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

C Programs on the ADSP-2181 EZ-KIT LITE

Last Modified: 5/6/96

This DSP EZ-Note explains in detail a basic ADSP-2100 family C program. This C routine provides a simple shell for use with the ADDS-21XX-EZ-LITE. The EZ-LITE is the low-cost ADSP-2181 based evaluation board. While the EZ-LITE comes with software, it does not include the C compiler. So this EZ-Note assumes that you have the complete version of the Analog Devices software (ADDS-21XX-SW-PC). The current version of the software is release 5.1.

Our main C program is CTIP35.C. As you can see from the listing, individual lines have been numbered. The lines are explained in the corresponding sections below. CTIP35.C when run on the EZ-LITE takes the input from the AD1847 stereo codec called LEFT_IN and RIGHT_IN and loops it back to the 1847 by writing to LEFT_OUT and RIGHT_OUT. You can add on to this simple talkthru program by modifying the inputs before writing to the outputs.

The file CTIP35.ZIP contains all the system files needed to test this application on your own EZ-LITE system. CTIP35.ZIP is available on the FTP site. Files included in the ZIP file are:

CTIP35.C
SIGNAL.H,
2181_HDR.DSP
TALK_47.DSP
BUILD.BAT
2181.ACH

The file TALK_47.DSP is an assembly routine that is called by CTIP35.C and initializes the interface between the ADSP-2181 and the AD1847. The file SIGNAL.H is a header file that contains macros for handling interrupts. 2181_HDR.DSP contains a modified run-time header for the 2181. BUILD.BAT is a batch file that assembles 2181HDR.DSP and then invokes the C compiler and all its switches. And finally 2181.ACH is an architecture file that can be used with the EZ-LITE.

For more information on C coding with the Analog Devices' DSPs refer to the C tools manual, the ADDS-21XX-SW-PC release note.

1) Extensions to the ANSI C compiler

The Analog Devices GNU based C compiler conforms to the ANSI C standard. However, there are additions or extensions to the ANSI C language that are specific to the ADSP processors. The instruction:

```
volatile int left_in, right_in;
```

is an example of an extension to the int data type. Using volatile will force the DSP to place the variables left_in and right_in into memory locations -- as opposed to manipulating the data in registers only. Another extension to ANSI C is the PM data type. Using the data type:

```
int pm coeff[10];
```

will create a buffer in the DSP's program memory for storing coefficients. Without the pm extension all data will be placed in data memory.

2) Accessing memory-mapped registers in C

The DSP reserves a section of internal memory for configuration registers. Configuring the serial ports, internal timer, IDMA, BDMA, memory wait states, etc. is accomplished by writing to these memory mapped registers. To accomplish this in C, use a #define command to create a label or pointer for the memory location. In our CTIP35.C program we're initializing the wait states for the IO memory space using:

```
#define DM_Wait_Reg *(int *) 0x3ffe.
```

Then in the code, you can write to that memory location using:

```
DM_Wait_Reg=0x0fff;
```

CTIP35.C

```
/* CTIP 35 */
/* this program can be used as a shell for C programs running on the
   EZ-LITE board or as an basic example of C code */
```

```
#include <signal.h>
#define  DM_Wait_Reg           *(int *)        0x3ffe

extern init_1847();
void new_sample();

asm(".var/dm/ram/circ rx_buf_[3];             /* Status + L data + R data */\n"\
    "\t.var/dm/ram/circ tx_buf_[3];           /* Cmd + L data + R data    */\n"\
    "\t.init tx_buf_: 0xc002, 0x0000, 0x0000; /* Initially set MCE        */\n"\
    "\t.global rx_buf_, tx_buf_;");

volatile int left_in, right_in;
volatile int left_out, right_out;
int stat_flag;

void main()
{
        init_1847();                            /* initialize 1847 interface */

        asm("set fl1;");                  /* set LED on EZ-LITE */

        DM_Wait_Reg=0x0fff;

        interrupt(SIGSPORT0RECV, new_sample);

        while(1){

          asm("idle;");                   /* wait for an interrupt */

          right_out=right_in;             /* loop back for talk-thru */
          left_out=left_in;

        }
}

void new_sample()
{
        asm("ena sec_reg;");
        asm("ax0 = dm(rx_buf_+1);");      /* receive new sample */
        asm("ax1 = dm(rx_buf_+2);");
        asm("dm(left_in_) = ax0;");       /* write to memory */
        asm("dm(right_in_) = ax1;");

        asm("ax0 = dm(left_out_);");      /* read output sample */
        asm("ax1 = dm(right_out_);");
        asm("dm(tx_buf_+1) = ax0;");      /* transmit to codec */
        asm("dm(tx_buf_+2) = ax1;");

        asm("dis sec_reg;");
}
```

3) Handling interrupts.
The DSP is an interrupt driven processor. In order to program those interrupts in C, it would be helpful if you understood what interrupts are available on your DSP of choice. The 2181 for example has interrupts for serial ports, external signals, timer, powerdown, etc. If you don't feel like cracking open the data sheet or user's manual, you can use the CTIP35.C as a shell and skip this section. If you want to understand how we've configured the interrupts or learn how to configure interrupts for your own system then read on.

The first step to handling interrupts is to include the header file SIGNAL.H in your program using the line:

```
#include <signal.h>
```

SIGNAL.H contains macros for each interrupt of each 2100 family processor.

Note: SIGNAL.H is included with your software tools. However, even if you have the latest software, you may not have the latest version of the file. It is available for download from our FTP site or is included with the files if you download CTIP35.ZIP.

For CTIP35.C, we need to set up the interrupt for Serial Port 0 Receive. The macro has the following syntax:

interrupt(SIGSPORT0RECV, *new_sample*);

This line identifies the function *new_sample* as the interrupt handler for serial port 0 receive interrupts.

4)Embedding assembly instructions in C code
Admittedly, as soon as you add in-line assembly code you begin to sacrifice some of your C code's portability. However, there are just some instructions that you can only perform in assembly. Creating a buffer can be done just as easily in C as in assembly. However, only using the assembly directive:

```
asm(".var/dm/ram/circ rx_buf_[3];");
```

can you create a circular buffer. The IDLE instruction and system register accesses such as
the IMASK also need to be done with assembly code. The general format for embedding an assembler instruction is:

```
asm("assembly code;");
```

To include several assembly lines of code, use the format:

```
asm("first instruction;" \
"second instruction;");
```

The backslash (\) has to be the last character on the line (including comments). You can also use \n and \t for creating line returns and tabs in you listing file.

5) Calling DSP programs/routines
Another way to add assembly code to your C routine is to place the DSP code in a C callable function. The program TALK_47.DSP contains code to initialize the 1847 codec. TALK_47.DSP contains the routine init_1847() which is called from C. An underscore must be added to labels and variables that are initialized in C and accessed in assembly. The DSP label in TALK_47.DSP is init_1847_.

6) Handling Real Time environments
Most DSP programs are based on external signals. This program waits until a Serial Port Receive interrupt occurs and then performs the loop-back. The WHILE instruction is used to create an infinite loop. The IDLE assembly instruction makes the processor wait for an interrupt. The catch is your code has to complete before the next interrupt would occur. For example, if you receive data from the 1847 every 16kHz you'd have 1/16kHz or 62.5 micro-seconds between interrupts. Using a 33Mhz 2181, this equates to 2062.5 DSP cycles.

BUILD.BAT

```
7    asm21 2181_hdr -c -2181
7    g21 -a 2181 -o ctip35 -I. -mreserved=i2,i3 -runhdr
     2181_hdr.obj ctip35.c
       talk_47.dsp -g -save-temps -mlistm
```

7) Compiling your program
Before using the g21 command to compile the CTIP35.C program, you need to assemble your new 2181 run time header. Use the -c switch to make the 2181_HDR.OBJ file case sensitive. The switches that are only needed for debug are the -g, -save-temps, and -mlistm.

After using the batch file to compile your C code, you can download the executable CTIP35.EXE to the EZ-LITE using the EZ-LITE monitor program that runs under windows.