x
                                                HPI is mapped */
int *hpi_ptr;          /* define and initialize pointer*/
             hpi_ptr = (int *)C6201_HPI;
```

Then, following Table 3, the following piece of code can be used to access the HPIA register:

```
/* Write dest_address to HPIA, with HOB=1   */

ptr_hpi[2] = (int)(dest_address & 0x0ffff);
ptr_hpi[3] = (int)((dest_address>>16)&0x0ffff);
```

Table 3. HPI Control Signals Function Selection Description With Host C6x

Address Generated by Host	**HPI Control Lines**		**HPI Register Accessed**
	HCNTL[1:0]	HHWIL	
HPI Base address + 0x00	00	0	HPIC 1st halfword
HPI Base address + 0x04	00	1	HPIC 2nd halfword
HPI Base address + 0x08	01	0	HPIA 1st halfword
HPI Base address + 0x0C	01	1	HPIA 2nd halfword
HPI Base address + 0x10	10	0	HPID 1st halfword, HPIA is post-incremented.
HPI Base address + 0x14	10	1	HPID 2nd halfword, HPIA is post-incremented.
HPI Base address + 0x18	11	0	HPID 1st halfword, HPIA not affected.
HPI Base address + 0x1C	11	1	HPID 2nd halfword, HPIA not affected.

Host Program to Boot Through the HPI

This section considers the example shown in Figure 1 to describe a host program used to boot the C6201. This particular example considers the EMIF of a host DSP C6000 communicating with the HPI of a slave C6000 DSP. The C code presented in Figure 3 can be run without any modifications on the C6000.

You can control the HPI through minor modifications to the C code presented in the example. The main modification required to port this code on another host is to change the way the HPI registers are accessed in accordance with the host memory map, the host-specific data types.

Initializing the TMS320C6000 Through the HPI

In addition to writing code into internal memory, the host may have to download code or data sections into one of the external memory spaces. The host must initialize EMIF registers prior to accessing any external memory spaces.

Figure 3 shows an example of C code that may be run on the host to write a single 32-bit value to the HPI. The host first writes the HPIC setting the HWOB bit, then writes the HPIA and then the HPID.

Figure 3. Function Used to Store a Word in C6x Memory Space Through the HPI

```
void C6x_write_word(int *ptr_hpi, int source_word, int dest_address)
{
     /* Write HPIC with HWOB=1,1st halfword transferred is least significant */

     /*                                      HCNTRL1   HCNTRL0   HHWIL      */
     ptr_hpi[0] = 0x0001; /* 1st halfword         0         0         0      */
     ptr_hpi[1] = 0x0001; /* 2nd halfword         0         0         1      */


     /* Write destination address to HPIA, 1st halfword is least significant */

     /*                                         HCNTRL1   HCNTRL0   HHWIL   */
     ptr_hpi[2] = (int)(dest_address & 0x0ffff);/*      0      1      0      */
     ptr_hpi[3] = (int)((dest_address>>16)&0x0ffff);/*  0      1      1      */


     /* Write source_word to HPID without address post-increment            */
     /* 1st half-word transferred is least significant                      */

     /*                                         HCNTRL1   HCNTRL0   HHWIL   */
     ptr_hpi[6] = (int)(source_word&0x0ffff); /*     1         1         0   */
     ptr_hpi[7] = (int)((source_word>>16)&0x0ffff);/*1         1         1   */
}
```

Appendix A provides a complete example. Lines 46 to 52 correspond to the EMIF initialization performed by the host processor through the HPI.

During the HPI boot process, only the CPU is maintained in reset. All the peripherals may be active. By accessing the on-chip peripheral registers, the host can initialize and start any C6201 peripheral. For example, depending on the system requirements, the host may have to initialize and start one serial port or DMA transfer.

Transferring Code and Data Sections

A program is composed of initialized sections and non-initialized sections. The host processor must load sections in the C6000 DSP to the correct address in accordance with the link command file.

The host must write a complete section at a given address. Figure 4 shows an example of C function that reads length 32-bit words of data from **source* and then writes through the HPI to the C6000 DSP address, dest_addr.

Figure 4. Function Used to Store a Buffer in C6x Memory Space Through the HPI

```
void C6x_write_section(int *ptr_hpi, short *source, int dest_add, int length)
{
int i;

     /* Write HPIC with HWOB=1,1st halfword transferred is least significant */

     /*                                       HCNTRL1   HCNTRL0   HHWIL      */
     ptr_hpi[0] = 0x0001; /* 1st halfword         0         0         0      */
     ptr_hpi[1] = 0x0001; /* 2nd halfword         0         0         1      */

     /* Write destination address to HPIA, 1st halfword is least significant */

     /*                                          HCNTRL1   HCNTRL0   HHWIL   */
     ptr_hpi[2] = (int)(dest_add & 0x0ffff);   /*     0         1        0    */
     ptr_hpi[3] = (int)((dest_add>>16)&0x0ffff);/*   0         1        1    */


     for(i=0 ; i < length ; i++)
     {
 /* Write source_word to HPID with address post-increment    */
 /* 1st half-word transferred is least significant          */
 /*          HCNTRL1     HCNTRL0     HHWIL  */
 ptr_hpi[4] = (int) *source++; /*  1  0  0  */
 ptr_hpi[5] = (int) *source++; /*  1  0  1  */
     }
}
```

The pointer **source* points to the location where the boot code for the slave C6000 DSP is stored. For example, the pointer might point to:

- ❒ External ROM mapped in the host memory map and containing the C6x boot code
- ❒ Data array (linked with host code) containing the boot code of the slave C6000 DSP
- ❒ Host peripheral that can receive the boot code of the slave C6000 DSP (for example, a serial port)

The second option is used in the complete example shown in Appendix A. Lines 21 and 22 give the inclusion of the header files containing the code (*code.h*) and the initialized data (*initia.h*).

Notice that this solution requires a recompilation of the host code each time the DSP code is modified. It also requires an automatic way to create a C array (containing the C6x program and initialized data) from a COFF file. Please refer to Appendix B for a complete description of this process.

Remove the Slave TMS320C6000 From Its Reset State

Once the host processor has performed all initialization and loaded all of the code and data sections into the C6201 memory spaces, it must release the C6201 from its reset state by writing a 1 in the DSPINT bit.

In the example we are considering:

```
/* Write HPIC with DSPINT=1                              */

/*                          HCNTRL1   HCNTRL0   HHWIL       */
/* 1st halfword        0         0         0       */
/* 2nd halfword        0         0         1       */

ptr_hpi[0] = 0x0002; /* 1st halfword */
ptr_hpi[1] = 0x0002; /* 2nd halfword */
```

Once DSPINT is written to 1, the CPU starts at address 0.

Creating a C6x Boot Code to be Downloaded by the Host

This section discusses how to built DSP code to be downloaded from the host through the host port interface.

As shown in the previous sections, the host processor has to write all code sections and all initialized data sections through the HPI to the C6000 memory space. The C6000 code has to be linked with the initialization of variables at load time (by invoking the linker with the option –cr), which enhances performance by reducing boot time and saving the memory used by the initialized data sections.

Figure 6 shows an example of a linker command file. (Figure 7 shows the beginning of a vector table example.) After linking C6x code, it has to be stored in a memory accessible by the host processor.

Basically, there are two main cases:

❒ If the host reads C6x code from an external memory containing only the DSP code, the user first has to program ROM with the C6x code.

  Texas Instruments provides a hex conversion utility that converts the output of the linker (a COFF object file) into one of the several standards suitable for loading into an EEPROM programmer. Figure 5 shows an example of a command file for the hex conversion utility that builds four files to program four 8-bit EEPROMs (assuming the host is connected to four 8-bit EEPROMs).

Figure 5. Command File for Hex Converter Utility (Four 8-Bit EEPROMs)

```
main.out
-i
-byte
-image
-memwidth 32
-romwidth 8
-order L

ROMS
{
   EPROM:  org = 0x0, length = 0x20000
   files = {u22.int, u24.int, u23.int, u25.int}
}
```

- If the host reads C6000 code from an external memory that does not contain only the DSP code, the user has to include both the host code and the DSP code into the same memory. In this case, the user has to link the C6000 code and the host code together.

  We have to convert the C6000 COFF file into a suite of data that can be linked with the host program. As most of the host applications are written in C language, we decided to develop a tool to convert a COFF file into a C array of bytes.

  This is typically the option used in the example described in Appendix A, in which the C6000 code sections are included in the header file *code.h* and C6000 initialized data sections are included in the file *initia.h*. Appendix B describes the procedure to build *code.h* and *initia.h*. This method requires a re-compilation of the host code each time the DSP code is modified.

Figure 6. Command File for the Linker

```
/*****************************************************/
/*  lnk.cmd                                          */
/*  Copyright ©   1996-1997  Texas Instruments Inc.  */
/*****************************************************/
-cr
vector.obj
main.obj

-o main.out
-heap  0x0200
-stack 0x0200
-l rts6201.lib

MEMORY
{
   VECS:  o = 00000000h     l = 0000200h
   PMEM:  o = 00000200h     l = 000FC00h
   DMEM:  o = 80000000h     l = 0010000h
   CE0:   o = 00400000h     l = 1000000h
   CE1:   o = 01400000h     l = 0010000h
   CE2:   o = 02000000h     l = 1000000h
   CE3:   o = 03000000h     l = 1000000h
}

SECTIONS
{
    vectors >      VECS
    .text   >      PMEM
    .far    >    DMEM
    .stack  >      DMEM
    .bss    >      DMEM
    .sysmem >      DMEM
    .cinit  >      DMEM
    .cio    >      DMEM
    .const  >      DMEM
    .data   >      DMEM
}
```

Figure 7. Example Interrupt Vector Table (vector.asm)

```
/****************************************************/
/*  vector.asm                                      */
/*  Copyright ©   1996-1997  Texas Instruments Inc.  */
/****************************************************/
.ref      _c_int00,_c_nmi01,_c_int04,_c_int05,
          .ref   _c_int06,_c_int07,_c_int08,_c_int09,
          .ref   _c_int10,_c_int11,_c_int12,_c_int13
          .ref   _c_int14, _c_int15
          .sect  vectors
RESET:        B   .S2    _c_int00
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
NMI:          B   .S2    _c_nmi01
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
RESV1:        B   .S2    RESV1
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
RESV2:        B   .S2    RESV2
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
INT4:         B   .S2    _c_int04
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP

INT5:         B   .S2    _c_int05
              NOP
                 NOP ……
```

Appendix A. Host Source Code to Boot the C6x Through the HPI

```
/***************************************************************************/
/* Host.c: Host program to boot load the C6x through the HPI.   */
/*     This program is an example which assumes that host needs to  */
/*     initialize first the external memory configuration registers   */
/*     and then needs to download the .text, .cint and .const    */
/*               */
/* Author : Eric Biscondi           */
/* Date   : 24 dec 97           */
/* Modifications:            */
/*               */
/*               */
/*       (c) Texas Instruments France   */
/****************************************************************************/
#include <stdio.h>
#include <stdlib.h>

/* Header files containing the code to program into the flash */
#include "code.h"/* contains initialized sections of code */
#include "initia.h" /* contains initialized sections of data */

#define C6201_HPI 0x01600000 /* Address of the 'C6201 HPI*/
#define DEBUG  0  /* Flag for conditional DEBUG info */


void C6x_write_section(int *ptr_hpi, short *source, int dest_add, int length);

void C6x_write_word(int *ptr_hpi, int source_word, int dest_address);

void init_host(void);


void main(void)
{
int *ptr_hpi;
int i, number_code, number_init;

  ptr_hpi = (int *)C6201_HPI;

  init_host(); /* Initialization of the Host processor */

  /* Initialization of the 'C6201 's EMIF    */

  C6x_write_word(ptr_hpi, 0x0000377d, Emif_global_control);
  C6x_write_word(ptr_hpi, 0x00000040, Emif_CE1_control);
  C6x_write_word(ptr_hpi, 0x00000030, Emif_CE0_control);
  C6x_write_word(ptr_hpi, 0x00000030, Emif_CE2_control);
  C6x_write_word(ptr_hpi, 0xffffff23, Emif_CE3_control);
  C6x_write_word(ptr_hpi, 0x03166000, Emif_SDRAM_control);
  C6x_write_word(ptr_hpi, 0x00000aaa, Emif_SDRAM_refresh);


  /* Determine the number of halfword contained in the code section */
  number_code = sizeof(code) / sizeof(code[0]);
  /* Write the code sections into the C6x memory mapped at 0 */
  C6x_write_section(ptr_hpi, (short *)&code , 0x0, number_code);


  /* Determine the number of halfword contained in the data section */
  number_init = sizeof(initia) / sizeof(initia[0]);

  /* Write the cinit sections into the C6x internal data memory */
  C6x_write_section(ptr_hpi,(short*)&initia,0x80000000, number_init);

#if DEBUG
  printf("TMS320C6201 boot code loaded\n");
#endif

       /* Wake up TMS320C6201 */
  ptr_hpi[0] = 0x0003; /* Writes 1st half to HPIC - 0x01600000 */

  ptr_hpi[1] = 0x0003; /* Writes 2nd half to HPIC - 0x01600004 */

#if DEBUG
  printf("TMS320C6201 is running \n");
#endif
}


void init_host(void)
{
  /* Initialize CE1 as an Asynchronous memory space       */
  *(int *)0x01800004 = 0x00e20322;
}


/***********************************************************************/
/* C6x_write_word                                       */
/* This routine is downloading data from source address to the C6x */
/* dest_address through the C6x Host Port Interface.   */
/* This routine accesses the HPID without automatic address increment */
/*              */
/* Inputs:             */
/* ptr_hpi:  pointer to the C6x HPI vase address    */
/* source_word:  address of the data to transfer to the C6x  */
/* dest_address: destination address to write to the C6x HPIA   */
/*              */
/*      (c) Texas Instruments France  */
/**********************************************************************/
void C6x_write_word(int *ptr_hpi, int source_word, int dest_address)
{
   /* Write HPIC with HWOB=1,1st halfword transferred is least significant */

     /*                                        HCNTRL1   HCNTRL0   HHWIL      */
     ptr_hpi[0] = 0x0001; /* 1st halfword        0         0         0      */
     ptr_hpi[1] = 0x0001; /* 2nd halfword        0         0         1      */


     /* Write destination address to HPIA, 1st halfword is least significant  */

     /*                                           HCNTRL1   HCNTRL0   HHWIL   */
     ptr_hpi[2] = (int)(dest_address & 0x0ffff);/*      0      1      0      */
     ptr_hpi[3] = (int)((dest_address>>16)&0x0ffff);/*  0      1      1      */
```

```
          /* Write source_word to HPID without address post-increment             */
          /* 1st half-word transferred is least significant                       */

          /*                                            HCNTRL1   HCNTRL0   HHWIL   */
          ptr_hpi[6] = (int)(source_word&0x0ffff); /*     1         1         0   */
          ptr_hpi[7] = (int)((source_word>>16)&0x0ffff);/*1         1         1   */
     }


     /**************************************************************************/
     /* C6x_write_section                                    */
     /* This routine is downloading data from source address to the C6x */
     /* dest_address through the C6x Host Port Interface.   */
     /* This routine accesses the HPID with automatic address increment */
     /*              */
     /* Inputs:             */
     /* ptr_hpi:  pointer to the C6x HPI vase address    */
     /* source_word: address of the data to transfer to the C6x  */
     /* dest_address: destination address to write to the C6x HPIA   */
     /* length: number of data to transfer      */
     /*              */
     /*       (c) Texas Instruments France */
     /**************************************************************************/
     void C6x_write_section(int *ptr_hpi, short *source, int dest_add, int length)
     {
     int i;

          /* Write HPIC with HWOB=1,1st halfword transferred is least significant */

          /*                                       HCNTRL1   HCNTRL0   HHWIL      */
          ptr_hpi[0] = 0x0001; /* 1st halfword         0         0         0      */
          ptr_hpi[1] = 0x0001; /* 2nd halfword         0         0         1      */

          /* Write destination address to HPIA, 1st halfword is least significant */

          /*                                            HCNTRL1   HCNTRL0   HHWIL   */
          ptr_hpi[2] = (int)(dest_add & 0x0ffff);   /*     0        1        0    */
          ptr_hpi[3] = (int)((dest_add>>16)&0x0ffff);/*   0        1        1    */


          for(i=0 ; i < length ; i++)
          {
       /* Write source_word to HPID with address post-increment    */
       /* 1st half-word transferred is least significant          */
       /*           HCNTRL1     HCNTRL0     HHWIL  */
       ptr_hpi[4] = (int) *source++; /*  1  0  0 */
       ptr_hpi[5] = (int) *source++; /*  1  0  1 */
          }
     }
```

Appendix B. Building a C Array of Values From a COFF File

To build a C array of values from a COFF file, the first step is to use the hex conversion utility to convert the COFF file into hexadecimal files. Figure 8 shows an example of a command file for the hex converter utility to convert the COFF file *C6201code.out* into two hex files: one for the code sections and another for the initialized data sections.

The code section is included in a file called *C6201code.a00*. The initialized data sections are included in a file called *C6201code.a01*.

Figure 8. Command File for the Hex Converter Utility

```
C6201code.out
-a
-byte
-image
-memwidth 32
-romwidth 32
-order M

ROMS
{
        /* Size of the internal pgm memory */
        PGM:        org = 0x00000000, length = 0x10000

        /* Size of the internal data memory */
        DATA:       org = 0x80000000, length = 0x10000
}
```

Then we developed a tool (*hex2aray.exe*) that takes a hexadecimal file as input and produces a C array of data included into a header file.

hex2aray –i <hex_input_file_name> –o <header_file_name>

In our example, we must invoke hex2aray to build *code.h* from C6201code.a00 and *initia.h* from C6201code.a01.

Figure 9 gives an example of a header file created by:

hex2aray –i C6201code.a00 –o code.h

Figure 9. Example Header File code.h Created by hex2aray.exe

```
/***************************************************************************/
/*  Header file containing C6x code generated from HEX2ARAY.EXE            */
/*                                                                         */
/*                                                                         */
/*  Date/Time created: 02/23/1998   15:40:16                               */
/*                                                                         */
/*                                (c) Copyright Texas Instruments          */
/***************************************************************************/

const char code[]={0x12,0x18,0x01,0x00,0x28,0x00,0x00,0x00,0x2A,0x00,0x00,
                       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x12,
                       0xE0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x12,0x00,
                       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x12,0x00,0x00,
                       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

                       …
                       };
```

CAUTION:

Note that the name of the array is the same as the file name.

The source code for *hex2aray.exe* is shown in Figure 10.

Figure 10. Source Code for hex2aray.c

```
/*============================================================================
//File : hex2aray.c
//Take an ASCII file generated by hex6x converter and convert it
//in an array of byte in C language.
//
//Date : November 26, 1997
//Author : Eric Biscondi
//Modification :
//
//                        (c) Copyright Texas Instruments France
//============================================================================*/

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <string.h>

extern directvideo;

void main(int argc,char *argv[])
{
FILE *out_file,*in_file;
char chn_fileo[32],chn_filei[32], name_array[32], charac;
int i,i1,in,out,out1,out2,line_count, zero_count, j, zero;
struct  time t;
struct date d;

      zero = 0x30;

      /*arguments coming from the shell   -------------------------------*/
      if ((argc>0)&&(strcmp(argv[1],"?")==0))
      {
            printf ("dspc6x  [-i <file_in>] [-o <file_out>]\n");
      }

      /*initialisation with shell's arguments----------------------------*/
      for (i=1; i<(argc); i++)
      {
            if (!strcmp(strlwr(argv[i]),"-i"))
            {
                  if (++i<=argc)
                  {
                        strcpy(chn_filei,argv[i]);
                  }
            }
            else if (!strcmp(strlwr(argv[i]),"-o"))
                  {
                        if (++i<=argc)
                        {
                               strcpy(chn_fileo,argv[i]);
                        }
                  }
                  else
                  {
                        printf ("dspc6x  [-i <file_in>] [-o <file_out>]\n");
                  }
      }

      /*open in file-----------------------------------------------------*/

      if ((in_file = fopen(chn_filei,"rb")) == NULL)
      {
            printf("Unable to open input file \n");
            exit (1);
      }
      if ((out_file = fopen(chn_fileo,"wb")) == NULL)
      {
            printf("Unable to open output file \n");
            exit (1);
      }

      gettime(&t);
      getdate(&d);

fprintf(out_file,"/**************************************************************************/\n");
fprintf(out_file,"/*  Header file containing C6x code generated from HEX2ARAY.EXE            */\n");
fprintf(out_file,"/*                                                          */\n");
fprintf(out_file,"/*                                                          */\n");
fprintf(out_file,"/*  Date/Time created: %02d/%02d/%02d   %2d:%02d:%02d                      */\n",
                            d.da_mon, d.da_day, d.da_year, t.ti_hour, t.ti_min, t.ti_sec);
fprintf(out_file,"/*                                                           */\n");
fprintf(out_file,"/*                                          (c) Copyright Texas Instruments*/\n");
fprintf(out_file,"/***************************************************************************/\n");

/* Suppress the file name extension */
      i=0;
      do
      {
      sscanf(chn_fileo,"%1s %s", &name_array[i], &chn_fileo);
      }while( name_array[i++] != '.');

      name_array[i-1]='\0';

      fprintf(out_file, "\n\n\nconst char %s[]={",name_array);

      line_count = 0;
      zero_count = 0;
      while((in = fgetc(in_file)) != EOF)
      {
            i = in&0x0ff;
            switch(i) {
                  case(0x02):
                        break;
                  case(0x20):
                        break;
                  case(0x0d):
                        break;
                  case(0x0a):
                        break;
                  case(0x24):                 /* Case $ */
                        while( (in=fgetc(in_file)) != 0x0a);
                        break;
                  default:
                        if( (in = fgetc(in_file)) == EOF) break;
                        i1 = in&0x0ff;

                        if( (i==0x30) && (i1==0x30) ) /* Zero detection */      {
                               zero_count++;
                        }
                        else   {
                               if(zero_count != 0)        {
                                             for(j=0 ; j < zero_count ; j++)        {
                                                 line_count++;
                                                 fprintf(out_file,"0x");
                                                 fputc(zero, out_file);
                                                 fputc(zero, out_file);
                                                 fprintf(out_file,",");    /* Printf "," */
                                                 if (line_count>10) {
                                                      line_count = 0;
                                                      fprintf(out_file,"\n          ");
                                                 }
                                             }
                                             zero_count=0;
                               }
                               line_count++;
                               fprintf(out_file,"0x");
                               fputc(i, out_file);
                               fputc(i1, out_file);
                               fprintf(out_file,",");     /* Printf "," */
                               if (line_count>10) {
                                             line_count = 0;
                                             fprintf(out_file,"\n          ");
                               }
                        }
            }
      }
      fprintf(out_file,"0x00};");
      fcloseall();
}
```

References

[1] *TMS320C62xx Peripherals Reference Guide*, Texas Instruments 1997, Literature number SPRU190.

[2] *TMS320C6x Optimizing C Compiler User's Guide*, Texas Instruments 1997, Literature number SPRA187.

[3] *TMS320C6x Assembly Language Tools–User's Guide*, Texas Instruments 1997, Literature number SPRU186.

TI Contact Numbers

INTERNET

TI Semiconductor Home Page
www.ti.com/sc

TI Distributors
www.ti.com/sc/docs/distmenu.htm

PRODUCT INFORMATION CENTERS

Americas

Phone	+1(972) 644-5580
Fax	+1(972) 480-7800
Email	sc-infomaster@ti.com

Europe, Middle East, and Africa

Phone	
Deutsch	+49-(0) 8161 80 3311
English	+44-(0) 1604 66 3399
Español	+34-(0) 90 23 54 0 28
Francais	+33-(0) 1-30 70 11 64
Italiano	+33-(0) 1-30 70 11 67
Fax	+44-(0) 1604 66 33 34
Email	epic@ti.com

Japan

Phone	
International	+81-3-3457-0972
Domestic	0120-81-0026
Fax	
International	+81-3-3457-1259
Domestic	0120-81-0036
Email	pic-japan@ti.com

Asia

Phone	
International	+886-2-23786800
Domestic	
Australia	1-800-881-011
TI Number	-800-800-1450
China	10810
TI Number	-800-800-1450
Hong Kong	800-96-1111
TI Number	-800-800-1450
India	000-117
TI Number	-800-800-1450
Indonesia	001-801-10
TI Number	-800-800-1450
Korea	080-551-2804
Malaysia	1-800-800-011
TI Number	-800-800-1450
New Zealand	000-911
TI Number	-800-800-1450
Philippines	105-11
TI Number	-800-800-1450
Singapore	800-0111-111
TI Number	-800-800-1450
Taiwan	080-006800
Thailand	0019-991-1111
TI Number	-800-800-1450
Fax	886-2-2378-6808
Email	tiasia@ti.com