Engineer To Engineer Note EE-9

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_applications@analog.com

SHARC-To-AD1847 EZ-LAB Loopback Example (In C)

Last Modified: 10/25/96

Introduction

This note provides example code that shows how to program the ADSP-2106x Serial Port on the SHARC EZ-LAB board to communicate with the AD1847 Modular Analog Front End (MAFE) Board. The ADSP-2106x DSP receives input from the AD1847 through the serial port and transmits the data back out the serial port to the AD1847.

Using The Loopback Example

The EZ-LAB board has a set of LEDs and push-buttons that let you control and monitor the example code. One LED blinks periodically, and the other is under the flag1 push-button control. The input select, input gain, and sample rate of the AD1847 can be altered while the program is running using the flag1 and irq1 push-buttons.

An example of flag input LEDs, flag input push-buttons, and irq1 push-button operation is as follows:

- The flag2 LED blinks periodically
- The flag0 LED toggles when you press the flag1 push-button

The following attributes of the AD1847 operation can be altered while the program is running:

- Input select (line or mic)
- Input gain (16 levels)
- Sample rate (16 frequencies)

The program has three modes: input-source select (mode 0), input-gain select (mode 1), and sample-rate select (mode 2). You can modify the parameters of each mode. Scroll through modes by holding the flag1 push-button and pressing the irq1 push-button. Scroll through the parameters for a mode by pressing the irq1 push-button. The modes and their possible parameter settings are as follows:

- Input source parameters: line (default) or microphone input
- Input Gain formula: level * 1.5dB (default level = 0 --> 0dB)
- Sample rates in kHz (in parameter order): (0) 8 (default), (1) 5.5125, (2) 16, (3) 11.025, (4) 27.42857, (5) 18.9, (6) 32, (7) 12.05, (8) N/A, (9) 37.8, (10) N/A, (11) 44.1, (12) 48, (13) 33.075, (14) 9.6, and (15) 6.615

Building The Loopback Example

To compile the example (for use with the diag21k utility), use the following command (in DOS):

```
g21k -a ttc.ach -o ttc.21k talkthru.c
```

To compile the example (for use with the ezldr utility), use the following commands (in DOS):

```
g21k -a ttc.ach -o ttc.21k talkthru.c
```

and

```
ldr21k -a ttc.ach -bhost -fascii
       -o ttc.ldr ttc.21k
```

Loopback Example Code

This loopback example uses an AD1847 MAFE on an ADSP-2106x SHARC EZ-LAB board. Following this code example, a corresponding system architecture file appears.

```
/*-------------------------------------*/
/*
talkthru.C
SHARC EZ-LAB MAFE AUDIO Example
Authors
(ASM version):  Jan 13, 1995,
Bob Senko, BittWare Research Systems
(C version):   Jun 13, 1995,
Larry Reinhard, BittWare Research Systems
*/
/*-------------------------------------*/

/* ADSP-21060 System Register bit definitions */
#include <def21060.h>
#include <21060.h>
#include <signal.h>
#include <sport.h>
#include <macros.h>

#define BRD_DISABLED 0
#define BRD_ENABLED  7          /* AD1847 pins: RESET*, PWRDOWN*, BM */

/* program operating modes */
#define MODE_INPUT_SOURCE_SELECT 0
#define MODE_INPUT_GAIN_SELECT   1
#define MODE_SAMPLERATE_SELECT   2

                                                                  /* DMA Chain pointer bit definitions */
#define CP_PCI 0x20000          /* Program-Controlled Interrupts bit */
#define CP_MAF 0x1ffff          /* Valid memory address field bits   */

#define SetIOP(addr, val)  (* (int *) addr) = (val)
#define GetIOP(addr)       (* (int *) addr)

/*---------------------------------------------------------------------------*/
DEF_PORT(mafe_reset,int,mafeadrs,dm);             /* mafe reset port */

/*---------------------------------------------------------------------------*/
#define SZ_regs_1847 16
int regs_1847[SZ_regs_1847] = {
       /* Note that the MCE bit is maintained throughout initial
```

```
           programming to hold off premature autocalibration. */
        0xc000,                 /* index 0 - left input control */
        0xc100,                 /* index 1 - right input control */
        0xc280,                 /* index 2 - left aux 1 input control */
        0xc380,                 /* index 3 - right aux 1 input control */
        0xc480,                 /* index 4 - left aux 2 input control */
        0xc580,                 /* index 5 - right aux 2 input control */
        0xc600,                 /* index 6 - left dac control */
        0xc700,                 /* index 7 - right dac control */
        0xc850,                 /* index 8 - data format */
        0xc909,                 /* index 9 - interface configuration */
        0xca00,                 /* index 10 - pin control */
        0xcb00,                 /* index 11 - no register */
        0xcc40,                 /* index 12 - miscellaneous information */
        0xcd00,                 /* index 13 - digital mix control */
        0xce00,                 /* index 14 - no register */
        0x8f00};                /* index 15 - no register */

int rx_buf[3];                          /* receive buffer */
int tx_buf[3] = {0xc000, 0, 0};         /* transmit buffer */

/* DMA chaining Transfer Control Blocks */
typedef struct {
   unsigned   lpath3;     /* for mesh mulitprocessing  */
   unsigned   lpath2;     /* for mesh multiprocessing  */
   unsigned   lpath1;     /* for mesh multiprocessing  */
   unsigned   db;         /* General purpose register  */
   unsigned   gp;         /* General purpose register  */
   unsigned** cp;         /* Chain Pointer to next TCB */
   unsigned   c;          /* Count register            */
   int        im;         /* Index modifier register   */
   unsigned * ii;         /* Index register            */
} _tcb;

_tcb rx_tcb = {0, 0, 0, 0, 0, 0, 3, 1, 0};      /* receive tcb */
_tcb tx_tcb = {0, 0, 0, 0, 0, 0, 3, 1, 0};      /* transmit tcb */

int cmd_blk[8];                                 /* command block */

static int mode;                                /* parameter mode variable */
static int xmit_count;
static int * xmit_ptr;
```

```
/*---------------------------------------------------------------------------*/
/* periodic timer interrupt */
void timer_hi_prior( int sig_num )
{
   sig_num=sig_num; /* quiet compiler warning */

   /* toggle flag 2 LED */
   set_flag(SET_FLAG2, TGL_FLAG);
}

/*---------------------------------------------------------------------------*/
/* IRQ1 (button pressed) */
void irq1_asserted( int sig_num)
{
   int temp;
   int num_cmds;

   if (poll_flag_in(READ_FLAG1, RETURN_FLAG_STATE) == 1)
   {  /* Not pressing FLAG1 button, change parameters of current mode */

      if (xmit_count == 0)
      {

        xmit_ptr = cmd_blk;            /* initialize command block pointer */
        num_cmds=0;

        switch (mode)
        {
           case MODE_INPUT_SOURCE_SELECT:
              /* toggle bit 6 of left source slct */
              regs_1847[0] = (regs_1847[0] & 0x40) ? (regs_1847[0] & ~0x40)
           : (regs_1847[0] |  0x40);
              *xmit_ptr++ = regs_1847[0];

              /* toggle bit 6 of right source slct */
              regs_1847[1] = (regs_1847[1] & 0x40) ? (regs_1847[1] & ~0x40)
           : (regs_1847[1] |  0x40);
              *xmit_ptr++ = regs_1847[1];
              num_cmds=2;
              break;
```

```
            case MODE_INPUT_GAIN_SELECT:
               temp = (regs_1847[0] + 1) & 0x0f;
               regs_1847[0] = (regs_1847[0] & ~0x0f) | temp;
               *xmit_ptr++ = regs_1847[0];
               temp = (regs_1847[1] + 1) & 0x0f;
               regs_1847[1] = (regs_1847[1] & ~0x0f) | temp;
               *xmit_ptr++ = regs_1847[1];
               num_cmds=2;
               break;

            case MODE_SAMPLERATE_SELECT:
               temp = (regs_1847[8] + 1) & 0x0f;
               regs_1847[8] = (regs_1847[8] & ~0x0f) | temp;
               *xmit_ptr++ = regs_1847[8];
               num_cmds=1;
               break;

            default: /* ensure we have a valid mode! */
               mode = MODE_INPUT_SOURCE_SELECT;
               break;
         }

         *xmit_ptr = regs_1847[15];  /* add terminating command */
         num_cmds++;

         xmit_ptr = cmd_blk;         /* reset xmit pointer */
         xmit_count += num_cmds;     /* kick of string of commands */

       }
   }
   else
   {  /* FLAG1 button detected, change current mode */
      if (++mode > MODE_SAMPLERATE_SELECT)
         mode = MODE_INPUT_SOURCE_SELECT;
   }
}

/*---------------------------------------------------------------------------*/
/* Serial port transmit DMA complete */
void spt0_asserted( int sig_num )
{
   if (xmit_count)              /* commands left to transmit? */
   {
      tx_buf[0] = *xmit_ptr++;   /* put command into TX buffer */
      xmit_count--;
   }
}

/*---------------------------------------------------------------------------*/
/* Serial port receive DMA complete */
/* (copies received data buffers to transmit data buffers) */
void spr0_asserted( int sig_num )
{
   tx_buf[1] = rx_buf[1];       /* left channel */
   tx_buf[2] = rx_buf[2];       /* right channel */
}

/*---------------------------------------------------------------------------*/
void setup_1847( void )
{

   mafe_reset = BRD_DISABLED;           /* put MAFE into reset    */
   asm("nop;nop;nop;");                 /* delay at least 100 ns  */
   mafe_reset = BRD_ENABLED;            /* take MAFE out of reset */

   /* Configure SHARC serial port SPORT0 */

   /* Multichannel communications setup */
   sport0_iop.mtcs  = 0x00070007;       /* transmit on words 0,1,2,16,17,18 */
   sport0_iop.mrcs  = 0x00070007;       /* receive on words 0,1,2,16,17,18  */
   sport0_iop.mtccs = 0x00000000;       /* no companding on transmit        */
   sport0_iop.mrccs = 0x00000000;       /* no companding on receive         */

   /* TRANSMIT CONTROL REGISTER */
   /* STCTL0 <= 0x001c00f2      */
   /* An alternate (and more efficient) way of doing this would be to   */
   /* write the 32-bit register all at once with a statement like this: */
   /*        SetIOP(STCTL0, 0x001c00f2);                                */
   /* But the following is more descriptive...                          */

   sport0_iop.txc.mdf   = 1;    /* multichannel frame delay (MFD)       */
   sport0_iop.txc.schen = 1;    /* Tx DMA chaining enable               */
   sport0_iop.txc.sden  = 1;    /* Tx DMA enable                        */
   sport0_iop.txc.lafs  = 0;    /* Late TFS (alternate)                 */
```

```
sport0_iop.txc.ltfs  = 0;    /* Active low TFS                         */
sport0_iop.txc.ditfs = 0;    /* Data independent TFS                   */
sport0_iop.txc.itfs  = 0;    /* Internally generated TFS               */
sport0_iop.txc.tfsr  = 0;    /* TFS Required                           */

sport0_iop.txc.ckre  = 0;    /* Data and FS on clock rising edge       */
sport0_iop.txc.gclk  = 0;    /* Enable clock only during transmission*/
sport0_iop.txc.iclk  = 0;    /* Internally generated Tx clock          */
sport0_iop.txc.pack  = 0;    /* Unpack 32b words into two 16b tx's     */

sport0_iop.txc.slen  = 15;   /* Data word length minus one             */
sport0_iop.txc.sendn = 0;    /* Data word endian 1 = LSB first         */
sport0_iop.txc.dtype = SPORT_DTYPE_RIGHT_JUSTIFY_SIGN_EXTEND;
     /* Data type specifier                  */
sport0_iop.txc.spen  = 0;    /* Enable (clear for MC operation)        */



/* RECEIVE CONTROL REGISTER */
/* SRCTL0 <= 0x1f8c00f2     */
sport0_iop.rxc.nch   = 31;   /* multichannel number of channels - 1  */
sport0_iop.rxc.mce   = 1;    /* multichannel enable                    */
sport0_iop.rxc.spl   = 0;    /* Loop back configure (test)             */
sport0_iop.rxc.d2dma = 0;    /* Enable 2-dimensional DMA array         */
sport0_iop.rxc.schen = 1;    /* Rx DMA chaining enable                 */
sport0_iop.rxc.sden  = 1;    /* Rx DMA enable                          */
sport0_iop.rxc.lafs  = 0;    /* Late RFS (alternate)                   */
sport0_iop.rxc.ltfs  = 0;    /* Active low RFS                         */
sport0_iop.rxc.irfs  = 0;    /* Internally generated RFS               */
sport0_iop.rxc.rfsr  = 0;    /* RFS Required                           */
sport0_iop.rxc.ckre  = 0;    /* Data and FS on clock rising edge       */
sport0_iop.rxc.gclk  = 0;    /* Enable clock only during transmission*/
sport0_iop.rxc.iclk  = 0;    /* Internally generated Rx clock          */
sport0_iop.rxc.pack  = 0;    /* Pack two 16b rx's into 32b word        */

sport0_iop.rxc.slen  = 15;   /* Data word length minus one             */
sport0_iop.rxc.sendn = 0;    /* Data word endian 1 = LSB first         */
sport0_iop.rxc.dtype = SPORT_DTYPE_RIGHT_JUSTIFY_SIGN_EXTEND;
     /* Data type specifier                  */
sport0_iop.rxc.spen = 0;     /* Enable (clear for MC operation)        */

/* Enable sport0 xmit & rcv irqs (DMA enabled) */
interruptf(SIG_SPR0I, spr0_asserted);
```

```
   interruptf(SIG_SPT0I, spt0_asserted);

   /* Set up Transmit Transfer Control Block for chained DMA */
   tx_tcb.ii = tx_buf;          /* DMA source buffer address                */
   tx_tcb.cp = &tx_tcb.ii;      /* define ptr to next TCB (point to self)   */
   SetIOP(CP2, (((int)&tx_tcb.ii) & CP_MAF) | CP_PCI);
        /* define ptr to current TCB (kick off DMA) */
        /* (SPORT0 transmit uses DMA ch 2)          */

   /* Set up Receive Transfer Control Block for chained DMA */
   rx_tcb.ii = rx_buf;          /* DMA destination buffer address           */
   rx_tcb.cp = &rx_tcb.ii;      /* define ptr to next TCB (point to self)   */
   SetIOP(CP0, (((int)&rx_tcb.ii) & CP_MAF) | CP_PCI);
        /* define ptr to current TCB (kick off DMA) */
        /* (SPORT0 receive uses DMA ch 0)           */

   xmit_ptr = regs_1847;        /* pointer to initialization commands       */
   xmit_count = SZ_regs_1847;   /* number of commands (starts command stream
           during next TX DMA interrupt)            */
   while (xmit_count)
      idle();                   /* wait for all commands to be tx'd */

   while (!(rx_buf[0] & 0x0002))
      idle();                   /* wait for AD1847 autocal to start */

   while (rx_buf[0] & 0x0002)
      idle();                   /* wait for AD1847 autocal to finish */
}

/*---------------------------------------------------------------------------*/
void init_21k( void )
{
   timer_off();                   /* diable timer */
   timer_set(11111111,11111111);  /* timer interrupt at 3 Hz */

   xmit_count = 0;              /* initial xmit count */
   xmit_ptr = regs_1847;        /* point to first command in list */
   mode = 0;                    /* initial mode */

   asm("#include <def21060.h>");
   asm("bit set mode2 IRQ1E;"); /* make IRQ1 edge sensitive */
   asm("bit clr mode1 NESTM;"); /* disable interrupt nesting */
   /* enable timer (high priority) & IRQ1 interrupts */
   interruptf(SIG_TMZ0, timer_hi_prior);
   interruptf(SIG_IRQ1, irq1_asserted);

   /* turn flag LEDs off */
   set_flag(SET_FLAG2, SET_FLAG);
   set_flag(SET_FLAG0, SET_FLAG);
}

/*---------------------------------------------------------------------------*/
void main ( void )
{
   init_21k();
   setup_1847();

   /* turn on all LEDs */
   set_flag(SET_FLAG2, CLR_FLAG);
   set_flag(SET_FLAG0, CLR_FLAG);

   timer_on();

   for(;;)
   {
      /* wait for flag 1 button press and then release */
      poll_flag_in(READ_FLAG1, FLAG_IN_LO_TO_HI);

      /* toggle flag 0 LED */
      set_flag(SET_FLAG0, TGL_FLAG);
   };
}
```

Architecture File For Example Code

This system architecture file supports the previous loopback example for the AD1847 MAFE and ADSP-2106x SHARC EZ-LAB board.

```
----------------------------------------------------------------------------------------
!TTC.ACH - Architecture Description File for the AD1847 C Talkthru Example

.SYSTEM         EZ_LAB;
!
! ADSP-21062 Memory Map:
```

```
!   ------------------------------------------------
!   Internal memory  0x0000 0000 to 0x0007 ffff
!   ------------------------------------------------
!                    0x0000 0000 to 0x0000 00ff  IOP Regs
!                    0x0000 0100 to 0x0001 ffff  (reserved)
!           Block 0  0x0002 0000 to 0x0002 7fff  Normal Word (32/48) Addresses
!                   (0x0002 0000 to 0x0002 4fff) 48-bit words
!                   (0x0002 0000 to 0x0002 7fff) 32-bit words
!           Block 1  0x0002 8000 to 0x0002 ffff  Normal Word (32/48) Addresses
!                   (0x0002 8000 to 0x0002 cfff) 48-bit words
!                   (0x0002 8000 to 0x0002 ffff) 32-bit words
!  alias of Block 1  0x0003 0000 to 0x0003 7fff  Normal Word (32/48) Addresses
!  alias of Block 1  0x0003 8000 to 0x0003 ffff  Normal Word (32/48) Addresses
!           Block 0  0x0004 0000 to 0x0004 ffff  Short Word (16) Addresses
!           Block 1  0x0005 0000 to 0x0005 ffff  Short Word (16) Addresses
!  alias of Block 1  0x0006 0000 to 0x0006 ffff  Short Word (16) Addresses
!  alias of Block 1  0x0007 0000 to 0x0007 ffff  Short Word (16) Addresses
!   ------------------------------------------------
!   Multiproc memory 0x0008 0000 to 0x003f ffff
!   ------------------------------------------------
!                    0x0008 0000 to 0x000f ffff  SHARC ID=001 Internal memory
!                    0x0010 0000 to 0x0017 ffff  SHARC ID=010 Internal memory
!                    0x0018 0000 to 0x001f ffff  SHARC ID=011 Internal memory
!                    0x0020 0000 to 0x0027 ffff  SHARC ID=100 Internal memory
!                    0x0028 0000 to 0x002f ffff  SHARC ID=101 Internal memory
!                    0x0030 0000 to 0x0037 ffff  SHARC ID=110 Internal memory
!                    0x0038 0000 to 0x003f ffff  SHARC ID=all Internal memory
!   ------------------------------------------------
!   External memory  0x0040 0000 to 0xffff ffff
!   ------------------------------------------------
!
! This architecture file allocates:
!         Internal 256 words of run-time header in memory block 0
!                  256 words of initialization code in memory block 0
!                  18K words of C code space in memory block 0
!                 1.5K words of C PM data space in memory block 0
!                  16K words of C DM data space in memory block 1
!                   8K words of C heap space in memory block 1
!                   8K words of C stack space in memory block 1
!         External MAFE ports in bank 2

.PROCESSOR = ADSP21062;
```

```
!   -------------------------------------------------------------
!   Internal memory Block 0
!       0x0002 0000 to 0x0002 4fff 48-bit words
!       0x0002 0000 to 0x0002 7fff 32-bit words
!   -------------------------------------------------------------
.SEGMENT/RAM/BEGIN=0x00020000 /END=0x000200ff /PM/WIDTH=48      seg_rth;
.SEGMENT/RAM/BEGIN=0x00020100 /END=0x000201ff /PM/WIDTH=48      seg_init;
.SEGMENT/RAM/BEGIN=0x00020200 /END=0x000249ff /PM/WIDTH=48      seg_pmco;
.SEGMENT/RAM/BEGIN=0x00024a00 /END=0x00024fff /PM/WIDTH=40      seg_pmda;

!   -------------------------------------------------------------
!   Internal memory Block 1
!       0x0002 8000 to 0x0002 cfff 48-bit words
!       0x0002 8000 to 0x0002 ffff 32-bit words
!   -------------------------------------------------------------
.SEGMENT/RAM/BEGIN=0x00028000 /END=0x0002bfff /DM/WIDTH=32      seg_dmda;
.SEGMENT/RAM/BEGIN=0x0002c000 /END=0x0002dfff /DM/WIDTH=32 /cheap seg_heap;
.SEGMENT/RAM/BEGIN=0x0002e000 /END=0x0002ffff /DM/WIDTH=32      seg_stak;

!   -------------------------------------------------------------
!   External memory (Banked and unbanked)
!       These addresses assume the default bank size of 8K (MSIZE=0)
!   -------------------------------------------------------------
.SEGMENT/PORT/BEGIN 0x00404000 /END=0x00404000 /DM/WIDTH=32     mafeadrs;
.SEGMENT/PORT/BEGIN 0x00404001 /END=0x00404001 /DM/WIDTH=32     mafedata;

.ENDSYS;
```