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

GENERATING BOOT EPROMS FOR THE SHARC EZ-KIT LITE'S ADSP-21061 WITH A MULTIPROCESSOR ID=001

Last Modified: 9/08/99
Contributed by: Joe Tykowski , Peninsula Solutions
Edited by: JT, DSP Applications Group

OVERVIEW

The SHARC EZ-Kit-Lite is useful as a design, development, and demonstration platform for Digital Signal Processing applications. As a demonstration tool, it is often preferable to generate a PROM for the EZ-Kit rather than download an application over the serial port. This app note describes how to generate a bootable PROM from a "C" source file. This app note uses the Talk-Through "C" example program that comes with the EZ-Kit. The information presented here can be applied to any "C" program. Programmers developing code for the SHARC EZ-LITE Lite should be aware that the ADSP-21061 has it's multiprocessor ID pins, ID_{0-2}, set to 001. This is important when setting up the command line switches for the boot loader, LDR21k.

ARCHITECTURE FILE

The EZ-Kit Lite package requires the use of the EZKIT.ACH file for specifying how the internal memory of the EZ-Kit is organized. The Lite compiler must use the file:

C:\EZ-KIT\21K\LIB\EZKIT.ACH

Note: the full "C" compiler version allows the user to specify which .ach file is used on the compiler command line.

The EZKIT.ACH file that is included for demonstration purposes should be saved for future reference. In order to make a bootable PROM, modify the file from this:

Table 1 - Original EZKIT.ACH file

```
!-------------------------------------------------------------------
.SYSTEM         SHARC_EZKIT_Lite;
!
! This architecture file is required for used with the SHARC EZ-KIT
! Lite development software. It is structured for use with the C
! compiler but also can be used with assembly code.
!
```

```
! This architecture file allocates:
!    Internal 133 words of 48-bit run-time header in memory block 0
!              16 words of 48-bit initialization code in memory block 0
!             619 words of 48-bit kernel code in memory block 0
!            7424 words of 48-bit C code space in memory block 0
!              4K words of 32-bit PM C data space in memory block 0
!
!              8K words of 32-bit C DM data space in memory block 1
!              4K words of 32-bit C heap space in memory block 1
!            3712 words of 32-bit C stack space in memory block 1
!             384 words of 32-bit kernel data in memory block 1

.PROCESSOR = ADSP21061;

!   ------------------------------------------------------------
!   Internal memory Block 0
!   ------------------------------------------------------------
.SEGMENT/RAM/BEGIN=0x00020000 /END=0x00020084 /PM/WIDTH=48        seg_rth;
.SEGMENT/RAM/BEGIN=0x00020085 /END=0x00020094 /PM/WIDTH=48        seg_init;
.SEGMENT/RAM/BEGIN=0x00020095 /END=0x000202ff /PM/WIDTH=48        seg_knlc;
.SEGMENT/RAM/BEGIN=0x00020300 /END=0x00021fff /PM/WIDTH=48        seg_pmco;
.SEGMENT/RAM/BEGIN=0x00023000 /END=0x00023fff /PM/WIDTH=32        seg_pmda;

!   ------------------------------------------------------------
!   Internal memory Block 1
!   ------------------------------------------------------------
.SEGMENT/RAM/BEGIN=0x00024000 /END=0x00025fff /DM/WIDTH=32        seg_dmda;
.SEGMENT/RAM/BEGIN=0x00026000 /END=0x00026fff /DM/WIDTH=32 /cheap seg_heap;
.SEGMENT/RAM/BEGIN=0x00027000 /END=0x00027e7f /DM/WIDTH=32        seg_stak;
.SEGMENT/RAM/BEGIN=0x00027e80 /END=0x00027fff /DM/WIDTH=32        seg_knld;

!   ------------------------------------------------------------
!   External Memory Select 1 is reserved for the UART.
!   ------------------------------------------------------------

.ENDSYS;
```

To this:

Table 2 - "Promable" EZKIT.ACH file

```
!---------------------------------------------------------------------
.SYSTEM         SHARC_EZKIT_Lite;
!
! This architecture file is required for used with the SHARC EZ-KIT
! Lite development software. It is structured for use with the C
! compiler but also can be used with assembly code.
!
! When compiling with g21k the "-nomem" switch should be used to
! disable unnecessary runtime initialization.
!
! This architecture file allocates:
!    Internal 133 words of 48-bit run-time header in memory block 0
!              16 words of 48-bit initialization code in memory block 0
!            8043 words of 48-bit C code space in memory block 0
!              4K words of 32-bit PM C data space in memory block 0
!
!              8K words of 32-bit C DM data space in memory block 1
!              4K words of 32-bit C heap space in memory block 1
!              4K words of 32-bit C stack space in memory block 1

.PROCESSOR = ADSP21061;
```

```
!   ------------------------------------------------------------
!   Internal memory Block 0
!   ------------------------------------------------------------
.SEGMENT/RAM/BEGIN=0x00020000 /END=0x00020084 /PM/WIDTH=48        seg_rth;
.SEGMENT/RAM/BEGIN=0x00020085 /END=0x00020094 /PM/WIDTH=48        seg_init;
.SEGMENT/RAM/BEGIN=0x00020095 /END=0x00021fff /PM/WIDTH=48        seg_pmco;
.SEGMENT/RAM/BEGIN=0x00023000 /END=0x00023fff /PM/WIDTH=32        seg_pmda;

!   ------------------------------------------------------------
!   Internal memory Block 1
!   ------------------------------------------------------------
.SEGMENT/RAM/BEGIN=0x00024000 /END=0x00025fff /DM/WIDTH=32        seg_dmda;
.SEGMENT/RAM/BEGIN=0x00026000 /END=0x00026fff /DM/WIDTH=32 /cheap seg_heap;
.SEGMENT/RAM/BEGIN=0x00027000 /END=0x00027fff /DM/WIDTH=32        seg_stak;

!   ------------------------------------------------------------
!   External Memory Select 1 is reserved for the UART.
!   ------------------------------------------------------------

.ENDSYS;
```

The files look very similar. The file shown in Table 2 reclaims the memory space previously occupied by the Ez-Kit's Kernel (used for downloading demonstration programs, examining memory, etc.) by removing the *seg_knlc* and *seg_knld* regions.

COMPILER COMMAND LINE

The compiler command line is shown in Table 3. Not all options shown are available for use with the EZ-Kit Lite compiler. They are shown for user's who may upgrade to the full "C" compiler version. This command will compile the "C" modules and link in the appropriate "C" library functions.

Table 3 - Compiler command line

```
C:\EZ-KIT\EZSHARC\TT\> g21k -o prom.exe -map  -nomem -save-temps -mlistm -Wall
-a c:\ez-kit\21k\lib\ezkit.ach -runhdr c:\ez-kit\21k\lib\060_hdr.obj tt.c
```

Where:

g21k	Invokes the "C" compiler.
`-o prom.exe`	Places the output in the "PROM.EXE" file.
`-map`	Generates a "PROM.MAP" file.
`-nomem`	Do not run the MEM21K initializer.
`-Wall`	Generate warning messages.
`-a c:\ez-kit\21k\lib\ezkit.ach`	Specify the architecture file.
`-runhdr c:\ez-kit\21k\lib\060_hdr.obj`	Specify the runtime header file. This

	includes “C” library initialization, interrupt vectors, and jump _main to the “C” program main() function.
`tt.c`	The “C” file(s) being compiled.

This command line generates the PROM.EXE file. This file must be processed by the *ldr21k* program to generate the actual prom data.

BOOT LOADER COMMAND LINE

Table 4 shows the command line syntax for creating a bootable prom file. It creates an Intel hex format file, PROM.IHX, which can be burned into a PROM using any standard PROM programmer.

Table 4 - Creating the PROM data file

```
C:\EZ-KIT\EZSHARC\TT\> ldr21k -bprom -o prom.ihx -v -id1exe=prom.exe
```

Where:

ldr21k	Invokes the “Boot Loader” program.
-bprom	Indicates that the boot method used will be of type **prom**.
-o prom.ihx	Specifies the output file name as “PROM.IHX”.
-v	Displays (optional) verbose output while running.
-id1exe=prom.exe	Identifies the input file for a single SHARC processor as “PROM.EXE”.

The new PROM is now ready to replace the EZ-Kit Lite Kernel PROM (U7). When powered up, the SHARC will boot from the PROM, load the application from the PROM, and start execution at the main() function in the tt.c program.

CONCLUSION:

This application note shows the steps required to create a bootable, “C” application for the SHARC using the SHARC EZ-Kit Lite, which have the $ID_{0\text{-}2}$ =001. Care must be taken to ensure the loader command line generates an EPROM file for the EZ-KIT Lite's 21061, which is hardwired with its ID pins = 1.

About the Author: Joe Tykowski is a Senior Software Engineer for Peninsula Solutions, Inc., in Winter Park, FL. Peninsula Solutions is a consulting firm which performs design, development, and support for embedded applications, as well as DOS/Windows environments. He has a Bachelor of Engineering Degree from Stevens Institute of Technology in Electrical Engineering / Computer Science.