Booting the TMS320C6201 through the Host Port Interface

APPLICATION REPORT: PRELIMINARY

Author : Eric Biscondi
LBE : DSP
Date : March 24, 1998

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

Contents

Figures

Tables

Booting the TMS320C6201 through the Host Port Interface

Abstract

Three types of boot processes are available on the TMS320C6201. This document discusses the HPI boot mode and it describes:

- how to connect a host with the ‘C6x ‘s 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.

Overview

The ‘C6201 uses various types of boot configuration. Mainly there are three types of boot process:

- The CPU starts direct execution at address 0.
- A 16K 32-bit words memory block is automatically copied from the beginning of the CE1 memory space to memory located at the address 0 through the DMA channel 0.
- A Host processor (connected to the ‘C6201 through the Host Port Interface) maintain the ‘C6201 core in reset while initializing the ‘C6201 memory space, including external memory spaces.

When the HPI boot process is selected by the BOOTMODE[4:0] pins during the reset, the ‘C6201’s CPU is held in reset while the remainder of the device is awakes from reset. That means that a host processor connected to the TMS320C6201 through the HPI may access and initialize all the ‘C6210 ‘s memory space as well as all the on-chip peripherals control registers. Once the host has initialized all the ‘C6201 environment, it writes a 1 to the DSPHINT bit in the HPI control register.

This documents describes the HPI boot process through a example in which one ‘C6x (the host) is talking to another.

How to connect the ‘C6201 to the Host Processor

Some systems are requiring the use of a host processor that communicates with the DSP. A dedicated port is available on the TMS320C6201: the Host Port Interface (HPI). The HPI is a 16-bit wide parallel port through which a host processor can access all the CPU’s memory space.

Figure 1 and Figure 2 give examples of connection between the ‘C6201 and host processors.

Figure 1. Example of connection between two TMS320C6201 through the HPI

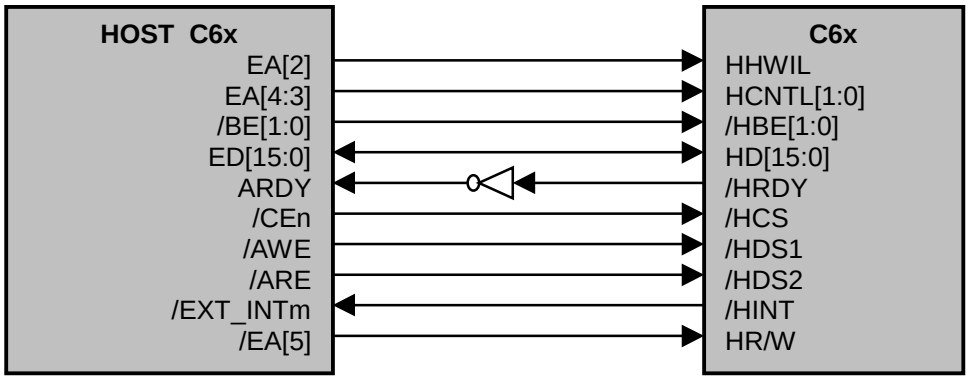
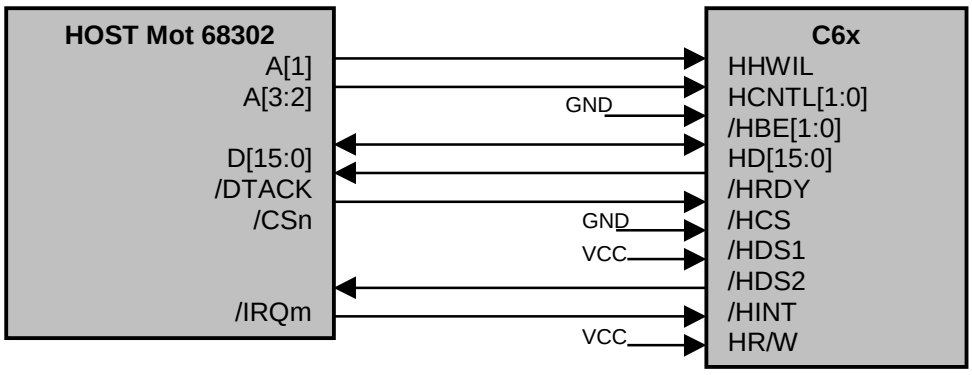


Figure 2. Example of connection between two TMS320C6201 through the HPI



Please refer to [5] for a complete description of the interface between a host processor and the ‘C6201’s Host Port Interface.

Table 1 describes the HPI external interface signals.

Table 1. HPI External Interface Signals

Signal Name	Signal Type (**I**nput/ **O**utput/ HI-**Z**)	Signal Count	Host Connection	Signal Function
HD15:0	I/O/Z	16	Data Bus	
HCNTL1:0	I	2	Address or control lines	Controls HPI access type.
HHWIL	I	1	Address or control lines	Halfword identification input.
HAS-	I	1	Address latch enable (ALE) or Address strobe or unused (tied high)	Differentiates address versus data values on multiplexed address/data host.
HBE-1:0	I	2	Byte enables	Data write byte enables
HR/W-	I	1	Read/Write strobe, address line, or multiplexed address/ data	Read/Write select
HCS-	I	1	Address or control lines	Data strobe inputs.
HDS1- HDS2-	I	2	Read strobe and write strobe or data strobe	Data strobe inputs.
HRDY-	O	1	Asynchronous ready	Ready status of current HPI access
HINT-	O	1	Host interrupt input.	Interrupt signal to Host

The 16-bit data bus (HD0-HD15) exchanges information with the host. Because of the 32-bit word structure of the chip architecture, all transfers with a host consist of two consecutive 16-bit half-words.

On host data (HPID) write accesses, the HBE[1:0]- byte enables select which bytes in a 32-bit accesses should be written.

HCNTL[1:0] indicate which internal HPI register is being accessed. The states of these two pins select access to the HPI address (HPIA), HPI data (HPID), or HPI control (HPIC) registers. Additionally, the HPID register can be accessed with an optional automatic address increment.

HPIA, HPIC, and HPID read accesses are performed as 32-bit accesses, and the byte enables are not used. The dedicated HHWIL pin indicates whether the first or second half-word is being transferred. An internal control register bit determines whether the first or second half-word is placed into the most significant half-word of a word.

The 16-bit data bus (HD0-HD15) exchanges information with the host. Because of the 32-bit word structure of the chip architecture, all transfers with a host consist of two consecutive 16-bit half-words.

On host data (HPID) write accesses, the HBE[1:0]- byte enables select which bytes in a 32-bit accesses should be written.

HCNTL[1:0] indicate which internal HPI register is being accessed. The states of these two pins select access to the HPI address (HPIA), HPI data (HPID), or HPI control (HPIC) registers. Additionally, the HPID register can be accessed with an optional automatic address increment.

HPIA, HPIC, and HPID read accesses are performed as 32-bit accesses, and the byte enables are not used. The dedicated HHWIL pin indicates whether the first or second half-word is being transferred. An internal control register bit determines whether the first or second half-word is placed into the most significant half-word of a word.

Using a host to boot the TMS320C6201

The HPI boot process

A host processor can directly access to the ‘C6201 memory space through the Host Port Interface. This peripheral allows a host processor to exchange information with the ‘C6201.

The HPI may also by used by the host to initialize and load a boot code in the ‘C6201. The HPI boot configuration is selected by the external pins BOOTMODE[4:0].

Table 2. HPI Boot configuration

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

The HPI boot process can operate in memory MAP1 (the CPU starts from internal program memory) or memory MAP0. In that case, 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.

When the RESET pin on the processor is driven low, then high, the device is reset. When the HPI boot process is selected, the CPU is then held in reset while the remainder of the device awakes from reset. At that time, a host processor (connected to the ‘C6201 through the HPI) can access all ‘C6201’s memory space, including internal, external and on-chip peripheral registers.

To release the ‘C6201’s CPU from its reset state, the host has to write a 1 to the DSPINT bit in the HPI control register (HPIC). The CPU then starts the program execution from address 0.

Accessing HPI registers from a Host

Depending on the connection used between the host and the ‘C6201, the way to access HPI registers from the host may differ.

Typically, the ‘C6x‘s HPI registers are mapped in the host memory map. HCNTRL[1:0] and HHWIL are connected to address lines of the host processors to select which register is accessed, following the Table 3.

Table 3. 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-bit to the CPU by combining successive 16-bit transfers. HHWIL identifies the first or second half-word of transfer and the bit HWOB determines the halfword ordering.

Example:

Let’s consider example given on Figure 1 and in which one C6x (host) is connected to the HPI of an another. 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 has to perform a memory access to CE1 space as shown on Table 4.

Using C language, a pointer may 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 4, the following piece of code may be used to access 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 4. HPI Control Signals Function Selection Description with a host C6x

Address generated by the host	HPI Control lines		HPI Register accessed
	HCNTL[1:0]	HHWIL	
HPI Base address + 0x00	00	0	HPIC 1st half-word
HPI Base address + 0x04	00	1	HPIC 2nd half-word
HPI Base address + 0x08	01	0	HPIA 1st half-word
HPI Base address + 0x0C	01	1	HPIA 2nd half-word
HPI Base address + 0x10	10	0	HPID 1st half-word, HPIA is post-incremented.
HPI Base address + 0x14	10	1	HPID 2nd half-word, HPIA is post-incremented.
HPI Base address + 0x18	11	0	HPID 1st half-word, HPIA not affected.
HPI Base address + 0x1C	11	1	HPID 2nd half-word, HPIA not affected.

Host program to boot the ‘C6201

This chapter is considering the example given on Figure 1 and describes a host program to boot load the ‘C6201. This particular example considers one C6x (host) talking to another. The C code presented below can be run without any modifications on a C6x.

The user to support another host can easily modify the C code presented in that example. The main modification to port this code on another host, is to change the way to access the HPI registers in accordance with the host memory map, the host specific data types.

Initialization of the TMS320C6201

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

On Figure 3 is shown an example of C code which may be run on the host to write a single 32-bit value to the ‘C6201. The host first writes the HPIC setting the HWOB bit, then it writes the HPIA, and then HPID.

Figure 3. Function to store a word in the '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    */
}
```

On Appendix A is given 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's peripherals. For example and depending on the system requirements, the host may have to initialize and start one serial port, or a DMA transfer.

Transferring code and data sections

A program is composed with initialized sections and non-initialized sections. The host processor has to load sections in the 'C6201 to the correct address, in accordance with the link command file.

The host has to write a complete section at a given address. On Figure 4 is given an example of C function which reads *length* 32-bit words data from **source* and then, write through the HPI to the C6x's address *dest_addr*.

Figure 4. Function to store a buffer in the 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* point to the location where the C6x boot code is stored. For example, this pointer might point to:

- an external ROM, mapped in the host memory map, containing the C6x boot code,
- an data array (linked with host code) containing the C6x boot code,
- a host peripheral which can receive the C6x boot code (for example a serial port).

The second option is used in the complete example given in Appendix A. On Lines 21 and 22 is given 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.

Remove the TMS320C6201 from its reset state

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

In the example we are considering in that document:

```
/* 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 chapter discusses how boot code can be generated either sing C code or assembly code. As we have seen in the previous chapters, the host processor has to write, through the HPI to the C6x memory space, all code sections and all initialized data sections.

The user has the possibility or not to use the auto-initialization of variables at load-time or at run-time.

When using auto-initialization at load-time (option –cr), the host has the responsibility of initializing all variables and to initialize the stack pointer.

When using auto-initialization at run-time (option –c), the linker will automatically include a function (c_int00), before the call of the function *main()*.

Figure 5 gives an example of linker command file in which the au-initialization at run-time is used. In that case, the host program has just to transfer the code section and the initialized data sections (.cinit, .const) in the C6x memory, then the function c_int00 (created by the –c option) will perform the stack pointer initialization and will initialize all global variables by copying the data from the initialization tables in the .cinit section to the .bss sections. (See [3] section 8.8 for any further details). This is typically the option used in the example given in Appendix A.

Figure 5. Command file for the linker

```
/*****************************************************/
/*  lnk.cmd                                          */
/*  Copyright ©   1996-1997  Texas Instruments Inc.  */
/*****************************************************/
-c
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
}
```

If the host is reading the C6x code from an external memory, the user has first to program ROM with the C6x code. Texas Instruments is providing a hex conversion utility, which converts the output of the linker (a COFF object file) into one of the several standards suitable for loading into an EEPROM programmer. Figure 6 shows an example of command file for the hex conversion utility that builds four files to program four 8-bit EEPROM. Assuming the host is connected to four 8-bit EEPROM. Refer to [4] Chapter 9 for any further details about the hex conversion utility.

Figure 6. Command file for the hex converter utility (four 8-bit EEPROM)

```
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}
}
```

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>

#include "test6201.h"
#include "prts.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       */
110

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

        /*                                          HCNTRL1   HCNTRL0   HHWIL   */
115     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             */
120     /* 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   */
125 }


    /*************************************************************************/
    /* C6x_write_section                                                     */
130 /*     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:                                                               */
135 /*     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                           */
    /*                                                                       */
140 /*                                    (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

******

Could include here a brief explanation on the use of a tool to convert an ASCII file into a C array of data

******

References

[1]. TMS320C62xx Peripherals Reference Guide, Texas Instruments 1997.

[2]. Application Report : Interfacing TMS320C62xx to external Asynchronous SRAM, Texas Instruments 1998.

[3]. TMS320C6x Optimizing C Compiler User's Guide, Texas Instruments 1997.

[4]. Application Report: Interfacing the TMS320C62xx to External Flash Memory, Texas Instruments 1997.

[5]. Application Report: TMS320C62xx Host Port Interface Application, Texas Instruments 1998.

[6]. TMS320C6x Assembly Language Tools – User's Guide, Texas Instruments 1997.