

FM.C

```

/*****
/*          F M . C          */
/*-----*/
/* Task      : Demonstrates accessing FM synthesis of the      */
/*            Sound Blaster card.                               */
/*-----*/
/* Author     : Michael Tischer / Bruno Jennrich              */
/* Developed on : 02/06/1994                                    */
/* Last update : 04/05/1995                                    */
/* Caution    : Set MASTER and MIDI volumes appropriately!    */
/*-----*/
/* COMPILER    : Borland C++ 3.1, Microsoft Visual C++ 1.5    */
/*****

```

```

/*--- Add Include files -----*/

```

```

#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

```

```

#include "kbd.h"
#include "win.h"
#include "sbutil.h"
#include "fmutil.h"
#include "dsputil.h"
#include "args.h"

```

```

/*- Remove following lines within a project: -----*/
#include "sbutil.c"
#include "fmutil.c"

```

```

#define DSP_VERSIONONLY                /* Use only version control */
#include "dsputil.c"
#include "win.c"
#include "args.c"

/*--- Macros and Constants -----*/
typedef struct tagKEY
{
    CHAR cKey;                        /* Key on keyboard */
    INT iNote;                        /* Note to be played */
    BYTE bOct;
} KEY;

SBBASE SBB;

KEY QWERTY[] = { { 'q', _c, 0 },
                  { 'w', _d, 0 },
                  { 'e', _e, 0 },
                  { 'r', _f, 0 },
                  { 't', _g, 0 },
                  { 'y', _a, 0 },
                  { 'u', _h, 0 },

                  { 'a', _c, 1 },
                  { 's', _d, 1 },
                  { 'd', _e, 1 },
                  { 'f', _f, 1 },
                  { 'g', _g, 1 },
                  { 'h', _a, 1 },
                  { 'j', _h, 1 },
                };

```

```
PBYTE screenbuf;
WINDOW win, tast, screen;
INTDATA Ints[ 15 ];
BOOLDATA Bools[ 6 ];
OBJECT Objects[ 20 ];
```

```
INT OPLOBJ,                /* "Constants" in order to react immediately */
MODEOBJ,                  /* to input. */
MODEINT,
MODEBOOL,
MODEY,
OPL3_4OPOBJ,
CHANNELOBJ,
OSCILLATOROBJ,
FRQOBJ,
FRQINT,
OCTAVEOBJ,
OCTAVEINT,
MAXOBJ;
```

```
typedef struct tagCHDATA
{
    INT iOctave, iFrequency, iFM, iFeedBack;
} CHDATA;
```

```
typedef struct tagOSCIDATA
{
    INT iAttack , iDecay, iSustain, iRelease;
    INT iShortADSR;
    INT iContADSR;
    INT iVibrato;
```

```

    INT iTremolo;
    INT iMute, iHiMute;
    INT iFRQ, iWave;
} OSCIDATA;

OSCIDATA OD;
OSCIDATA ALLOSC[ 2 ][ 18 ];

CHDATA CH;
CHDATA ALLCH[ 2 ][ 9 ];

INT iOpl;                                /* Current Oscillator */
INT iChannel;                            /* Current Channel */
INT iOscillator;
INT iOPL3;                               /* OPL3 chip present? */
INT iOPL3_4OP;                           /* 4 operators ? */

/*****
/* Func : Custom message function */
**-----**/
/* Input : iAct - OBJECT to which message refers/applies */
/*          iMsg - Message type */
/*          iParam - Integer parameter */
/*          lParam - Long parameter */
*****/
VOID Func( INT iAct, INT iMsg, INT iParam, LONG lParam )
{
    if( iMsg == MSG_CHANGED )
    {
        fm_SetOscillator( iOpl, iOscillator, /* Modulator */
                           ( BYTE )OD.iAttack, ( BYTE )OD.iDecay,
                           ( BYTE )OD.iSustain, ( BYTE )OD.iRelease,

```

```

        ( BYTE )OD.iShortADSR, ( BYTE )OD.iContADSR,
        ( BYTE )OD.iTremolo, ( BYTE )OD.iVibrato,
        ( BYTE )OD.iMute, ( BYTE )OD.iHiMute,
        ( BYTE )OD.iFRQ, ( BYTE )OD.iWave );

if( iAct == OPLOBJ )                                /* OPL change */
{
    ALLCH[ ( INT )lParam ][ iChannel ] = CH; /* save current data */
    ALLOSC[ ( INT )lParam ][ iOscillator ] = OD;

    CH = ALLCH[ iOpl ][ iChannel ];                /* Load new data */
    iOscillator = fm_GetModulator( iChannel );
    OD = ALLOSC[ iOpl ][ fm_GetModulator( iOscillator ) ];

    win_OBJECTPrintArray( &win, Objects, MAXOBJ, iAct );
}

if( iAct == OPL3_4OPOBJ )                            /* Number of operators */
{
    if( iOPL3_4OP )
    {
        CH.iFM = 0;
        win_OBJECTInitINT( &Objects[MODEOBJ],
                            &Ints[MODEINT], 0, MODEY, 100, 1,
                            "Cell linking:", 0, 3, &CH.iFM );
        fm_QuadroChannel( FM_FIRSTOPL2, TRUE, TRUE, TRUE );
        fm_QuadroChannel( FM_SECNDOPL2, TRUE, TRUE, TRUE );
    }
    else
    {
        CH.iFM = TRUE;
        win_OBJECTInitBOOL( &Objects[MODEOBJ],

```

```

        &Bools[MODEBOOL], 0, MODEY, 100, 1,
        "FM Synthesis          : ",
        DT_YESNO, &CH.iFM );
    fm_QuadroChannel( FM_FIRSTOPL2, FALSE, FALSE, FALSE );
    fm_QuadroChannel( FM_SECNDOPL2, FALSE, FALSE, FALSE );
}

win_OBJECTPrintArray( &win, Objects, MAXOBJ, iAct );
}

if( iAct == OSCILLATOROBJ )                /* Oscillator change */
{
    ALLOSC[ iOpl ][ ( INT )lParam ] = OD;
    OD = ALLOSC[ iOpl ][ iOscillator ];
    CH = ALLCH[ iOpl ][ fm_GetChannel( iOscillator ) ];
    iChannel = fm_GetChannel( iOscillator );

    win_OBJECTPrintArray( &win, Objects, MAXOBJ, iAct );
}

if( iAct == CHANNELOBJ )                   /* Channel change */
{
    ALLCH[ iOpl ][ ( INT )lParam ] = CH;

    CH = ALLCH[ iOpl ][ iChannel ];
    iOscillator = fm_GetModulator( iChannel );
    OD = ALLOSC[ iOpl ][ fm_GetModulator( iOscillator ) ];

    win_OBJECTPrintArray( &win, Objects, MAXOBJ, iAct );
}
fm_SetOscillator( iOpl, iOscillator,
    ( BYTE )OD.iAttack, ( BYTE )OD.iDecay,

```

```

        ( BYTE )OD.iSustain, ( BYTE )OD.iRelease,
        ( BYTE )OD.iShortADSR, ( BYTE )OD.iContADSR,
        ( BYTE )OD.iTremolo, ( BYTE )OD.iVibrato,
        ( BYTE )OD.iMute, ( BYTE )OD.iHiMute,
        ( BYTE )OD.iFRQ, ( BYTE )OD.iWave );
    /* Save current status of channel/oscillator */
    ALLOSC[ iOpl ][ iOscillator ] = OD;
    ALLCH[ iOpl ][ iChannel ] = CH;
}

if( iMsg == MSG_KEY )                                /* Keypress */
{
    switch( iParam )
    {
        case KBD_F1:
            fm_PlayBassDrum( iOpl, FALSE ); fm_PlayBassDrum( iOpl, TRUE );
            break;
        case KBD_F2:
            fm_PlayHiHat( iOpl, FALSE ); fm_PlayHiHat( iOpl, TRUE );
            break;
        case KBD_F3:
            fm_PlayTomTom( iOpl, FALSE ); fm_PlayTomTom( iOpl, TRUE );
            break;
        case KBD_F4:
            fm_PlaySnareDrum( iOpl, FALSE ); fm_PlaySnareDrum( iOpl, TRUE );
            break;
        case KBD_F5:
            fm_PlayTopCymbal( iOpl, FALSE ); fm_PlayTopCymbal( iOpl, TRUE );
            break;

        default:
            { int i;

```

```

fm_PlayChannel( iOpl, iChannel, ( BYTE )FALSE );
for( i = 0; i < sizeof( QWERTY ) / sizeof( KEY ); i++ )
    if( tolower( iParam ) == QWERTY[ i ].cKey )
    { INT old;
      fm_SetChannel( iOpl, iChannel,                /* Channel number */
                    ( BYTE )( QWERTY[ i ].bOct +    /* Octave */
                              ALLCH[ iOpl ][ iChannel ].iOctave ),
                    QWERTY[ i ].iNote,
                    ( BYTE ) ALLCH[ iOpl ][ iChannel ].iFM,
                    ( BYTE ) ALLCH[ iOpl ][ iChannel ].iFeedBack );
      fm_PlayChannel( iOpl, iChannel, ( BYTE )TRUE );

      /* Adapt display to new values! */
      win_LoVideo( &win );
      if( iAct == OCTAVEOBJ ) win_HiVideo( &win );
      old = *Ints[ OCTAVEINT ].pValue;
      *Ints[ OCTAVEINT ].pValue = QWERTY[ i ].bOct + CH.iOctave;
      win_OBJECTPrint( &win, &Objects[ OCTAVEOBJ ] );
      *Ints[ OCTAVEINT ].pValue = old;
      win_LoVideo( &win );

      if( iAct == FRQOBJ ) win_HiVideo( &win );
      *Ints[ FRQINT ].pValue = QWERTY[ i ].iNote;
      win_OBJECTPrint( &win, &Objects[ FRQOBJ ] );
    }
  }
}

/*-----*/

```



```

/*-- M A I N   P R O G R A M                               --*/
/*-----*-----*/

VOID main( VOID )
{ INT i, j, y, o, ints, bools;

  if( sb_GetEnviron( &SBB, getenv( "BLASTER" ) ) == NO_ERROR )
    fm_SetBase( &SBB, TRUE );
  else
    if( fm_GetAdLib( &SBB ) != NO_ERROR )
    {
      printf("BLASTER environment variable not available\n");
      printf("and no AdLib compatible sound card found!\n");
      exit(0);
    }

  iOPL3 = fm_QuadroOn( TRUE );
  iOPL3_4OP = FALSE;

  for( j = 0; j < 2; j++ )
  {
    for( i = 0; i < 9; i++ )
    {
      ALLCH[j][i].iOctave = 0;
      ALLCH[j][i].iFrequency = 0;
      ALLCH[j][i].iFM = TRUE;
      ALLCH[j][i].iFeedBack = 0;
      fm_SetChannel( j, i, ( BYTE )ALLCH[j][i].iOctave,
                     ALLCH[j][i].iFrequency,
                     ( BYTE )ALLCH[j][i].iFM,
                     ( BYTE )ALLCH[j][i].iFeedBack );
      /* Assign channels to outputs */
    }
  }
}

```

```

    if( iOPL3 ) fm_ChannelLR( j, i, j, !j );
}
for( i = 0; i < 18; i++)
{
    ALLOSC[j][i].iAttack = 10;
    ALLOSC[j][i].iDecay = 3;
    ALLOSC[j][i].iSustain = 15;
    ALLOSC[j][i].iRelease = 6;
    ALLOSC[j][i].iShortADSR = TRUE;
    ALLOSC[j][i].iContADSR = FALSE;
    ALLOSC[j][i].iTremolo = TRUE;
    ALLOSC[j][i].iVibrato = TRUE;
    ALLOSC[j][i].iMute = 0;
    ALLOSC[j][i].iHiMute = 0;
    ALLOSC[j][i].iFRQ = 3;
    ALLOSC[j][i].iWave = 1;

    fm_SetOscillator( j, i,
        ( BYTE )ALLOSC[j][i].iAttack,    ( BYTE )ALLOSC[j][i].iDecay,
        ( BYTE )ALLOSC[j][i].iSustain,    ( BYTE )ALLOSC[j][i].iRelease,
        ( BYTE )ALLOSC[j][i].iShortADSR,  ( BYTE )ALLOSC[j][i].iContADSR,
        ( BYTE )ALLOSC[j][i].iTremolo,    ( BYTE )ALLOSC[j][i].iVibrato,
        ( BYTE )ALLOSC[j][i].iMute,       ( BYTE )ALLOSC[j][i].iHiMute,
        ( BYTE )ALLOSC[j][i].iFRQ,        ( BYTE )ALLOSC[j][i].iWave );
    }
}

iOpl = 0;
iChannel = 0;

OD = ALLOSC[ iOpl ][ 0 ];
CH = ALLCH[ iOpl ][ 0 ];

```

```

o = 0;
y = 0;
ints = 0;
bools = 0;

OPLOBJ = -1;
if( iOPL3 )
{
    OPLOBJ = o;
    win_OBJECTInitINT( &Objects[o++], &Ints[ints++], 0, y++, 100, 1,
                      "OPL no.                :", 0, 1, &iOpl );

    OPL3_4OPOBJ = o;
    win_OBJECTInitBOOL( &Objects[o++], &Bools[bools++], 0, y++, 100, 1,
                      "4 operator          : ", DT_YESNO,
                      &iOPL3_4OP );

    y++; /* blank line */
}

CHANNELOBJ = o;
win_OBJECTInitINT( &Objects[o++], &Ints[ints++], 0, y++, 100, 1,
                  "Channel              (0-8):", 0, 8, &iChannel );

OCTAVEOBJ = o;
OCTAVEINT = ints;
win_OBJECTInitINT( &Objects[o++], &Ints[ints++], 0, y++, 100, 1,
                  "Octave                (0-8):", 0, 8, &CH.iOctave );

FRQOBJ = o;
FRQINT = ints;
win_OBJECTInitINT( &Objects[o++], &Ints[ints++], 0, y++, 100, 1,
                  "Frequency          (0-1024):", 0, 1024, &CH.iFrequency );

```

```

MODEOBJ = o;
MODEY = y;
MODEINT = ints;
MODEBOOL = bools;

if( iOPL3_4OP )
    win_OBJECTInitINT( &Objects[o++], &Ints[ints], 0, y++, 100, 1,
        "Cell link      :", 0, 3, &CH.iFM );
else
    win_OBJECTInitBOOL( &Objects[o++], &Bools[bools], 0, y++, 100, 1,
        "FM synthesis      : ", DT_YESNO, &CH.iFM );

ints++;
bools++;

win_OBJECTInitINT( &Objects[o++], &Ints[ints++], 0, y++, 100, 1,
    "Feedback      (0- 3):", 0, 3, &CH.iFeedBack );
y++; /* blank line */

OSCILLATOROBJ = o;
win_OBJECTInitINT( &Objects[o++], &Ints[ints++], 0, y++, 100, 1,
    "Oscillator      (0-17):", 0, 17, &iOscillator );
win_OBJECTInitINT( &Objects[o++], &Ints[ints++], 0, y++, 100, 1,
    "Attack          (0-15):", 0, 15, &OD.iAttack );
win_OBJECTInitINT( &Objects[o++], &Ints[ints++], 0, y++, 100, 1,
    "Decay           (0-15):", 0, 15, &OD.iDecay );
win_OBJECTInitINT( &Objects[o++], &Ints[ints++], 0, y++, 100, 1,
    "Sustain         (0-15):", 0, 15, &OD.iSustain );
win_OBJECTInitINT( &Objects[o++], &Ints[ints++], 0, y++, 100, 1,
    "Release         (0-15):", 0, 15, &OD.iRelease );
win_OBJECTInitBOOL( &Objects[o++], &Bools[bools++], 0, y++, 100, 1,
    "Short-ADSR      : ", DT_ONOFF, &OD.iShortADSR );
win_OBJECTInitBOOL( &Objects[o++], &Bools[bools++], 0, y++, 100, 1,

```

```

        "Cont.-ADSR          : ", DT_ONOFF, &OD.iContADSR );
win_OBJECTInitBOOL( &Objects[o++], &Bools[bools++], 0, y++, 100, 1,
        "Vibrato           : ", DT_ONOFF, &OD.iVibrato );
win_OBJECTInitBOOL( &Objects[o++], &Bools[bools++], 0, y++, 100, 1,
        "Tremolo           : ", DT_ONOFF, &OD.iTremolo );
win_OBJECTInitINT( &Objects[o++], &Ints[ints++], 0, y++, 100, 1,
        "Mute              (0-63):", 0, 63, &OD.iMute );
win_OBJECTInitINT( &Objects[o++], &Ints[ints++], 0, y++, 100, 1,
        "Hi-Mute           (0- 3):", 0, 3, &OD.iHiMute );
win_OBJECTInitINT( &Objects[o++], &Ints[ints++], 0, y++, 100, 1,
        "* Frq             (0-15):", 0, 15, &OD.iFRQ );
if( ioPL3 )
    win_OBJECTInitINT( &Objects[o++], &Ints[ints++], 0, y++, 100, 1,
        "Wave              (0- 7):", 0, 7, &OD.iWave );
else
    win_OBJECTInitINT( &Objects[o++], &Ints[ints++], 0, y++, 100, 1,
        "Wave              (0- 3):", 0, 3, &OD.iWave );

MAXOBJ = o;
win_GetScreenSettings( &screen );
screenbuf = win_Save( &screen );
win_Clr( &screen );

win_Init( &win, 1, 1, 30, 22, 0x1F, 0x71, WIN_ACTIVE );
win_Clr( &win );

win_Init( &tast, 32, 1, 44, 22, 0x1F, 0x17, WIN_ACTIVE );
win_Clr( &tast );

win_printf(&tast, "      Ý Ý Û Ý Ý Ý Û      Ý Ý Û Ý Ý Ý Û\n", -1 );
win_printf(&tast, "      Ý Ý Û Ý Ý Ý Û      Ý Ý Û Ý Ý Ý Û\n", -1 );

```

```

win_printf(&tast, "    û û û û û û û    û û û û û û û\n", -1 );
win_printf(&tast, "    û û û û û û û    û û û û û û û\n", -1 );
win_printf(&tast, "    q w e r t y u    a s d e f g h\n", -1 );
win_printf(&tast, "        1. Octave        2. Octave\n", -1 );
win_printf(&tast, "\n", -1 );
win_printf(&tast, "F1 - BassDrum    F2 - HiHat    F3 - TomTom\n", -1 );
win_printf(&tast, "F4 - SnareDrum    F4 - TopCymbal\n\n", -1 );
win_printf(&tast, "Navigation: \n", -1);
win_printf(&tast, "** <CURSOR UP> & <CURSOR DOWN>\n", -1);
win_printf(&tast, " Choice of setting\n\n", -1 );
win_printf(&tast, "** <+> & <->\n", -1);
win_printf(&tast, " different value\n\n", -1 );
win_printf(&tast, "<ESC> for Exit\n\n" );
win_printf(&tast, "Micro Piano (c)\n Michael Tischer &\n", -1 );
win_printf(&tast, " Bruno Jennrich", -1);
/* Enable percussion mode */
fm_PercussionMode( FM_FIRSTOPL2, TRUE );
fm_PercussionMode( FM_SECNDOP2, TRUE );

win_OBJECTProcessArray( &win, Objects, MAXOBJ, Func );

fm_Reset(); /* Complete silence */
win_Restore( screenbuf, TRUE );
}

```