Technical notes on using Analog Devices DSPs, processors and development tools
Contact our technical support at dsp.support@analog.com and at dsptools.support@analog.com
Or visit our on-line resources http://www.analog.com/ee-notes and http://www.analog.com/processors

Tips & Tricks on the ADSP-2106x SHARC® EPROM and Host Bootloader

Contributed by Stefan Hacker

Rev 2 – March 24, 2004

Introduction

After creating ADSP-2106x SHARC® processor application code and verifying it in simulation or emulation, one of final tasks is creating a boot image for an EPROM or host processor. Though this is an easy task using the VisualDSP++® tools set, some users want to add specific functionality to the boot loader or created their own. This document discusses the output of the elfloader tool and the arrangement of the data in the boot image. It shows you where to add changes to modify the boot loaders to adapt to different host processor widths or types.

Boot Loading

By default, the ADSP-2106x SHARC processor is configured to load 256 words of 48-bit width (instruction size) after reset by DMA. DMA channel 6 is initialized for the ADSP-21060/ 61/ 62 and DMA channel 8 is initialized for the ADSP-21065L. Both DMA channels have in common the 0x40 offset in the interrupt vector table of the ADSP-2106x. The DMA settings for EPROM booting are shown in Table 1.

Numbers printed in italics are not initialized but are assumed by the DMA engine.

Since user application code is far larger than words, a boot kernel ensures that the complete user application code and data is loaded into the internal memory spaces of the SHARC. Additionally, external memories and data ports are initialized by the kernel.

Boot Loader Names

060_prom.asm and 065l_prom.asm are the source files for the ADSP-2106x PROM-based loaders. 060_host.asm and 065l_host.asm are the source files for the ADSP-2106x host-based boot loader.

		21060/ 61/ 62	21065L
Boot Space		4M x 8-bit	8M x 8-bit
DMAC Register		DMAC6=0x2A1	DMAC0=0x2A1
II6	IIEP0	0x20000	0x8000
IM6	IMEP0	*0x1*	*0x1*
C6	CEP0	0x100	0x100
EI6*	EIEP0*	0x40 0000	0x80 0000
EM6	EMEP0	*0x1*	*0x1*
EC6	ECEP0	0x600	0x600
IRQ Vector		0x20040	0x8040

Table1: EPROM boot setting with BSO bit set

* BMS space

Depending on the selected processor and the boot loading type, one of the four kernels is loaded into seg_ldr at reset. The corresponding architecture file is either named 060_ldr.ldf or 065l_ldr.ldf. These source files are located in the \21K\LDR subdirectory within the VisualDSP++ tools set.

Boot Kernel Structure

Each kernel begins by defining macros for the various IOP registers used in the code, followed by a table containing the interrupt vectors, up to and including the DMA interrupt aligned to the external port buffer 0 (EP0I). Most of these high-priority interrupts are not necessary for the operation of the boot loader and are filled with

NOP or RTI instructions. The only interrupts used by the kernel are the reset interrupt ___lib_RSTI (offset 0x04-0x07) and the EP0I interrupt ___lib_EP0I (offset 0x40-0x43).

Beginning from the start_loader label the kernel code can be divided into four main sections:

Start_loader Initializes some required registers and determines for EPROM booting the processor ID of the SHARC in a multiprocessor environment.

Load_memory Starts to parse the information from the boot source and copies it into the required memory location or clears not preplaced memory segments.

Final_init Swaps out the boot kernel and replaces it with the user application code.

read_PROM_word Sets up the new DMA transfer to collect a new 48-bit word.

EPROM Boot Kernel Operation

After reset, the core processor is held in an IDLE mode until the first 256 words, each 48-bits wide, have been loaded by DMA into the internal memory. The External Port DMA interrupt for EP0I is activated upon the completion of the DMA transfer. The core processor will start execution of the just-loaded boot kernel by branching to the vector interrupt location for EP0I.

In ___lib_EP0I the DMA control register setting is stored in R2 for later restoration and the DMA channel is disabled temporarily by clearing the DMA enable bit in the control register. Having completed the IRQ service, the core processor starts up the loader program.

Beginning from Start_Loader some required registers are initialized. This is be a good place to start up external memories like SDRAM on the ADSP-21065L or to set wait states and wait modes.

/BMS is deactivated and normal external memory selects are activated by clearing the BSO -bit in the SYSCON register. Three copies of SYSCON are used in the program one that contains the original value of SYSCON, one that contains SYSCON with the BSO-bit set so that an ADSP-2106x can gain access to the boot EPROM, and a third with the BSO bit cleared. When BSO=1, the EPROM packing mode -bits in the DMACx control register are ignored and 8-to-48-bit packing is forced. For the ADSP-21065L a 32-bit-wide system bus is assumed. (Note that 8-to-48-bit packing is available only on the ADSP-2106x during DMA reads from /BMS space with the BSO bit set).

When one of the external port DMA channels is being used in conjunction with the BSO bit, none of the other three channels may be used. When BSO=1, /BMS is not asserted by a core processor access, only during a DMA transfer. This allows your bootstrap program (run by the ADSP-2106x core) to perform other external accesses to non-boot memory.

The IMASK register is set finally to allow the EP0I interrupt and the MODE1 register is set to enable interrupts and nesting.

Having completed the setup, the DMA engine on the ADSP-2106x processors is used to collect 48-bit words from the EPROM. As an external boot EPROM allows starting a complete multiprocessor cluster, the proper section in the EPROM must be determined by checking the processor ID in the SYSTAT register. The code beginning from get_addr label will parses a seven entry 48-bit table stored in the EPROM (hex offset 0x600 = 6*0x100 = 256 instruction words) to find start address of boot section for this processor. Every entry of the table is formatted as:

address(32 -bit) processor ID (16 -bit)

As an example the readback 0x8002062A0001 for an ADSP-21065L translates into an EPROM offset of 0x8002062A and processor ID `0001`.

Having determined the offset, the DMAC6/DMAC0 control register is set to 0x2A1 and DMA parameters are set up to read data word-by-word

beginning from the starting address of the boot section corresponding to the processor ID in the boot EPROM. Each 48-bit word is transferred into address 0x20004 / 0x8004 for dispatching. Because the image in the EPROM contains program memory code sections and data memory sections with different sizes, a preamble is stored before each boot block. The preamble with the attached boot block is formatted as shown in Table 2.

0x0000 0000 DDDD	D (data type tag)
0xAAAA AAAA LLLL	A (address), L (length)
0xBOOT BOOT BOOT	Boot data
:	:
0xBOOT BOOT BOOT	Boot data

Table 2. Boot Section Header

Each initialization block is identified by a 16-bit tag placed before the block. Each type of initialization has a unique tag number.

Tag Number		Initialization Type
0	0x0	FINAL INIT
1	0x1	ZERO DM16
2	0x2	ZERO DM32
3	0x3	ZERO DM40
4	0x4	INIT DM16
5	0x5	INIT DM32
6	0x6	INIT DM40
7	0x7	ZERO PM16
8	0x8	ZERO PM32
9	0x9	ZERO PM40
10	0xA	ZERO PM48
11	0xB	INIT PM16
12	0xC	INIT PM32
13	0xD	INIT PM40
14	0xE	INIT PM48

Table 3. Section Header Types

The boot kernel initializes internal and external memories by reading the data from EPROM using a routine called Read_Prom_Word and writing it to a specific location of memory (0x20004 / 0x8004). For a zero-valued format data block whose tag is 1, 2, 3, 7, 8, 9, or 10, an initialization of 16- or 32-bit memory is done in a loop, which writes a zero value to memory, reducing the required space in the EPROM.

Any initialization of 40- or 48-bit PM memory uses a write with the PX register set to zero. For a non-zero format data block whose tag is 4, 5, 6, 11, 12, 13, or 14 the kernel enters a loop which reads one 48-bit word from EPROM and writes the appropriate width value to memory. This loop is repeated once for each word being initialized.

When the boot loader has completed parsing a boot block, it continues with the next tag and executes the appropriate initialization routine until the kernel reaches the FINAL_INIT (0x0) boot tag.

In the final initialization stage, the kernel loads the first 256 words of the target executable file and will overwrites itself. When the loader detects the tag, it reads the next 48-bit word. This word indicates the instruction to be located at 0x20040 / 0x8004 when the loading is close to being completed. This instruction is saved into the 48-bit PX register so that the boot loader can now finish initializing internal memory. The kernel requires an RTI instruction at address 0x020040 / 0x8040 which is temporarily placed, because an EP0 interrupt is generated when the initialization is completed. The R9 register is loaded with 0xbdb0000 containing the encoded instruction PM(0,I8)=PX. This writes the desired customer instruction over the RTI used by the boot kernel with I8 pointing to 0x20040 / 0x8040.

Before the DMA sequence is initiated, the core processor is trapped in a pseudo loop by issuing

```
DO ___lib_RSTI UNTIL EQ;
FLUSH CACHE;
R0=0x20004;  /* 0x8004 on '65l */
PCSTK=R0;
<DMA init>   /* some code here */
IDLE;
```

and manually adding the return address 0x20004 / 0x8004 on the stack. The loop terminates on an equal condition. Because the code will be overwritten by the DMA sequence, it is necessary to invalidate the cache with a FLUSH CACHE instruction.

The last 256 48-bit words are loaded into memory over the boot loader while the core processor is idling. Upon completion, the RTI is executed at address 0x20040 / 0x8040, returning the core processor to address 0x20004 / 0x8004, so that the next instruction to be carried out is filled with following instruction line:

```
R0=R0-R0,DM(I4,M5)=R9,PM(I12,M13)=R11
```

which is read from the EPROM. This instruction clears the loop condition (r0=r0-r0) puts PM(0,I8)=PX (held in R9) into 0x20004 / 0x8004, and sets SYSCON back to the original value. At loop termination of the loop, the program sequencer is set back to 0x20004 / 0x8004. The PX write exchanges the previously placed RTI at 0x20040 / 0x8040 with the user instruction and then proceeds to program location 0x20005 / 0x8005 which should be the beginning of user application code.

For easier understanding of the data placement in the EPROM, its image is parsed and is included in this document.

Host Boot Kernel Operation

In many ways, the host boot kernel works like the EPROM boot kernel. This section outlines only the differences between them.

Host booting uses slave DMA instead of PROM booting's master DMA.

		21060/ 61/ 62	21065L
Host Timing		synchr./asynchr.	asynchr.
SYSCON		0x10	0x20
DMAC Register		DMAC6=0xA1	DMAC0=0xA1
II6	IIEP0	0x20000	0x8000
IM6	IMEP0	*0x1*	*0x1*
C6	CEP0	0x100	0x100
IRQ Vector		0x20040	0x8040

Table 4. Host Boot Setting

At first it is important to verify that the packing HPM bits in SYSCON and DMAC6 supports the 16-48 packing mode (HBW bits and DMAC0 8-48 packing ADSP-21065L) as this is the default mode. If this not selected, the first write of the host processor has to change the settings of SYSCON, otherwise the generic boot loader will not work.

As soon as this first adaptation has been made or is verified, the host starts writing the first 256 instruction words as packed data to the external port buffer 0 of the I/O processor at offset location 0x4. This may be done in one host bus request cycle, where the /HBR pin (synchronous) or /HBR and /CS pins (asynchronous) of the selected processor must be driven low. The host interface of the slave responds with /HBG and ACK pins (synchronous) or /HBG and REDY pins (asynchronous) to recombine the data words to instructions and places them beginning in 0x20000 / 0x8000 in internal memory.

Having written the first 256 instruction words, the slave's DMA internal Cx register elapses and the DSP wakes up and starts executing the boot kernel beginning from ___lib_EP0I, (DMA interrupt vector), immediately turning off the DMA channel by setting DEN=0 and locking the external bus with BUSLK bit in MODE2. If data/code is to be placed externally, the host processor must give up bus mastership or a deadlock will occur. The ADSP-2106x cannot drive external signals and cannot parse new data presented by the host processor.

Beginning from this point timing of the host processor is essential. Now the ADSP-2106x expects now single-instruction word size slave

DMA sequences in which the user is presenting again three 16-bit-wide (six 8-bit-wide ADSP-21065L) wide data chunks on EPB0 (0x04). These words form a 48-bit-wide instruction word which is placed into 0x20004 / 0x8004 and is then parsed. So the user just continues writing data to EPB0. A change in the IOP destination address is not necessary.

If the host continues writing data to buffer 0, the EPB0 FIFO and slave write FIFO will fill up and REDY (asynchronous) will be de-asserted. This is the handshake signal to the host processor to extend further accesses.

The structure of the boot image is quite similar to the EPROM boot structure; the only difference is the missing multiprocessor boot table after the boot kernel. A host may boot a multiprocessor system by selecting multiple /CS pins asynchronously or directly in MMS space (synchronously).

Note: If large arrays must be initialized in external memory, quite some time may be required until the ADSP-2106x returns bus control back to the host processor. If such waiting periods result in time-out on the host, you can specify the time-out switch of the elfloader to break these initializations into smaller pieces allowing the host processor to obtain bus control earlier.

Having downloaded the initialization data, the last 256 instruction words may be written again in a single access to EPB0, as this only replaces code which is placed internally. The host boot loader swapping mechanism is identical to the EPROM boot loading sequence.

Boot Kernel Caveats

The kernel assumes that IMDW is 0 during the booting process before it is set to 1 in the final boot stage of the kernel. Also remember that when using any of the power-up booting modes, location 0x20004 / 0x8004 must not contain a valid instruction since it is not executable during the booting sequence. Place a NOP or IDLE instruction at this location.

If the kernel is going to initialize external memory, ensure that the appropriate values are set in SYSCON and WAIT register and that they are correct, otherwise the processor may hang.

Note that SDRAM (ADSP-21065L) needs a power-up routine before it is accessible. This is reached by placing the init code into the loader kernel.

Be aware that the value in DMACx is non-zero and that the IMASK is set to allow DMACx interrupts. Because the EP0I interrupt remains enabled in IMASK, it must be cleared before this DMA channel may be used again. Otherwise, unintended interrupts may occur. Additionally, reset DMACx to 0x0 before reinitializing or a new DMA sequence may not start.

User Changes to Boot Loader Sources

The EPROM Boot loader does not allow too many changes, as during the first 256 words instruction load a packing mode of 8-to-48 is forced with the BSO bit. So if the user places a 16-bit wide EPROM, the first 256 instructions must be spread across the first 0x600 addresses of the EPROM showing only the lowest eight-bits populated.

Changing the DMA channel to a higher number and its initialization sequence allows the use of different packaging modes. The BSO bit is required, as otherwise no /BMS memory strobe is generated. With these modifications, the user could boot from a 16-bit wide EPROM instead of an 8-bit wide EPROM.

The host boot loader offers more options: the packing mode in SYSCON and DMACx can be changed so that different bus widths (16 or 32 bits) are possible. You must ensure proper ordering of the data words to be written over

EPB0, or the initialization of the processor will fail.

Appendix

EPROM boot image Example

EPROM boot image decoded and partitioned, outlining important parts, broken into 48-bit words:

:020000	04	0000 boot record start	FA
:200000	00	000000000000 000000000000 000000000000 000000000000 000000008000 4480 +- start of boot kernel	9C
:200020	00	00043E06 000000000000 000000000000 000000000000 000000000000 00000000	78
:200040	00	0000 000000000000 000000003E0B 000000003E0B 000000003E0B 000000003E0B	7C
..	..	..	..
:2005E0	00	0000 000000000000 000000000000 000000000000 000000000000 000000000000 end of boot kernel -+	FB
:200600	00	00002A060080 010000000000 020000000000 030000000000 040000000000 0500 +- 8000062A0000 = offset of 0x8000062A in EPROM for processor ID=0	1B
:0A0620	00	00000000 060000000000	CA
:20062A	00	0E0000000000 0E0000810000 00000000790F 000000000000 008007000A14 0000 ! ! +- start of program code ! +- start address 0x8100, length 0x0E = 14_{dez} +- section tag INIT_PM48	E6
:20064A	00	38002C14 000040000C14 001BB700DF0F 001BB700DE0F 200000000A14 00100000	EB
:20066A	00	0B14 100000000D14 000000008000 0A8100003E06 000060004C14 000000003E0B	C8
:20068A	00	000000000000 000000003E0B 000000000000 000000000000 000000000000 0000 ! ! +- start of user RTH ! +- instruction to be placed at 0x8040 +- section tag FINAL_INIT	07
:2006AA	00	00000000 0020802D7339 008100003E06 000000003E0B 000000003E0B 00000000 ! +- „jump 0x8100“ +- R0=R0-R0, DM(I4,M5)=R9, PM(I12,M13)=R11	60
:2006CA	00	0000 000000000000 000000000000 000000000000 000000003E0B 000000003E0B	7E
:2006EA	00	000000003E0B 000000003E0B 0C8100003E06 000000003E0B 000000003E0B 0000	FB
..	..	..	..
:200C6A	00	0000 000000000000 000000000000 000000000000 000000000000 000000000000	6A
:0C0C8A	00	000000000000 000000000000 +- end of user code	5E
:000000	01	End of EPROM image	FF

Table 4. Boot image for ADSP-21065L

Application Code

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

  Blink Program

  History:
   1.0.1.0   20-JUL-99 HS
             changed directives .segment to .section and removed .endseg
 *****************************************************************************/

/*****************************************************************************
  Below stated constants represent the system with
    system clock (clock) in kHz and
    blink rate (brate) in Hz
 *****************************************************************************/
#define clock 60000
#define brate 5

#define delta_t (clock * 1000) / brate
#include <def21060.h>

.section/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 */
    jump tmz_svc;
         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 */

/*************************************************************************
  Main Program
 *************************************************************************/
.section/pm  seg_pmco;

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

    TCOUNT = delta_t;              /* set up timer section */
    TPERIOD = delta_t;

    -bit set MODE2 TIMEN;
    -bit set MODE1 IRPTEN;
    -bit set IMASK TMZHI;

w_loop:
    idle;                          /* wait for timer interrupt */
    jump w_loop;


/*************************************************************************
  Timer IRQ operation
 *************************************************************************/
tmz_svc:
    -bit tgl ASTAT FLG2 | FLG3;    /* toggle flags */
    rti;
```

Listing 1. Application Code stored in EPROM

References

[1] *ADSP-2106x SHARC User's Manual.* Second Edition. May 1997. Analog Devices Inc.

[2] *ADSP-21065L SHARC DSP Users Manual.* Revision 2.0, July 2003. Analog Devices Inc.

[3] *ADSP-21065L SHARC DSP Technical Reference.* Revision 2.0, July 2003. Analog Devices Inc.

[4] *VisualDSP++ 3.5 Loader Manual for 32-bit Processors*. Revision 1.0, March 2004. Analog Devices Inc.

Document History

Revision	Description
Rev 2 – March 24, 2004 *by Robert Hoffmann*	Updated PROM and host booting section, include a boot table for host booting
Rev 1 – July 20, 1999 *by Stefan Hacker*	Initial Release