Technical Notes on using Analog Devices' DSP components and development tools
Phone: (800) ANALOG-D, FAX: (781) 461-3010, EMAIL: dsp.support@analog.com, FTP: ftp.analog.com, WEB: www.analog.com/dsp

In-System-Programming (ISP) of ADSP-2106x boot-images into FLASH Memories

Contributed by HS, last change 17-06-2000

Introduction:

Modern embedded DSP systems are these days equipped with non-volatile memory devices such as EEPROM or FLASH memories. This allows in an easy way a later reprogramming with a new content either to correct problems of the current firmware or to enhance capabilities of the system.

All members of the ADSP-2106x SHARC family can be booted from a single 8bit wide memory device. The Engineer- to- Engineer Note EE-55 discusses the connection to and access of byte-programmable FLASH memory components thoroughly. To understand and transfer the content of this application note to the own system, the user should be familiar with the content of EE-55.

This Engineer to Engineer Note will describe the process of generating the required boot image for the FLASH memory device using the v tools set and how a first time start-up of the system can be achieved.

Declarations and Naming:

Before heading on, it is necessary to define common terms: the User Application Program (UAP) is the software written by the user which will later function on the embedded DSP system as application software running as the main task or little operating system. The boot image of this software is called User Application Boot Image or UAB.

As the ADSP-2106x is the processor which will program the FLASH device while it is mounted in the system (in- system- programming or on- target-programming), it is necessary to generate a FLASH Programming Module (FPM) around the UAB performing this task. As this software is downloaded to system using one of the available In- Circuit-Emulators (EZ-ICE) provided by White Mountain DSP, there is no need to generate a boot image of it.

Preparations:

Before the user can start the on- target-programming sequence, a well- tested and valid user application must be present. Please verify first proper execution of your UAP source code before programming it into the FLASH device.

Having passed this first requirement, the user can continue with the process by generating the boot image of the user application code (UAB). This is supported through the ADSP-2106x loader pane in the v Options dialog, as shown in figure 1.

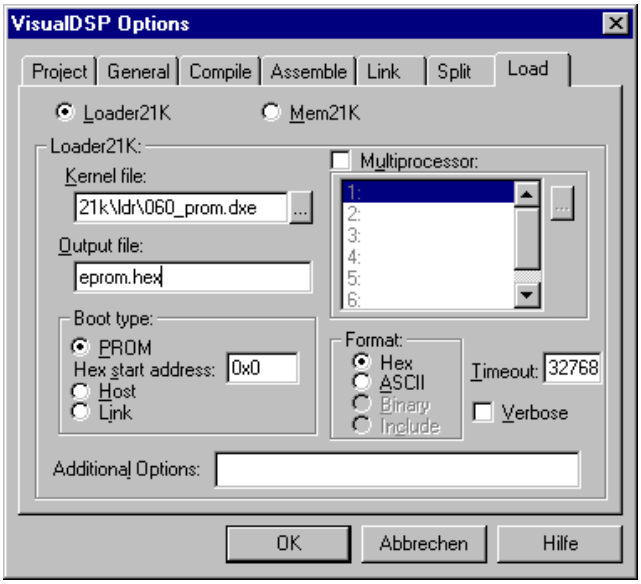

Figure1: Loader Options Pane

As the ADSP-2106x will boot the user application from FLASH memory, the **PROM** boot method must be selected. Now, the user has several options for his target system, which can be selected by the boot kernel file, the multi- processor option check box, a boot image memory offset and finally the type of the output file. Please be aware that multiple DSP evaluation systems from Analog Devices have processor IDs set to "1" and not to "0". This requires the multiprocessor option, even for single DSP systems. For further documentation of the switches, please consult your v documentation.

The easiest processing of the User Application Boot image is given when as output file format **ASCII** is selected, although the pane defaults to hex. This will generate a boot stream nicely sorted in 16bit

wide words, where following mapping as shown in table 1 can be found:

16bit ASCII		BOOT IMAGE arrangement
Byte 05	Byte 06	48bit [0x0506 0x0304 0x0102]
Byte 03	Byte 04	
Byte 01	Byte 02	
Byte 11	Byte 12	48bit [0x1112 0x0910 0x0708]
Byte 09	Byte 10	
Byte 07	Byte 08	
Byte 17	Byte 18	48bit [0x1718 0x1516 0x1314]

Table 1: Byte arrangement in output file

This arrangement of data comes very close to the placement of the data in the EPROM, where only the high and low byte must be swapped and separated into single bytes. For an in depth explanation of the boot record, please consult the Engineer- to- Engineer Note called “Tips & Tricks on the ADSP-2106x EPROM and HOST bootloader” (EE-56) available from Analog Devices.

Embedding UAB into FPM:

Having generated the User Application Boot image, the next step can be taken: including the UAB file into the ADSP-2106x program, which will take care of burning the data into the FLASH memory device. As defined before, it is called FLASH Programming Module (FPM).

Please note: depending on the ADSP-2106x you use, certain size restrictions in terms of lines in the UAB file apply. The following table (Table 2) shows the number of lines in the image before further action has to be applied. Furthermore, the equivalent EPROM image size is given. This size restriction is introduced due to the size of the internal memory bank 1 of the ADSP-2106x.

Processor	Max UAB lines	EPROM size
ADSP-21060	43690	2048 kbit
ADSP-21062	21845	1024 kbit
ADSP-21061	10922	512 kbit
ADSP-21065	5461	256 kbit

Table 2: Max. Lines of 16bit words in the image

When the generated file matches the stated requirements, the FPM project can be opened. This project contains a MAIN module, which picks the data out of the internal memory of the ADSP-2106x and exercises the FLASH server as discussed in EE-55. The Linker Description File (LDF) reserves the complete second memory block as 16bit wide data area to hold the UAB file.

Adaptations in the generic FPM:

The MAIN module of the FLASH Programming Module is very generic and needs only few changes to adapt to the user’s needs. These are:

#define f_size <NUMBER>

which holds the line number count in the UAB. Hint: During development, <NUMBER> could be set to the maximum possible lines as shown in Table 2 to insure that all possible lines are written, even if they are not filled by the boot image. As a trade off, it will take longer than necessary to finish the FPM.

The second change to be applied is patching in the FPM the correct file name for the UAB, so that the linker will generate a proper executable. The entry point is:

.var f_data[f_size] = "<NAME>";

Please fill in for <NAME> the filename and path to your User Application Boot image (UAB) as otherwise the linker complains for missing files. The MAIN module of the FPM can be found attached to this document as listing 1.

In- System- Programming:

Finally to kick-off the programming and a restart of your system running the new software out of the FLASH, open a v Debugger session connecting through the EZ-ICE to your hardware target and load the executable. Upon the completion of the download, press RUN and the UAB will be written to the FLASH memory device. When end of lines counter has decremented to “0”, a soft-reset is generated and the ADSP-2106x will reload its application code from the FLASH.

References:

v Documentation

EE-55: FLASH Application Note

EE-56: Tips & Tricks On Eprom / Host Booting

Technical Notes on using Analog Devices' DSP components and development tools
Phone: (800) ANALOG-D, FAX: (781) 461-3010, EMAIL: dsp.support@analog.com, FTP: ftp.analog.com, WEB: www.analog.com/dsp

```
/*****************************************************************************

  ANALOG DEVICES
  EUROPEAN DSP APPLICATIONS

  FLASH IT!

      will write application code from data segment to the FLASH

  History:
   1.0.1.0  18-JAN-98   HS
   1.0.1.1  21-JAN-98   HS

 *****************************************************************************/

/*****************************************************************************
  Global defines: change upon different codes
 *****************************************************************************/
#define f_size 1611           /* number of lines in blink.ldr */

/*****************************************************************************
  Global include files
 *****************************************************************************/
#include <def21060.h>
#include "flash.h"

/*****************************************************************************
  Interrupt vector table
 *****************************************************************************/
.section seg_rth;

      nop; nop; nop; nop;                /*        reserved */
      nop; jump start; rti; rti;         /* RSTI   reset vector */
      nop; nop; nop; nop;                /*        reserved */
      rti; rti; rti; rti;                /* SOVFI  stack overflow */
      rti; rti; rti; rti;                /* TMZHI  high timer */
      rti; rti; rti; rti;                /* VIRPTI vector interrupt */
      rti; rti; rti; rti;                /* IRQ2I  irq2 vector */
      rti; rti; rti; rti;                /* IRQ1I  irq1 vector */
      rti; rti; rti; rti;                /* IRQ0I  irq0 vector */
      nop; nop; nop; nop;
      rti; rti; rti; rti;                /* SPR0I  DMA0 SP0_RX */
      rti; rti; rti; rti;                /* SPR1I  DMA1 SP1_RX */
      rti; rti; rti; rti;                /* SPT0I  DMA2 SP0_TX */
      rti; rti; rti; rti;                /* SPT1I  DMA3 SP1_TX */
      rti; rti; rti; rti;                /* LP2I   DMA4 link buffer 2 */
      rti; rti; rti; rti;                /* LP3I   DMA5 link buffer 3 */
      rti; rti; rti; rti;                /* EP0I   DMA6 ext.port buf 0 */
      rti; rti; rti; rti;                /* EP1I   DMA7 ext.port buf 1 */
      rti; rti; rti; rti;                /* EP2I   DMA8 ext.port buf 2 */
      rti; rti; rti; rti;                /* EP3I   DMA9 ext.port buf 3 */
      rti; rti; rti; rti;                /* LSRQ   link port service req */
      rti; rti; rti; rti;                /* CB7I   circ buffer 7 overflow */
      rti; rti; rti; rti;                /* CB15I  circ buffer 15 overflow */
      rti; rti; rti; rti;                /* TMZLI  low timer */
      rti; rti; rti; rti;                /* FIXI   fixed overflow */
```

```
      rti; rti; rti; rti;                /* FLTOI  floating overflow */
      rti; rti; rti; rti;                /* FLTUI  floating underflow */
      rti; rti; rti; rti;                /* FLTII  floating invalid exception */
      rti; rti; rti; rti;                /* SFT0I  software irq 0 */
      rti; rti; rti; rti;                /* SFT1I  software irq 1 */
      rti; rti; rti; rti;                /* SFT2I  software irq 2 */
      rti; rti; rti; rti;                /* SFT3I  software irq 3 */


/*************************************************************************
  File to be flashed out
 *************************************************************************/
.section seg_fout;
.var f_data[f_size] = "blink.ldr";



/*************************************************************************
  Flash Code Section
 *************************************************************************/
.section seg_pmco;

start:      irptl = 0;         /* clear any pending interrupts */
      nop;
      bit set mode2 FLG2O | FLG1O | FLG0O | FLG3O;
      bit clr astat FLG3 | FLG2 | FLG1 | FLG0;

      r1 = 0xb000;             /* ADSP-21061: adjusting MSIZE */
      r0 = dm(SYSCON);
      r0 = r0 OR r1;
      dm(SYSCON) = r0;

      call flash_setup;        /* flash setup */

      r0 = 0x0;                /* erase sector 0 */
      dm(address) = r0;
      call sect_erase;
                               /* CAVEAT:
                                *
                                * depending on code size,
                                * multiple sectors in the
                                * FLASH must be erased;
                                * easier but lengthly:
                                *   CHIP_ERASE()

 */
restart:

      i0 = f_data;             /* pointer to data */
      l0 = 0x0;


      r2 = 0x0;                /* set address to 0 */


      r0 = dm(i0,1);           /* fetch new 16bit */
      r1 = fext r0 by 0:8;     /* get lower 8 bits */

/*************************************************************************
```

```
  Flash Programming Loop
 ************************************************************************/
      lcntr = f_size, do prog_loop until lce;

                  dm(d_byte) = r1;
                  dm(address) = r2;
                  call prog_byte;
                  r2 = r2 + 1;

                  r1 = fext r0 by 8:8;    /* get higher 8 bits */
                  dm(d_byte) = r1;
                  dm(address) = r2;
                  call prog_byte;         /* may not be placed in last */
                                       /* three cycles of a loop    */
                  r2 = r2 + 1;
                  r0 = dm(i0,1);          /* fetch new 16bit */
      prog_loop:  r1 = fext r0 by 0:8;    /* get lower 8 bits */

done: r0=0x1;
      dm(SYSCON)=r0;                      /* force reboot from EPROM */
```

Listing 1: FPM Flash Programming Module

```
ARCHITECTURE(ADSP-21061)
SEARCH_DIR( $ADI_DSP\21k\lib )
$LIBRARIES = lib060.dlb, libc.dlb;
$OBJECTS = $COMMAND_LINE_OBJECTS;

MEMORY
{     seg_rth  { TYPE(PM RAM) START(0x00020000) END(0x0002007f) WIDTH(48) }
      seg_pmco { TYPE(PM RAM) START(0x00020080) END(0x00020fff) WIDTH(48) }
      seg_dmda { TYPE(DM RAM) START(0x00021000) END(0x00021fff) WIDTH(32) }

      seg_fout { TYPE(DM RAM) START(0x00048000) END(0x0004ffff) WIDTH(16) }
}
PROCESSOR p0
{     LINK_AGAINST( $COMMAND_LINE_LINK_AGAINST)
      OUTPUT( $COMMAND_LINE_OUTPUT_FILE )
      SECTIONS
      {     seg_rth {
                  INPUT_SECTIONS( $OBJECTS(seg_rth) $LIBRARIES(seg_rth))
            } >seg_rth
            seg_pmco {
                  INPUT_SECTIONS( $OBJECTS(seg_pmco) $LIBRARIES(seg_pmco))
            } >seg_pmco
            seg_dmda {
                  INPUT_SECTIONS( $OBJECTS(seg_dmda) $LIBRARIES(seg_dmda))
            } > seg_dmda
            seg_fout {
                  INPUT_SECTIONS( $OBJECTS(seg_fout) $LIBRARIES(seg_fout))
            } > seg_fout
      }
}
```

Listing 2: LDF File declaring 16bit space to hold UAB data in the seg_fout section