

```

/*****
/*
/*----- F M U T I L . C -----*/
/* Task          : Utilities for accessing FM synthesis of      */
/*                  Sound Blaster card.                          */
/*-----*/
/* Author         : Michael Tischer / Bruno Jennrich           */
/* Developed on    : 02/06/1994                                  */
/* Last update     : 04/05/1995                                  */
/*-----*/
/* COMPILER       : Borland C++ 3.1, Microsoft Visual C++ 1.5  */
/*****
#ifdef __FMUTIL_C
/* FMUTIL.C be #Included */
#define __FMUTIL_C
/*--- Add include files -----*/

#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <sys\timeb.h>

#include "types.h"
#include "sbutil.h"
#include "fmutil.h"
#include "dsputil.h"

/*--- Global Variables -----*/

BYTE OPL2Mirror[ 2 ][ 256 ];      /* Mirroring Sound Blaster registers */
/* Offsets of oscillator */
BYTE Oscillator[ 18 ] = { 0x00,0x01,0x02,0x03,0x04,0x05,0x08,0x09,0x0a,
                          0x0b,0x0c,0x0d,0x10,0x11,0x12,0x13,0x14,0x15 };
/* Numbers of modulator/carrier oscillators */
BYTE Modulator[ 9 ] = { 0,1,2,6,7,8,12,13,14 };
/* Modulators */
BYTE Carrier [ 9 ] = { 3,4,5,9,10,11,15,16,17 };
/* Carrier (Carrier ) */
INT Channel [ 18 ] = { 0, 1, 2, 0, 1, 2, 3, 4, 5,
                      3, 4, 5, 6, 7, 8, 6, 7, 8 };
SBBASE _FP FMBASE;

/*****
/* fm_Write: Write value to Sound Blaster register and mirror */
/*-----*/
/* Input   : iOpl      - Number of OPL2 chip (0,1) (only for OPL3) */
/*           iReg       - Number of register to be written          */
/*           iVal       - New register value                        */
/* Output  : Success of operation                                   */
/*           TRUE       - Able to set register                     */
/*           FALSE      - Register not set (no second OPL2!)       */
/*****
INT fm_Write( INT iOpl, INT iReg, BYTE iVal )
{ INT iOplBase;
  if( iOpl && ( FMBASE.uDspVersion < DSP_3XX ) ) return FALSE;
/* send register to be written */

  iOplBase = FMBASE.iDspPort;
  if( iOpl ) iOplBase += 2;
  outp( iOplBase + SB_REGPORT, iReg );
/* Wait to prevent card from being "swallowed" */
  inp ( iOplBase + SB_REGPORT );
/* send register contents to be written */
  outp( iOplBase + SB_DATAPORT, iVal );
/* write new register value for read accesses to memory */
  if( iOpl ) OPL2Mirror[ FM_SECNDOP2 ][ iReg ] = iVal;
  else      OPL2Mirror[ FM_FIRSTOP2 ][ iReg ] = iVal;
  return TRUE;
}

/*****
/* fm_WriteBit: Clear/set bit in Sound Blaster register */
/*-----*/
/* Input   : iOpl      - Number of OPL2 chip (0,1) (only for OPL3) */
/*           iReg       - Number of register to be written          */
/*           iBit       - Number of bit to be changed (0-7)         */
/*           bSet       - TRUE: Set Bit                              */

```

```

/*          FALSE: Clear Bit                                     */
/* Output : Success of operation                                */
/*          TRUE  - Able to set register                        */
/*          FALSE - Register not set (no second OPL2!)         */
/******/
INT fm_WriteBit( INT iOpl, INT iReg, INT iBit, BYTE bSet )
{
    return fm_Write( iOpl, iReg,
        ( BYTE ) ( ( OPL2Mirror[ ( iOpl ? 1 : 0 ) ][ iReg ] &
            ( ~( 1 << iBit ) ) ) |          /* hide bit to be changed... */
            ( ( bSet ? 1 : 0 ) << iBit ) ) ); /* and if necessary, show */
}

/******/
/* Reset_Card: Reset all card registers. Well-defined exit     */
/*          condition!                                          */
/******/
VOID fm_Reset( VOID )
{
    INT i;
    for( i = 0; i <= 255; i++ )
    {
        fm_Write( FM_FIRSTOPL2, ( BYTE ) i, 0);          /* 1. OPL2 */
        fm_Write( FM_SECNDOPL2, ( BYTE ) i, 0);          /* 2. OPL2 */
    }
}

/******/
/* fm_SetBase : Furnish routines with initialized SB base     */
/*          structure                                           */
/******/
/* Input  : pSBBASE - Address of an initialized SB base structure */
/*          iReset  - if <> 0 - Reset Sound Blaster              */
/******/
VOID fm_SetBase( PSBBASE pSBBASE, INT iReset )
{
    dsp_SetBase( pSBBASE, iReset );
    FMBASE = *pSBBASE;
    if( iReset ) fm_Reset();
}

/******/
/* fm_GetAdLib: Look for ports on AdLib compatible card       */
/******/
/* Input  : pSBBASE - Address of SB base structure to be      */
/*          initialized                                         */
/* Output : NO_ERROR - AdLib compatible card found             */
/*          else      - not an AdLib card                      */
/******/
INT fm_GetAdLib( PSBBASE pSBBASE )
{
    WORD cx;

    pSBBASE->iDspPort = ADLIB_PORT;          /* AdLib Port 0x388/0x389 */
    pSBBASE->iMixPort = -1;
    pSBBASE->iMixPort = -1;
    pSBBASE->uDspVersion = 0xFFFFU;          /* will be initialized later */
    pSBBASE->iDspIrq = -1;
    pSBBASE->iDspDmaB = -1;
    pSBBASE->iDspDmaW = -1;

    fm_SetBase( pSBBASE, TRUE );             /* Initialize base port and mirror */

    fm_Write( FM_FIRSTOPL2, AL_TIMER1, 0xFF ); /* Initial value Timer 1 */
    fm_Write( FM_FIRSTOPL2, AL_TISTATE, AL_TIRESET ); /* Reset timer */
    fm_Write( FM_FIRSTOPL2, AL_TISTATE, AL_TI1STARTSTOP ); /* start */

    cx = 0xFFFF;
    while( !( inp( pSBBASE->iDspPort ) & AL_STAT1OVRFLW ) && cx ) cx--;
    return cx ? NO_ERROR : ERROR;
}

/******/
/* fm_GetChannel : Get channel number of an oscillator        */
/******/
/* Input  : o_nr - Number (not offset) of oscillator          */

```

```

/* Output : Number of matching channels */
/*****
INT fm_GetChannel( INT o_nr ){ return Channel[ o_nr ]; }

/*****
/* fm_GetModulator : Get oscillator number of a channel */
/*****
/* Input : ch_nr - Number of channel */
/* Output : Oscillator number of matching modulator */
/*****
INT fm_GetModulator( INT ch_nr ){ return Modulator[ ch_nr ]; }

/*****
/* fm_GetCarrier : Get oscillator number of a channel */
/*****
/* Input : ch_nr - Number of channel */
/* Output : Oscillator number of matching carrier */
/*****
INT fm_GetCarrier( INT ch_nr ){ return Carrier[ ch_nr ]; }

/*****
/* fm_SetOscillator: Set oscillator parameters */
/*****
/* Input : iOpl - Number of OPL2 chip (only for OPL3!) */
/* iOsc - Number (not offset) of oscillator */
/* bA - Attack ( 0 - 15 ) */
/* bD - Decay ( 0 - 15 ) */
/* bS - Sustain ( 0 - 15 ) */
/* bR - Release ( 0 - 15 ) */
/* bShortADSR - Envelope shortening with increasing octave */
/* bContADSR - Continuous envelope on/off */
/* bVibrato - Vibrato on/off */
/* bTremolo - Tremolo on/off */
/* bMute - Mute ( 0: loud, 63: soft ) */
/* bHiMute - High mute */
/* 0: 0dB per octave */
/* 1: 3dB per octave */
/* 2: 1.5dB per octave */
/* 1: 6dB per octave */
/* bFrqFactor - Multiplication parameters for frequency */
/* bWave - Wave form ( 0 - 3 ) */
/* Output : none */
/*****
/* Info: Oscillator numbers != Oscillator offsets */
/* Frequency parameter is not a linear factor */
/*****
VOID fm_SetOscillator( INT iOpl, INT iOsc, BYTE bA, BYTE bD, BYTE bS,
                     BYTE bR, BYTE bShortADSR, BYTE bContADSR,
                     BYTE bVibrato, BYTE bTremolo, BYTE bMute,
                     BYTE bHiMute, BYTE bFrqFactor, BYTE bWave
                     )
{
    /* Set all single bits */
    fm_Write( iOpl, 0x20 + Oscillator[ iOsc ],
              (BYTE)((OPL2Mirror[ iOpl ][0x20+Oscillator[iOsc]]&0xF0)| bFrqFactor));
    fm_WriteBit( iOpl, 0x20 + Oscillator[ iOsc ], 4, bShortADSR );
    fm_WriteBit( iOpl, 0x20 + Oscillator[ iOsc ], 5, bContADSR );
    fm_WriteBit( iOpl, 0x20 + Oscillator[ iOsc ], 6, bVibrato );
    fm_WriteBit( iOpl, 0x20 + Oscillator[ iOsc ], 7, bTremolo );

    /* Attack (Hi-Nibble) and Decay (Lo-Nibble) in bytes */
    fm_Write( iOpl, 0x60 + Oscillator[ iOsc ],
              ( BYTE ) ( bD | ( bA << 4 ) ) );

    /* Set mute */
    fm_Write( iOpl, 0x40 + Oscillator[ iOsc ],
              ( BYTE ) ( bMute | ( bHiMute << 5 ) ) );
    /* Sustain (Hi-Nibble) and Release (Lo-Nibble) in one byte */
    fm_Write( iOpl, 0x80 + Oscillator[ iOsc ],
              ( BYTE ) ( bR | ( bS << 4 ) ) );

    fm_WriteBit( iOpl, 0x01, 5, TRUE ); /* Allow wave form change */
    fm_Write( iOpl, 0xE0 + Oscillator[ iOsc ], bWave );
    /* Write wave form.*/
    fm_WriteBit( iOpl, 0x01, 5, FALSE );
    /* Do not allow wave form change */
}

```

```

/*****
/* fm_SetModulator : Program modulator of a channel */
/*-----*/
/* Input   : iOpl      - Number of OPL2 chip (only for OPL3!) */
/*           iChn      - Channel number */
/*           bA        - Attack ( 0 - 15 ) */
/*           bD        - Decay ( 0 - 15 ) */
/*           bS        - Sustain ( 0 - 15 ) */
/*           bR        - Release ( 0 - 15 ) */
/*           bShortADSR - Envelope shortening with increasing octave */
/*           bContADSR  - Continuous envelope on/off */
/*           bVibrato   - Vibrato on/off */
/*           bTremolo   - Tremolo on/off */
/*           bMute      - Mute ( 0: loud, 63: soft ) */
/*           bHiMute    - High mute */
/*                       0: 0dB per octave */
/*                       1: 3dB per octave */
/*                       2: 1.5dB per octave */
/*                       1: 6dB per octave */
/*           bFrqFactor - Multiplication parameter for frequency */
/*           bWave      - Wave form ( 0 - 3 ) */
/* Output   : none */
/*-----*/
VOID fm_SetModulator( INT iOpl, INT iChn, BYTE bA, BYTE bD, BYTE bS,
                     BYTE bR, BYTE bShortADSR, BYTE bContADSR,
                     BYTE bVibrato, BYTE bTremolo, BYTE bMute,
                     BYTE bHiMute, BYTE bFrqFactor, BYTE bWave
                     )
{
    fm_SetOscillator( iOpl, Modulator[ iChn ], bA, bD, bS, bR,
                     bShortADSR, bContADSR, bVibrato, bTremolo,
                     bMute, bHiMute, bFrqFactor, bWave );
}

/*****
/* fm_SetCarrier : Carrier / Program carrier of a channel */
/*-----*/
/* Input   : iOpl      - Number of OPL2 chip (only for OPL3!) */
/*           iChn      - Channel number */
/*           bA        - Attack ( 0 - 15 ) */
/*           bD        - Decay ( 0 - 15 ) */
/*           bS        - Sustain ( 0 - 15 ) */
/*           bR        - Release ( 0 - 15 ) */
/*           bShortADSR - Envelope shortening with increasing octave */
/*           bContADSR  - Continuous envelope on/off */
/*           bVibrato   - Vibrato on/off */
/*           bTremolo   - Tremolo on/off */
/*           bMute      - Mute ( 0: loud, 63: soft ) */
/*           bHiMute    - High mute */
/*                       0: 0dB per octave */
/*                       1: 3dB per octave */
/*                       2: 1.5dB per octave */
/*                       1: 6dB per octave */
/*           bFrqFactor - Multiplication parameter for frequency */
/*           bWave      - Wave form ( 0 - 3 ) */
/* Output   : none */
/*-----*/
/* Info: Oscillator numbers != Oscillator offsets */
/*       Frequency parameter is not a linear factor */
/*****
VOID fm_SetCarrier( INT iOpl, INT iChn, BYTE bA, BYTE bD, BYTE bS,
                   BYTE bR, BYTE bShortADSR, BYTE bContADSR,
                   BYTE bVibrato, BYTE bTremolo, BYTE bMute,
                   BYTE bHiMute, BYTE bFrqFactor, BYTE bWave
                   )
{
    fm_SetOscillator( iOpl, Carrier[ iChn ], bA, bD, bS, bR,
                     bShortADSR, bContADSR, bVibrato, bTremolo,
                     bMute, bHiMute, bFrqFactor, bWave );
}

/*****
/* fm_SetChannel: Set channel */
/*-----*/
/* Input   : iOpl      - Number of OPL2 chip (only for OPL3!) */
/*           iChn      - Number (AND offset) of channel */

```

```

/*      bOct      - Octave (0 - 8) */
/*      iFrq      - Frequency (0 - 1023) */
/*      bFM       - FM synthesis or added oscillator */
/*      bFeedBack - Modulator feedback */
/* Output : none */
/*****
VOID fm_SetChannel( INT iOpl, INT iChn, BYTE bOct, INT iFrq, BYTE bFM,
                   BYTE bFeedBack
                   )
{
    /* Set LoByte of frequency */
    fm_Write( iOpl, 0xA0 + iChn, ( BYTE ) ( iFrq & 0xFF ) );
    /* Set 3 high frequency bits and octave */
    fm_Write( iOpl, 0xB0 + iChn,
              ( BYTE ) ( ( ( iFrq >> 8 ) & 0x3 ) | ( bOct << 2 ) ) );
    /* Oscillator link and modulator feedback */

    if( OPL2Mirror[ 1 ][ 5 ] & 1 ) /* OPL3-Stereo mode */
    {
        fm_Write( iOpl, 0xC0 + iChn,
                  ( BYTE )
                  ( ( OPL2Mirror[ iOpl ][ 0xC0 ] & 0xF1 ) | ( bFeedBack << 1 ) ) );
        fm_QuadroMode( iOpl, iChn, bFM );
    }
    else
        fm_Write( iOpl, 0xC0 + iChn,
                  ( BYTE ) ( ( OPL2Mirror[ iOpl ][ 0xC0 ] & 0xF0 ) |
                            ( ( bFM ? 0 : 1 ) | ( bFeedBack << 1 ) ) ) );
}

/*****
/* fm_SetCard: Set sound card parameters */
/*-----*/
/* Input : iOpl - Number of OPL2 chip (only for OPL3!) */
/*          bVibrato - TRUE : Vibrato depth = 14 / 100 of channel frequency */
/*          FALSE : Tremolo depth = 7 / 100 of channel frequency */
/*          bTremolo - TRUE : Tremolo depth = 4.8 dB */
/*          FALSE : Tremolo depth = 1 dB */
/* Output : none */
/*-----*/
/* Info: Tremolo and vibrato depth can only be set for all oscillators */
/*****
VOID fm_SetCard( INT iOpl, BYTE bVibrato, BYTE bTremolo )
{
    fm_WriteBit( iOpl, 0xBD, 6, bVibrato );
    fm_WriteBit( iOpl, 0xBD, 7, bTremolo );
}

/*****
/* fm_PlayChannel: Switch channel on and off */
/*-----*/
/* Input : iOpl - Number of OPL2 chip (only for OPL3!) */
/*          iChn - Number of Channels */
/*          bOn - TRUE: Switch sound on, FALSE: Switch sound off */
/* Output : none */
/*****
VOID fm_PlayChannel( INT iOpl, INT iChn, BYTE bOn )
{
    fm_WriteBit( iOpl, 0xB0 + iChn, 5, bOn );
}

/*****
/* Helper functions */
/*-----*/
/* Switch rhythm mode or rhythm instrument on and off */
/*****
VOID fm_PlayHiHat ( INT iOpl, BYTE bOn )
{ fm_WriteBit( iOpl, 0xBD, 0, bOn ); }
VOID fm_PlayTopCymbal ( INT iOpl, BYTE bOn )
{ fm_WriteBit( iOpl, 0xBD, 1, bOn ); }
VOID fm_PlayTomTom ( INT iOpl, BYTE bOn )
{ fm_WriteBit( iOpl, 0xBD, 2, bOn ); }
VOID fm_PlaySnareDrum ( INT iOpl, BYTE bOn )
{ fm_WriteBit( iOpl, 0xBD, 3, bOn ); }

```

```

VOID fm_PlayBassDrum ( INT iOpl, BYTE bOn )
{ fm_WriteBit( iOpl, 0xBD, 4, bOn ); }
VOID fm_PercussionMode( INT iOpl, BYTE bOn )
{ fm_WriteBit( iOpl, 0xBD, 5, bOn ); }

/*****
/* fm_PollTime : Waits specified number of thousandths of a second */
**-----**
/* Input : lMilli : Number of thousandths to wait */
/*****
VOID fm_PollTime( LONG lMilli )
{ struct timeb tb;
  LONG lEnd, lAct;

  ftime( &tb );
  lAct = tb.time * 1000L + tb.millitm; /* Get current time */
  lEnd = lAct + lMilli; /* Get end time */
  while( lAct < lEnd ) /* Current time < End time */
  {
    ftime( &tb ); /* Convert current time to thousandths: */
    lAct = tb.time * 1000L + tb.millitm;
  }
}

/*-- OPL3 - 4 Operator Synthesis -----*/

/*****
/* fm_QuadroOn : Toggle stereo operation of OLP */
**-----**
/* Input : iOn - TRUE : Enable second OPL2 (OPL3 mode) */
/*          FALSE : Disable second OPL2 (Compatibility mode) */
/*          */
/* Output : TRUE - Second OPL2 exists */
/*          FALSE - No second OPL2 */
**-----**
/* Note: Within the fm_Write(Bit) function the program determines */
/*        whether the second OPL2 can be addressed */
/*        or whether the Sound Blaster card has an OPL3. */
/*****
INT fm_QuadroOn( INT iOn )
{ /* Bit 0 set in register 5': second OPL2 active */
  return fm_WriteBit( FM_SECNDOP2, 0x05, 0, ( BYTE )iOn );
}

/*****
/* fm_ChannelLR : Set/place/put channel to left and/or right output */
**-----**
/* Input : iOpl - Number of OPL2 chip (only for OPL3!) */
/*          iChn - Number of Channels */
/*          iL - != 0 : Set channel to left output */
/*          iR - != 0 : Set channel to right output */
**-----**
/* Note: This function only works with an OPL3 */
/*****
VOID fm_ChannelLR( INT iOpl, INT iChn, INT iL, INT iR )
{
  fm_WriteBit( iOpl, 0xC0 + iChn, 4, ( BYTE )iL );
  fm_WriteBit( iOpl, 0xC0 + iChn, 5, ( BYTE )iR );
}

/*****
/* fm_QuadroChannel: Set number of operators for a channel */
**-----**
/* Input : iOpl - Number of OPL2 chip (only for OPL3!) */
/*          iCH0 != 0 : Channel 0 of OPL has 4 operators */
/*          == 0 : Channel 0 of OPL has 2 operators */
/*          iCH1 != 0 : Channel 1 of OPL has 4 operators */
/*          == 0 : Channel 1 of OPL has 2 operators */
/*          iCH2 != 0 : Channel 2 of OPL has 4 operators */
/*          == 0 : Channel 2 of OPL has 2 operators */
/*****
VOID fm_QuadroChannel( INT iOpl, INT iCH0, INT iCH1, INT iCH2 )
{
  fm_WriteBit( FM_SECNDOP2, 0x04, iOpl ? 3 + 0 : 0 + 0, ( BYTE )iCH0 );
  fm_WriteBit( FM_SECNDOP2, 0x04, iOpl ? 3 + 1 : 0 + 1, ( BYTE )iCH1 );
  fm_WriteBit( FM_SECNDOP2, 0x04, iOpl ? 3 + 2 : 0 + 2, ( BYTE )iCH2 );
}

```

```

}

/*****
/* fm_QuadroMode : Set cell linking for 4 operator          */
/*                  channels.                                */
/*-----*/
/* Input   : iOpl  - Number of OPL2 chip (only for OPL3!)    */
/*           iChn  - Number of channels (0-2)                */
/*           iMode - Link mode (0-3)                         */
/*-----*/
/* Note: The mode is distributed over the two "old" link bits */
/*       of both 2 operator channels of a 4 operator          */
/*       channel!                                              */
*****/
VOID fm_QuadroMode( INT iOpl, INT iChn, INT iMode )
{
    if( iChn < 3 )
    {
        fm_WriteBit( iOpl, 0xC0 + iChn, 0, (BYTE)(iMode & 0x01) );
        fm_WriteBit( iOpl, 0xC0 + iChn + 3, 0, (BYTE)(iMode & 0x02) );
    }
}
#endif

```