Notes on using Analog Devices' DSP, audio, & video components from the Computer Products Division
Phone: (800) ANALOG-D or (781) 461-3881, FAX: (718) 461-3010, EMAIL: dsp.support@analog.com

Interfacing Byte Programmed Flash Memories to the ADSP-2106x SHARC series

Contributed by S.H. *20-Jul-99*

Introduction:

Until recently, an EPROM was chosen as standard external boot memory to start up the processor in the system. However, if you needed to change the data of your EPROM, you had unseal your product, remove the EPROM from the system, erase it by exposing it to hard UV-light, reprogram it using a dedicated EPROM burner. Finally, it could be fitted back into the device.

As this is a complicated and time consuming procedure, a better solution is to update the code or data while the non-volatile memory component remains in the system. This is a key issue for todays embedded designs, where system specifications and requirements are continuously updated, especially in systems, where you have to save application data in a simple and efficient fashion during runtime.

With the advent of FLASH memories, it is now possible to save data permanently and update it when necessary without removing the component from the system. FLASH memories are also an asset in systems that need to save data during a power outage or brownout. The DSP can store its code/data contents from volatile internal memory to an external, non-volatile memory, and on revival of the system, rewrite the old information back to the DSP.

This Engineering Note demonstrates how to interface a FLASH memory to the ADSP-2106x SHARC series using a spare memory segment for FLASH writes. The supplied code shows you how to access FLASH devices, using AMD NOR FLASHs as example components, and how to perform the standard operations like read, write, erase and system identification.

FLASH Wiring:

As the set-up for data unpacking and handling the unlock sequences is time consuming and complicated using DMA write operations, the update of the FLASH is handled by the core processor. Therefore to allow core reads and writes and to access the FLASH during system boot, it is necessary to connect logically the boot memory strobe (~BMS) and one of the four available memory strobes (~MSx) for the core accesses. The wiring for 8bit wide FLASH memories and the ADSP-21060/ 61/ 62 can be found in Figure 1, while Figure 2 is for the connection using an ADSP-21065L. Please note that the amount of address lines required is dependent on the size of the FLASH.

DSP	FLASH
A[0..A20*]	A[0..A20*]
D[16..23]	D[0..7]
~BMS & ~MSx	~CE
~WR	~WE
~RD	~OE

Figure 1. ADSP-21060 / 61 / 62 -> FLASH Wiring

DSP	FLASH
A[0..A20*]	A[0..A20*]
D[0..7]	D[0..7]
~BMS & ~MSx	~CE
~WR	~WE
~RD	~OE

Figure 2. ADSP-21065L -> FLASH Wiring

During system boot, the ADSP sees the FLASH as an EPROM and uses its standard DMA booting mechanism by asserting ~BMS and data packing from 8bit to 48bit.

Basic AMD NOR FLASH Operation:

The AMD FLASH memory space is splitted into a several sectors, each sector can be protected individually. This allows to have an emergency set-up if the previous program code exchange failed for some reason. Additionally this gives safety for applications

where the FLASH is used for both, system booting and data written to it during program execution. The sector boundaries are invisible for read / write, resulting is seamless operation across sector boundaries, easing the use of the FLASH.

The AMD FLASH uses a byte-by-byte programming protocol, typical for NOR FLASH types. A series of command words are written to the FLASH (using ADSP-2106x core writes in this example), essentially "unlocking" the device so the proper information can be stored or retrieved.

The following operations, called Embedded Programming Algorithms, can be performed on an AMD FLASH:

Autoselect:
Returns the manufacturer, model number, and the protection status of any sector.

Byte Write:
Programs a single data word (8 or 16bit wide) into an unprotected FLASH sector.

Sector Erase:
Erases a sector of unprotected memory.

Chip Erase:
Erases all unprotected sectors on a FLASH.

Sector Protect:
Returns status of sector protection bit for the requested sector on the FLASH.

Please consult the appropriate AMD Data Sheet for more memory programming information and for 16bit wide I/O operations.

FLASH access using the external port:

All transactions to memory mapped peripherals are handled by the 32bit wide external port of the ADSP-2106x SHARC. As the FLASH memory is only 8bits or 16bits wide, only the lower data lines will carry relevant data. Unused data lines will present the last value the bus was driven to and have to be masked out. Additionally the ~RD or ~WR line are asserted and the corresponding memory strobe ~MSx.

The ADSP-21065L offers fixed external memory blocks, each 16Mwords large, while the ADSP-21060/ 61/ 62 has configurable banks, defaulting to 8kwords. Keep in mind, to assert the proper ~MSx signal, it is important to set the MSIZE bits in the SYSCON register of the IOP processor accordingly, so that the full addressable area of FLASH memory fits into a single external memory bank.

Flash Server Definitions:

Attached to this paper are a number of software modules used for accessing and controlling i.e. the AMD29LV010, AMD29LV020, AMD20LV040, and AMD29F040 FLASH memories, just to name few devices. This software offers six entry points in the server code that are function names for FLASH operations. Before using the functions, some global definitions (#define *LABEL*), placed in the software header, have to be set up to match your FLASH component. These are:

mem_offset:
0x400000 /* default for 21060/61/62 */
This value is the offset to the external memory, so that the memory corresponding strobe ~MSx can be asserted by the core processor. Permissible values for the ADSP-21065L are 0x20000, 0x1000000, 0x2000000 or 0x3000000.

memory_bank:
0x0 /* memory bank 0, ~MS0 */
Needs to be set for setting the wait states in the WAIT register. Permissible values range from 0x0 to 0x3.

To successfully start the embedded programming algorithms, an unlock sequence, a combination of address and data values has to be written first to the FLASH. *Ulock1_a*(ddress) and *Ulock1_b*(yte) represent the first combination of address- and data word, while *Ulock2_a*(ddress) and *Ulock2_b*(byte) are the second pair. *Ulock1* and *Ulock2* can be changed depending on the FLASH type and connection (8/16bit). Please consult the AMD FLASH memory datasheet for the applicable listing of unlock address / word combinations.

ulock1_a:
0x5555 /* default for 8bit wide */

ulock1_b:
0x00AA /* default for 8bit wide */
ulock2_a:
0x2AAA /* default for 8bit wide */
ulock2_b:
0x0055 /* default for 8bit wide */

Flash Server Software:

These API functions provided by the FLASH software are listed and described below:

flash_setup:
Sets the number of wait states in external memory segment.

prog_byte:
Unlocks and writes the value of **d_byte** into the FLASH at **address**.

sect_erase:
Erases one sector of the FLASH starting at **address**.

flash_erase:
Erases the complete FLASH.

flash_ident:
Returns in **d_byte** the vendor code and device ID. Some FLASH components return in register **r12** the boot block location.

sect_protect:
Return information whether sector at **address** was protected during the programming cycle.

More information on each of these functions, including the parameters that are passed into each subroutine is included in the comments of the supplied code. As additional reference, please consult the FLASH memory datasheet.

To apply the FLASH server code, it must be assembled first and linked into your calling software. In order to make the flash server code "visible" to pre-existing code in other modules, the subroutines that will be used must be declared as external functions. This can be done by including the header file **flash.h** into your calling program.

After the flash server module (listing 1), a small example program is given (listings 2-4), showing how to implement the FLASH programming software. For more information on linking and calling functions located in separate modules, please refer to the *ADSP-21000 Family Assembler Tools and Simulator Manual*.

FLASH Caveats:

Byte Programming Approach:

You cannot correctly program a byte of data in the FLASH memory unless the contents of that memory cell first equals 0xFF. Essentially, a byte program only converts a logic '1' to a logic '0'. A sector or chip erase operation is the only operation that allows to set back a logic '0' to a logic '1'. Reprogramming a cell without erasing it may cause an error condition flagged by the FLASH server in the status variable **error**.

Additional Information:

For more information about interfacing ADSP-218x DSPs to various FLASH|es, please consult the following sources:

ADSP-2100 Family User's Manual
ADSP-2100 Family Assembler Tools and Simulator Manual
http://www.analog.com
http://www.amd.com

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

  ANALOG DEVICES
  EUROPEAN DSP APPLICATIONS

  FLASH API server for memory mapped AMD FLASH Devices

  History:
       1.0.1.0             15-JUN-99     HS

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

/*****************************************************************************
  Below stated defines represent the memory where the FLASH memory device is
  connected to and must be changed accordingly to reflect the actual device
  pleacement. Please consult the ADSP-2106x USER'S MANUAL to figure out the
  correct values for memory offset to the external memory bank and by what
  strobes the FLASH can be accessed.
 *****************************************************************************/

#define mem_offset   0x400000      /* offset to memory bank of FLASH */
#define memory_bank  0x2           /* selected memory bank */


/*****************************************************************************
  Ulock1_a and Ulock1_b represent the first combination of address and data
  required to access the embedded algorithm. Ulock2_a and Ulock2_b are the
  second pair of address and data to enter embedded algorithms. Ulock1_a and
  Ulock2_a can change depending on the FLASH type and connection (8/16bit).
  Please consult the AMD Flash memory datasheet for a complete listing.
 *****************************************************************************/
#define ulock1_a     0x5555
#define ulock1_b     0x00AA

#define ulock2_a     0x2AAA
#define ulock2_b     0x0055

#define u_mem1_a     ulock1_a + mem_offset
#define u_mem2_a     ulock2_a + mem_offset


/*****************************************************************************
  The global definition exports the functions and the labels to a calling
  main program.
 *****************************************************************************/
.global flash_setup;              /* entry to memory setup */
.global prog_byte;                /* entry to program a byte */
.global sect_erase;               /* entry to erase a sector */
```

```
.global flash_erase;              /* entry to erase complete FLASH */
.global flash_ident;              /* entry to identify FLASH */
.global sect_protect;                    /* entry to sector protect read */
.global address;                  /* user defined address for FLASH operation */
.global d_byte;                   /* data byte to be written to FLASH */
.global error;                           /* result register for FLASH operation */

#include <def21060.h>


/*****************************************************************************
  Declaration of few memory cells required for accessing the API
 *****************************************************************************/
.segment/dm seg_dmda;
.var address;
.var d_byte;
.var error;
.endseg;


/*****************************************************************************
  Program code of the API fuctions
 *****************************************************************************/
.segment/pm seg_pmco;


/*****************************************************************************
 *
 * Global setup for the mapped meory area
 *
 * REGISTER usage summary:
 *
 * input   : none
 * modify  : WAIT (0x2) register of IOP area
 * output  : none
 * destroy : r12, r13, r14, r15
 * calls   : none
 *
 *****************************************************************************/
flash_setup:
       r12 = memory_bank;         /* get assigned memory bank */
       r13 = 0x5;
       r12 = r12 * r13 (UUI);            /* compute position */

       r13 = 0x1f;                /* compute the mask out */
       r13 = lshift r13 by r12;   /* shift it to memory bank */
       r13 = NOT r13;                    /* and invert it */

       r14 = 0x16;                /* set-up of memory wait states and mode */
       r14 = lshift r14 by r12;   /* shift it into position */

       r15 = dm(WAIT);
       r15 = r15 AND r13;
       r15 = r15 OR r14;
       dm(WAIT) = r15;            /* adapt the settings */

       rts;
```

```
/*****************************************************************************
 *
 * FLASH Application Server
 *
 *   API: program byte to FLASH memory
 *
 *     d_byte  : value to be written
 *     address : destination address of value
 *
 * REGISTER usage summary:
 *
 * input   : none
 * modify  : error flagged with dm(error) <> 0 for error condition
 * output  : none
 * destroy : r12, i4, m4
 * calls   : init_seq, DQ7_poll
 *
 *****************************************************************************/
prog_byte:
       call init_seq;

       r12 = 0xA0;                /* load command byte */
       dm(u_mem1_a) = r12;        /* write command byte to FLASH */

       i4 = dm(address);          /* load user address */
       m4 = mem_offset;           /* load memory offset */
       r12 = dm(d_byte);          /* read byte to program */
       dm(m4,i4) = r12;           /* perform the access */

       call DQ7_poll;                   /* check for end of sequence */

       rts (db);
       nop;
       nop;









/*****************************************************************************
 *
 * FLASH Application Server
 *
 *   API: erase specified sector of FLASH memory
 *
 *     address : destination address of sector to erase
 *
 * REGISTER usage summary:
 *
 * input   : none
 * modify  : error flagged with dm(error) <> 0 for error condition
 * output  : none;
 * destroy : r12, i4, m4
 * calls   : init_seq, DQ7_poll
 *
```

```
 ***************************************************************************/
sect_erase:
       call init_seq;

       call init_seq(db);
       r12 = 0x80;                 /* load first command byte */
       dm(u_mem1_a) = r12;         /* write command byte to FLASH */

       i4 = dm(address);           /* load user address */
       m4 = mem_offset;            /* load memory offset */
       r12 = 0x30;                 /* load second command byte */
       dm(m4,i4) = r12;            /* perform the access */

       call DQ7_poll;                    /* check for end of sequence */

       rts (db);
       nop;
       nop;



/***************************************************************************
 *
 * FLASH Application Server
 *
 *   API: erase complete FLASH memory
 *
 * REGISTER usage summary:
 *
 * input   : none
 * modify  : error flagged with dm(error) <> 0 for error condition
 * output  : none
 * destroy : r12
 * calls   : init_seq, DQ7_poll
 *
 ***************************************************************************/
flash_erase:
       call init_seq;

       call init_seq (db);
       r12 = 0x80;                 /* load first command byte */
       dm(u_mem1_a) = r12;         /* write command byte to FLASH */

       r12 = 0x10;                 /* load second command byte */
       dm(u_mem1_a) = r12;         /* perform the access */

       call DQ7_poll;                    /* check for end of sequence */

       rts (db);
       nop;
       nop;
```

```
/*****************************************************************************
 *
 * FLASH Application Server
 *
 *   API: identify connected FLASH memory
 *
 *     d_byte : holds readback of device ID
 *     r12    : holds readback of bootblock location
 *
 * REGISTER usage summary:
 *
 * input   : none
 * modify  : none
 * output  : d_byte, r12
 * destroy : r12, i4
 * calls   : init_seq
 *
 *****************************************************************************/
flash_ident:
       call init_seq;

       r12 = 0x90;                /* load command byte */
       dm(u_mem1_a) = r12;        /* write command byte to FLASH */

       i4 = mem_offset;
       r12 = dm(i4,2);            /* read back manufacturer ID */

       dm(d_byte)=r12;
       r12 = dm(i4,2);            /* read back boot block location */

       rts(db);
       r13 = 0xF0;                /* reset FLASH memory */
       dm(0,i4) = r13;


/*****************************************************************************
 *
 * FLASH Application Server
 *
 *   API: check for hard sector protection
 *
 *     address : sector to be interrogated
 *     d_byte  : holds readback of sector protection
 *
 * REGISTER usage summary:
 *
 * input   : none
 * modify  : none
 * output  : none
 * destroy : r12, r13, i4, m4
 * calls   : init_seq
 *
 *****************************************************************************/
sect_protect:
       call init_seq;

       r12 = 0x90;                /* load command byte */
       dm(u_mem1_a) = r12;        /* write command byte to FLASH */
```

```
       i4 = mem_offset;
       m4 = dm(address);

       r12 = dm(m4, i4);          /* read back protect info */
       dm(d_byte)=r12;

       rts(db);
       r13 = 0xF0;                /* reset FLASH memory */
       dm(0,i4) = r13;




/*****************************************************************************
 *
 * FLASH Application Server
 *
 *   SUB init sequence
 *
 * REGISTER usage summary:
 *
 * input   : none
 * modify  : none
 * output  : none
 * destroy : r12
 * calls   : none
 *
 *****************************************************************************/
init_seq:
       r12 = ulock1_b;            /* load first unlock byte */
       dm(u_mem1_a) = r12;        /* write first unlock byte to FLASH */

       rts(db);                   /* return to calling module */
       r12 = ulock2_b;            /* load second unlock byte */
       dm(u_mem2_a) = r12;        /* write second unlock byte to FLASH */


/*****************************************************************************
 *
 * FLASH Application Server
 *
 *   SUB poll end of algorithm
 *
 * REGISTER usage summary:
 *
 * input   : none
 * modify  : ASTAT
 * output  : none
 * destroy : r12, r13, r14, i4, m4
 * calls   : none
 *
 *****************************************************************************/
DQ7_poll:
       r13 = dm(m4,i4);           /* first read */
       r14 = dm(m4,i4);           /* second read */
```

```
       comp(r13,r14);                    /* equal ? */
       if eq jump (pc,DQ7_exit), r12=r12-r12;   /* no toggle, exit */

       btst r14 by 0x5;           /* timeout ? */
       if sz jump (pc,DQ7_poll);  /* no, re-read status */

       r13 = dm(m4,i4);           /* first read */
       r14 = dm(m4,i4);           /* second read */

       comp(r13,r14);                    /* equal ? */
       if eq jump (pc,DQ7_exit), r12=r12-r12;   /* yes, exit */

       btst r14 by 0x5;           /* timeout ? */
       if sz jump (pc,DQ7_exit), r12=r12-r12;   /* no, exit */

       r12 = 0xF0;                /* reset FLASH */
       dm(m4,i4) = r12;

DQ7_exit:
       rts (db);
       dm (error) = r12;
       nop;

.endseg;
```

Listing 1: ADSP-2106x FLASH server software

```
/*****************************************************************************
 ANALOG DEVICES
 EUROPEAN DSP APPLICATIONS

 FLASH API server for memory mapped AMD FLASH Devices

 History:
       1.0.1.0             15-JUN-99     HS

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

.extern flash_setup;              /* entry to memory setup */
.extern prog_byte;                /* entry to program a byte */
.extern sect_erase;               /* entry to erase a sector */
.extern flash_erase;              /* entry to erase complete FLASH */
.extern flash_ident;              /* entry to identify FLASH */
.extern sect_protect;                    /* entry to sector protect read */

.extern address;                  /* user defined address for FLASH operation */
.extern d_byte;                          /* data byte to be written to FLASH */
.extern error;                           /* result register for FLASH operation */
```

Listing 2: flash.h include file

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

  ANALOG DEVICES
  EUROPEAN DSP APPLICATIONS

  Main Module calling FLASH API server for memory mapped AMD FLASH Devices

  History:
       1.0.1.0             15-JUN-99     HS

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

/*_________________interrupt vector table_____________*/
.segment/pm 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 */

.endseg;

/*___________________Main Program____________________ */
.segment/pm   seg_pmco;

start: irptl = 0;                /* clear any pending interrupts */
       nop;
```

```
       r1 = 0xb000;                       /* set MSIZE bits in SYSCON */
       r0 = dm(SYSCON);
       r0 = r0 OR r1;
       dm(SYSCON) = r0;

       call flash_setup;                  /* test case #1 setup */

       r0 = 0x0;                          /* erase sector 0 */
       dm(address) = r0;
       call sect_erase;

       r0 = 0x0;                          /* write first 256 location */
       dm(address) = r0;
       dm(d_byte) = r0;


       lcntr = 0x100, do prog_loop until lce;
                           call prog_byte;
                           r0 = r0 + 1;
                           dm(address) = r0;
       prog_loop:    dm(d_byte) = r0;

stop:
       jump stop;                         /* test completed */

.ENDSEG;
```

Listing 3: ADSP-2106x calling program

```
asm21k flash -adsp21060 -l
asm21k main -adsp21060 -l
ld21k main flash -a 061_prz.ach
```

Listing 4: ADSP-2106x calling program