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

An AD1847/ADSP-2181 loopback example using a single index register for SPORT autobuffering

Last Modified: 05/06/97

This Edition of Analog Devices Engineer's Note will provide some insight into interfacing the AD1847 SoundPort Codec to an ADSP-2181 using a single index register for transmitting and receiving data via SPORT autobuffering ad1847. The code was tested using an AD1847 codec on our ADSP-21XX EZ-KIT-LITE Evaluation Board. The example talkthru program uses the AD1847's 1 wire mode for transmitting and receiving of data in slots 0-5. The supplied listing contains the following for the 1-wire, talkthru example:

ADSP 2181 initialization

ADSP 1847 Codec initialization

Interrupt service routines

Changes from the MIC2OUT.DSP version:

1) The ADI EZ-KIT-LITE version did not set the digital mix control (a minor bug)

2) This version still takes two index registers to set up, but only needs one index register to run. The EZ-KIT-LITE version requires three index registers for initialization and two to run.

3) 1 wire mode is used for the SPORT interface. This prevents simultaneous tx and rx autobuffer requests, regardless of bus activity.

4) The same single-buffer TDM frame interface is used both for initialization and operation.

5) The initialization commands now come from program memory rather than data memory. In a real-time application, this saves the bother of initializing a data memory array.

6) Since only the SPORT0 interrupt can be active during initialization, the control flag is passed in the AF register rather than in a variable.

7) Only the SPORT0 RX interrupt is used, never the TX. In addition, the SPORT0 RX interrupt vector is altered after initialization for maximum speed & flexibility.

A single index interface to the AD1847 works because the autobuffer transfers are always separated by at least one 16 bit time slot (about 4us at 8 kHz). Transmit autobuffers happen at the beginning of a time slot, Rx autobuffers at the end. Since the Tx slots come before the Rx slots, there is never a conflict.

Note that this does not apply if two or more AD1847's are cascaded. In such a system, hardware should be added to allow a dummy time slot between the 6 slots used for each AD1847. This still allows for up to four AD1847's on one bus.

Source Code Listing for AD1847drv.dsp

```
.module/RAM/ABS=0 loopback;

.const  IDMA=                   0x3fe0;
.const  BDMA_BIAD=              0x3fe1;
.const  BDMA_BEAD=              0x3fe2;
.const  BDMA_BDMA_Ctrl=         0x3fe3;
.const  BDMA_BWCOUNT=           0x3fe4;
.const  PFDATA=                 0x3fe5;
.const  PFTYPE=                 0x3fe6;

.const  SPORT1_Autobuf=         0x3fef;
.const  SPORT1_RFSDIV=          0x3ff0;
.const  SPORT1_SCLKDIV=         0x3ff1;
.const  SPORT1_Control_Reg=     0x3ff2;
.const  SPORT0_Autobuf=         0x3ff3;
.const  SPORT0_RFSDIV=          0x3ff4;
.const  SPORT0_SCLKDIV=         0x3ff5;
.const  SPORT0_Control_Reg=     0x3ff6;
.const  SPORT0_TX_Channels0=    0x3ff7;
.const  SPORT0_TX_Channels1=    0x3ff8;
.const  SPORT0_RX_Channels0=    0x3ff9;
.const  SPORT0_RX_Channels1=    0x3ffa;
.const  TSCALE=                 0x3ffb;
.const  TCOUNT=                 0x3ffc;
.const  TPERIOD=                0x3ffd;
.const  DM_Wait_Reg=            0x3ffe;
.const  System_Control_Reg=     0x3fff;

.var/pm/ram                    init_cmds[14];

.var/dm/ram/circ                buf[6];
.var/dm/ram                     vol[2];

{-------------------------------------------------------------------------
 Note that the order here seems a little funny.  The control field is
where
  it is due to buffering in Sport0's transmitter.  It is actually the
control
  field that will be transmitted in the next frame.
-------------------------------------------------------------------------}
#define left_out  buf
#define right_out buf+1
#define control   buf+2
#define status    buf+3
#define left_in   buf+4
#define right_in  buf+5
```

```
.init init_cmds:
        0xc00600,     {
                        Left input control reg
                        b7-6: 0=left line 1
                              1=left aux 1
                              2=left line 2
                              3=left line 1 post-mixed loopback
                        b5-4: res
                        b3-0: left input gain x 1.5 dB
                      }
        0xc10600,     {
                        Right input control reg
                        b7-6: 0=right line 1
                              1=right aux 1
                              2=right line 2
                              3=right line 1 post-mixed loopback
                        b5-4: res
                        b3-0: right input gain x 1.5 dB
                      }
        0xc28800,     {
                        left aux 1 control reg
                        b7  : 1=left aux 1 mute
                        b6-5: res
                        b4-0: gain/atten x 1.5, 08= 0dB, 00= 12dB
                      }
        0xc38800,     {
                        right aux 1 control reg
                        b7  : 1=right aux 1 mute
                        b6-5: res
                        b4-0: gain/atten x 1.5, 08= 0dB, 00= 12dB
                      }
        0xc48800,     {
                        left aux 2 control reg
                        b7  : 1=left aux 2 mute
                        b6-5: res
                        b4-0: gain/atten x 1.5, 08= 0dB, 00= 12dB
                      }
        0xc58800,     {
                        right aux 2 control reg
                        b7  : 1=right aux 2 mute
                        b6-5: res
                        b4-0: gain/atten x 1.5, 08= 0dB, 00= 12dB
                      }
        0xc68000,     {
                        left DAC control reg
                        b7  : 1=left DAC mute
                        b6  : res
                        b5-0: attenuation x 1.5 dB
                      }
        0xc78000,     {
                        right DAC control reg
                        b7  : 1=right DAC mute
                        b6  : res
```

```
                b5-0: attenuation x 1.5 dB
            }
0xc85000,     {
                data format register
                b7  : res
                b5-6: 0=8-bit unsigned linear PCM
                      1=8-bit u-law companded
                      2=16-bit signed linear PCM
                      3=8-bit A-law companded
                b4  : 0=mono, 1=stereo
                b0-3: 0=  8.
                      1=  5.5125
                      2= 16.
                      3= 11.025
                      4= 27.42857
                      5= 18.9
                      6= 32.
                      7= 22.05
                      8=   .
                      9= 37.8
                      a=   .
                      b= 44.1
                      c= 48.
                      d= 33.075
                      e=  9.6
                      f=  6.615
               (b0) : 0=XTAL1 24.576 MHz; 1=XTAL2 16.9344 MHz
            }
0xc90900,     {
                interface configuration reg
                b7-4: res
                b3  : 1=autocalibrate
                b2-1: res
                b0  : 1=playback enabled
            }
0xca0000,     {
                pin control reg
                b7  : logic state of pin XCTL1
                b6  : logic state of pin XCTL0
                b5  : master - 1=tri-state CLKOUT
                      slave  - x=tri-state CLKOUT
                b4-0: res
            }
0xcc0000,     {
                miscellaneous information reg
                b7  : 1=16 slots per frame, 0=32 slots per frame
                b6  : 1=2-wire system, 0=1-wire system
                b5-0: res
            }
0xcd0000,     {   digital mix control reg
                b7-2: attenuation x 1.5 dB
                b1  : res
                b0  : 1=digital mix enabled
```

```
                    }

       0xaf0000;                    { else set done flag and }

{*****************************************************************************
 *
 *  Interrupt vector table
 *
 *****************************************************************************}
        jump start;  rti; rti; rti;     {00: reset }
        rti;         rti; rti; rti;     {04: IRQ2 }
        rti;         rti; rti; rti;     {08: IRQL1 }
        rti;         rti; rti; rti;     {0c: IRQL0 }
        rti;         rti; rti; rti;     {10: SPORT0 tx }
vrx0: jump frame_int;                   {14: SPORT0 rx }
                     rti; rti; rti;
        jump irqe;   rti; rti; rti;     {18: IRQE }
        rti;         rti; rti; rti;     {1c: BDMA }
        rti;         rti; rti; rti;     {20: SPORT1 tx or IRQ1 }
        rti;         rti; rti; rti;     {24: SPORT1 rx or IRQ0 }
        rti;         rti; rti; rti;     {28: timer }
        rti;         rti; rti; rti;     {2c: power down }


{*****************************************************************************
 *
 *  ADSP 2181 initialization
 *
 *****************************************************************************}
start:

      i1=^buf;
      l1=%buf;

      i4=SPORT0_Autobuf;
      l4=0;
      m1 = 1;
      m7=1;

{================== S E R I A L   P O R T   #0   S T U F F ==================}
    dm(i4,m7)=b#0000001010010111;   { - single index
            ---------------1  rx autobuffer
            --------------1-  tx autobuffer
            ------------01--  rmreg = M1
            ---------010----  rireg = I2
            -------01-------  tmreg
            ----010---------  tireg
            ---0------------  biasrnd for mac
            --0-------------
            -0--------------  clk dis
            0--------------- }

    dm(i4,m7)=0;  { SPORT0_RFSDIV }
```

```
    dm(i4,m7)=0;  { SPORT0_SCLKDIV }
    dm(i4,m7)=b#1000011000001111;   { SPORT0_Control_Reg
            ------------1111  16b word
            ----------00----  format
            -------
                |||   |!|+------- receive framing logic 0=pos, 1=neg
                |||   |!+-------- transmit data valid logic 0=pos, 1=neg
                |||   |+========= RFS 0=ext, 1=int
                |||   +---------- multichannel length 0=24, 1=32 words
                ||+-------------- | frame sync to occur this number of clock
                ||                | cycle before first bit
                ||                |
                ||                |
                |+--------------- ISCLK 0=ext, 1=int
                +---------------- multichannel 0=disable, 1=enable
-----------------------------------------------------}
{ Sport0 multi channel enables }

    dm(i4,m7)=b#0000000000000111;   {  tx 15..0 }
    dm(i4,m7)=b#0000000000000111;   { tx 31..16 }
    dm(i4,m7)=b#0000000000111000;   { rx }
    dm(i4,m7)=b#0000000000111000;


{============== S Y S T E M   A N D   M E M O R Y   S T U F F ==============}
        ax0 = b#0000111111111111;   dm (DM_Wait_Reg) = ax0;
            {   |+-/+-/+-/+-/+-/- ! IOWAIT0
                ||  |  !  |       !
                ||  |  !  |       !
                ||  |  !  +------ | IOWAIT1
                ||  |  !           |
                ||  |  !           |
                ||  |  +---------- ! IOWAIT2
                ||  |              !
                ||  |              !
                ||  +------------- | IOWAIT3
                ||                 |
                ||                 |
                |+================ ! DWAIT
                |                   !
                |                   !
                +------------------ 0
            }

        ax0 = b#0001000000000000;   dm (System_Control_Reg) = ax0;
            {   +-/!||+-----/+-/- | program memory wait states
                |  !|+------------ SPORT1 1=serial port, 0=FI, FO, IRQ0, IRQ1,..
                |  !+------------- SPORT1 1=enabled, 0=disabled
                |  +============== SPORT0 1=enabled, 0=disabled
            }


        ifc = b#00000011111111;         { clear pending interrupt }
```

```
        nop;


        icntl = b#00000;
            {     ||||+- | IRQ0: 0=level, 1=edge
                  |||+-- | IRQ1: 0=level, 1=edge
                  ||+--- | IRQ2: 0=level, 1=edge
                  |+---- 0
                  |----- | IRQ nesting: 0=disabled, 1=enabled
            }


        mstat = b#1000000;
            {     ||||||+- | Data register bank select
                  |||||+-- | FFT bit reverse mode (DAG1)
                  ||||+--- | ALU overflow latch mode, 1=sticky
                  |||+---- | AR saturation mode, 1=saturate, 0=wrap
                  ||+----- | MAC result, 0=fractional, 1=integer
                  |+------ | timer enable
                  +------- | GO MODE
            }



{*************************************************************************
*******
*
*  ADSP 1847 Codec initialization
*
* At this point, the rx autobuffer is already running, but w/o
interrupts
* enabled.  I1,m1  operates a 3 byte auto-buffer for TX_BUF.  We write
to
* TX0 here to start the tx autobuffer going.
*
*  Every time the tx autobuffer wraps, an interrupt occurs that will
* clear STAT_FLAG
*************************************************************************
*******}


        i4 = ^init_cmds;

      af=pass 0;
        ifc = b#00000011111111;     { clear any pending interrupt }
        nop;
        {   enable rx interrupt }
        imask = b#0000110000;
            {     |||||||||+ | timer
                  ||||||||+- | SPORT1 rec or IRQ0
                  |||||||+-- | SPORT1 trx or IRQ1
                  ||||||+--- | BDMA
                  |||||+---- | IRQE
```

```
                  ||||+----- | SPORT0 rec
                  |||+------ | SPORT0 trx
                  ||+------- | IRQL0
                  |+-------- | IRQL1
                  +--------- | IRQ2
            }

      idle;                         { wait for a wrap }
        ax0 = 0xc000;               { start interrupt }
        tx0 = ax0;
        ar=14;
        af = pass ar;               { interation count for tx int }

check_init:
        af = pass af;               { buffer to be sent to }
        if ne jump check_init;      { the codec            }

check_aci1:
        ax0 = dm(status);           { once initialized, wait for codec }
        ar = ax0 and 2;             { to come out of auto-calibration }
        if eq jump check_aci1;      { wait for bit set }

check_aci2:
        ax0 = dm (status);          { wait for bit clear }
        ar = ax0 and 2;
        if ne jump check_aci2;
        idle;

        ar = 0x8600;
        dm (control) = ar;    { un-mute left DAC }
        idle;

        ar = 0x8700;          { un-mute right DAC }
        dm (control) = ar;
        idle;

      i4=^xecho;              { re-vector frame interrupt service routine }
      ax0=pm(i4,m7);
      i4=^vrx0;
      pm(i4,m7)=ax0;


{-------------------------------------------------------------------------
 -
 -  wait for interrupt and loop forever
 -
 -------------------------------------------------------------------------}

talkthru:       idle;
        jump talkthru;
```

```
{*****************************************************************************
 *
 *  Interrupt service routines
 *
 *****************************************************************************}

{-----------------------------------------------------------------------------
 -
 -  Frame interrupt for Codec initialization & interface
 -
 -----------------------------------------------------------------------------}
frame_int:
      af = pass af;
      if eq rti;              { if not initializing }
        ax0=pm(i4,m7);        { ad1847 command }
      dm(control)=ax0;        { place in transmit slot 0    }
        af=af-1;
      rti;

xecho:      jump echo;        { frame proc. for after initialization }

echo: ar=dm(right_in);        { loopback inputs to outputs }
      dm(right_out)=ar;
      dm(left_out)=ar;
      rti;

{------------------}

irqe:   reset fl1;
        rti;

.endmod;
```