

FM.PAS

```
{*****}
{                                     F M . P A S                                     }
{-----}
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
{*****}
```

```
{ $define DSP_VERSIONONLY }           { Use version control only }
Uses crt,WIN,SBUTIL,FMUTIL,DSPUTIL,ARGS,DOS;
```

Type

KEY = Record

```
    cKey : char;                { Key on keyboard }
    iNote : Integer;            { Note to be played }
    bOct  : Byte;
```

End;

CHDATA = Record

```
    iOctave,
    iFrequency : Integer;
    iFM        : Boolean;
    iFeedBack  : Integer;
```

End;

OSCIDATA = Record

```
    iAttack,
    iDecay,
```

```

iSustain,
iRelease   : Integer;
iShortADSR,
iContADSR,
iVibrato,
iTremolo   : Boolean;
iMute,
iHiMute,
iFRQ,
iWave      :Integer
End;

```

```

Const

```

```

  QWERTY : Array[0..13] of KEY= ( ( cKey:'Q'; iNote:_c; bOct:0 ),
                                   ( cKey:'W'; iNote:_d; bOct:0 ),
                                   ( cKey:'E'; iNote:_e; bOct:0 ),
                                   ( cKey:'R'; iNote:_f; bOct:0 ),
                                   ( cKey:'T'; iNote:_g; bOct:0 ),
                                   ( cKey:'Y'; iNote:_a; bOct:0 ),
                                   ( cKey:'U'; iNote:_h; bOct:0 ),

                                   ( cKey:'A'; iNote:_c; bOct:1 ),
                                   ( cKey:'S'; iNote:_d; bOct:1 ),
                                   ( cKey:'D'; iNote:_e; bOct:1 ),
                                   ( cKey:'F'; iNote:_f; bOct:1 ),
                                   ( cKey:'G'; iNote:_g; bOct:1 ),
                                   ( cKey:'H'; iNote:_a; bOct:1 ),
                                   ( cKey:'J'; iNote:_h; bOct:1 ) );

```

```

var
  SBB                : SBBASE;
  screenbuf          : ^Byte;

```

```

wind, key, screen : WINDOW ;
Ints                : Array[0..14] of INTDATA ;
Bools               : Array[0..5] of BOOLDATA;
Objects             : Array[0..19] of OBJ;

OPLOBJ,                                { 'Constants' in order to react immediately }
MODEOBJ,                                { to input.                               }
MODEINT,
MODEBOOL,
MODEY,
OPL3_4OPOBJ,
CHANNELOBJ,
OSCILLATOROBJ,
FRQOBJ,
FRQINT,
OCTAVEOBJ,
OCTAVEINT,
MAXOBJ                : Integer;

OD                : OSCIDATA;
ALLOSC : Array[0..1,0..17] of OSCIDATA;

CH                : CHDATA;
ALLCH  : Array[0..1,0..8] of CHDATA;

iOpl,                                { Current oscillator }
iChannel,                            { Current channel   }
iOscillator : Integer;
iOPL3,                                { OPL3 chip present? }
iOPL3_4OP  : Boolean;                { 4 operators ?   }

i, j, y, o, int, bool : Integer;

```

```

{*****}
{ Func : Custom message function }
{*-----*}
{ Input : iAct    - OBJECT to which message refers/applies }
{           iMsg   - Message type }
{           iParam - Integer parameter }
{           lParam - Long parameter }
{*****}
{$f+}

```

```

Procedure Func( iAct, iMsg, iParam : Integer; lParam : Longint );

```

```

var i,
    old : Integer;

```

```

Begin

```

```

    if iMsg = MSG_CHANGED then

```

```

        Begin

```

```

            fm_SetOscillator( iOpl,                      { Modulator }
                               iOscillator,
                               OD.iAttack,
                               OD.iDecay,
                               OD.iSustain,
                               OD.iRelease,
                               Byte ( OD.iShortADSR ),
                               Byte ( OD.iContADSR ),
                               Byte ( OD.iTremolo ),
                               Byte ( OD.iVibrato ),
                               OD.iMute,
                               OD.iHiMute,
                               OD.iFRQ,
                               OD.iWave );

```

```

if iAct = OPLOBJ then                                { OPL change }
  Begin
    ALLCH[lParam, iChannel] := CH;                    { save current data }
    ALLOSC[lParam ,iOscillator] := OD;

    CH := ALLCH[iOpl, iChannel];                      { load new data }
    iOscillator := fm_GetModulator( iChannel );
    OD := ALLOSC[iOpl, fm_GetModulator( iOscillator )];

    win_ObjectPrintArray( wind, @Objects, MAXOBJ, iAct );
  End;

if iAct = OPL3_4OPOBJ then                            { Number of operators }
  Begin
    if iOPL3_4OP then
      Begin
        CH.iFM := FALSE;
        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 );
      End
    else
      Begin
        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 );
      End
    end
  End

```

```

        fm_QuadroChannel( FM_SECNDOPPL2, FALSE, FALSE, FALSE );
    End;
    win_ObjectPrintArray( wind, @Objects, MAXOBJ, iAct );
End;

if iAct = OSCILLATOROBJ then                                { Oscillator change }
    Begin
        ALLOSC[iOpl ,lParam] := OD;
        OD := ALLOSC[iOpl , iOscillator];
        CH := ALLCH[iOpl , fm_GetChannel( iOscillator )];
        iChannel := fm_GetChannel( iOscillator );

        win_ObjectPrintArray( wind, @Objects, MAXOBJ, iAct );
    End;

if iAct = CHANNELOBJ then                                    { Channel change }
    Begin
        ALLCH[iOpl ,lParam] := CH;
        CH := ALLCH[iOpl , iChannel];
        iOscillator := fm_GetModulator( iChannel );
        OD := ALLOSC[iOpl , fm_GetModulator( iOscillator )];

        win_ObjectPrintArray( wind, @Objects, MAXOBJ, iAct );
    End;

fm_SetOscillator( iOpl, iOscillator,
    OD.iAttack, OD.iDecay,
    OD.iSustain, OD.iRelease,
    Byte ( OD.iShortADSR ), Byte ( OD.iContADSR ),
    Byte ( OD.iTremolo ), Byte ( OD.iVibrato ),
    OD.iMute, OD.iHiMute,
    OD.iFRQ, OD.iWave );

```

```

        { save current status of channel/oscillator }
    ALLOSC[iOpl ,iOscillator] := OD;
    ALLCH[iOpl ,iChannel] := CH;
End;

if iMsg = MSG_KEY then
    { keypress }
Begin
    fm_SetChannel( iOpl, iChannel,
                   { Channel number }
                   Byte ( ALLCH[iOpl ,iChannel].iOctave ),
                   Byte ( ALLCH[iOpl ,iChannel].iFrequency ),
                   Byte ( ALLCH[iOpl ,iChannel].iFM ),
                   ALLCH[iOpl ,iChannel].iFeedBack );

    case iParam of
        KBD_F1:
            begin
                fm_PlayChannel( iOpl, 6, FALSE );
                fm_PlayBassDrum( iOpl, FALSE );
                fm_PlayBassDrum( iOpl, TRUE );
            end;

        KBD_F2:
            begin
                fm_PlayChannel( iOpl, 7, FALSE );
                fm_PlayHiHat( iOpl, FALSE );
                fm_PlayHiHat( iOpl, TRUE );
            end;

        KBD_F3:
            begin
                fm_PlayChannel( iOpl, 8, FALSE );
                fm_PlayTomTom( iOpl, FALSE );
            end;
    end;
end;

```

```

        fm_PlayTomTom( iOpl, TRUE );
    end;

KBD_F4:
    begin
        fm_PlayChannel( iOpl, 7, FALSE );
        fm_PlaySnareDrum( iOpl, FALSE );
        fm_PlaySnareDrum( iOpl, TRUE );
    end;

KBD_F5:
    begin
        fm_PlayChannel( iOpl, 8, FALSE );
        fm_PlayTopCymbal( iOpl, FALSE );
        fm_PlayTopCymbal( iOpl, TRUE );
    end;

else
    Begin
        fm_PlayChannel( iOpl, iChannel, FALSE );
        for i := 0 to sizeof( QWERTY ) div sizeof( KEY ) - 1 do
            if UpCase( char( iParam ) ) = QWERTY[i].cKey then
                Begin
                    fm_SetChannel( iOpl, iChannel,
                                   { Channel number }
                                   (QWERTY[i].bOct + Byte (ALLCH[iOpl,iChannel].iOctave)),
                                   QWERTY[i].iNote, Byte ( ALLCH[iOpl,iChannel].iFM ),
                                   ALLCH[iOpl,iChannel].iFeedBack );
                    fm_PlayChannel( iOpl, iChannel, TRUE );

                    {-- Adapt display to new values! -----}
                    win_LoVideo( wind );
                    if iAct = OCTAVEOBJ then win_HiVideo( wind );

```



```

        old := Ints[OCTAVEINT].pValue^;
Ints[OCTAVEINT].pValue^ := QWERTY[i].bOct+Byte (CH.iOctave);
win_ObjectPrint( wind, Objects[OCTAVEOBJ] );
Ints[OCTAVEINT].pValue^ := old;
win_LoVideo( wind );

        if iAct = FRQOBJ then win_HiVideo( wind );
Ints[FRQINT].pValue^ := QWERTY[i].iNote;
win_ObjectPrint( wind, Objects[FRQOBJ] );
End;

    End;

End;

End;
{$f-}

{-----}
{-- M A I N   P R O G R A M                               --}
{-----}

Begin
    if sb_GetEnviron( SBB, GetEnv( 'BLASTER' ) ) = NO_ERROR then
        fm_SetBase( SBB, TRUE )
    else
        if fm_GetAdLib( SBB ) <> NO_ERROR then
            Begin
                Writeln( 'BLASTER environment variable not present ');
                Writeln( 'and no AdLib compatible sound card found! ');
                Halt( 0 );
            End;
        End;
    End;
End;

```

```

iOPL3 := fm_QuadroOn( TRUE );
iOPL3_4OP := FALSE;                                {-- 4 Operators possible ? --}

for j := 0 to 1 do
  Begin
    for i := 0 to 8 do
      Begin
        ALLCH[j,i].iOctave := 0;
        ALLCH[j,i].iFrequency := 0;
        ALLCH[j,i].iFM := TRUE;
        ALLCH[j,i].iFeedBack := 0;
        fm_SetChannel( j, i, ALLCH[j,i].iOctave,
                      ALLCH[j,i].iFrequency,
                      Byte ( ALLCH[j,i].iFM ),
                      ALLCH[j,i].iFeedBack );
                      { Assign channels to outputs }
        if iOPL3 then fm_ChannelLR(j,i, Boolean (j), not Boolean(j) );
      End;
    for i := 0 to 17 do
      Begin
        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 := 7;
        ALLOSC[j,i].iWave := 1;
      End;
    end for i;
  end for j;

```

```

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

    End;

End;

iOpl := 0;
iChannel := 0;
iOscillator := 0;

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

o := 0;
y := 0;
int := 0;
bool := 0;

OPLOBJ := -1;
if iOPL3 then
    Begin

```

```

OPLOBJ := o;
win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
                  'OPL no.                : ', 0, 1, @iOpl );
Inc( o );Inc( int );Inc( y );

OPL3_4OPOBJ := o;
win_ObjectInitBOOL( Objects[o], Bools[bool], 0, y, 100, 1,
                  '4 operators            : ', DT_YESNO, @iOPL3_4OP );
Inc( o );Inc( bool );Inc( y, 2 );
End;

CHANNELOBJ := o;
win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
                  'Channel                (0-8): ', 0, 8, @iChannel );
Inc( o );Inc( int );Inc( y );

OCTAVEOBJ := o;
OCTAVEINT := int;
win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
                  'Octave                (0-8): ', 0, 8, @CH.iOctave );
Inc( o );Inc( int );Inc( y );

FRQOBJ := o;
FRQINT := int;
win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
                  'Frequency            (0-1023): ', 0, 1023, @CH.iFrequency );
Inc( o );Inc( int );Inc( y );

MODEOBJ := o;
MODEY := y;
MODEINT := int;
MODEBOOL := bool;

```

```

if iOPL3_4OP then
    win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
        'Cell linking      ', 0, 3, @CH.iFM )
else
    win_ObjectInitBOOL( Objects[o], Bools[bool], 0, y, 100, 1,
        'FM synthesis      ', DT_YESNO, @CH.iFM );
Inc( o );Inc( y );Inc( bool );Inc( int );

win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
    'Feedback      (0- 3):', 0, 3, @CH.iFeedBack );
Inc( o );Inc( int );Inc( y,2 );

OSCILLATOROBJ := o;
win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
    'Oscillator      (0-17):', 0, 17, @iOscillator );
Inc( o );Inc( int );Inc( y );

win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
    'Attack          (0-15):', 0, 15, @OD.iAttack );
Inc( o );Inc( int );Inc( y );

win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
    'Decay           (0-15):', 0, 15,@OD.iDecay );
Inc( o );Inc( int );Inc( y );

win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
    'Sustain         (0-15):', 0, 15, @OD.iSustain );
Inc( o );Inc( int );Inc( y );

win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
    'Release         (0-15):', 0, 15, @OD.iRelease );

```

```

Inc( o );Inc( int );Inc( y );

win_ObjectInitBOOL( Objects[o], Bools[bool], 0, y, 100, 1,
    'Short ADSR          : ', DT_ONOFF, @OD.iShortADSR );
Inc( o );Inc( bool );Inc( y );

win_ObjectInitBOOL( Objects[o], Bools[bool], 0, y, 100, 1,
    'Cont. ADSR          : ', DT_ONOFF, @OD.iContADSR );
Inc( o );Inc( bool );Inc( y );

win_ObjectInitBOOL( Objects[o], Bools[bool], 0, y, 100, 1,
    'Vibrato             : ', DT_ONOFF, @OD.iVibrato );
Inc( o );Inc( bool );Inc( y );

win_ObjectInitBOOL( Objects[o], Bools[bool], 0, y, 100, 1,
    'Tremolo             : ', DT_ONOFF, @OD.iTremolo );
Inc( o );Inc( bool );Inc( y );

win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
    'Mute                (0-63):', 0, 63, @OD.iMute );
Inc( o );Inc( int );Inc( y );

win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
    'Hi Mute             (0- 3):', 0, 3, @OD.iHiMute );
Inc( o );Inc( int );Inc( y );

win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
    '* Frq               (0-15):', 0, 15, @OD.iFRQ );
Inc( o );Inc( int );Inc( y );
if iOPL3 then
    win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
        'Wave             (0- 7):', 0, 7, @OD.iWave )

```

```

else
    win_ObjectInitINT( Objects[o], Ints[int], 0, y, 100, 1,
                      'Wave          (0- 3):', 0, 3, @OD.iWave );
Inc( o );Inc( int );Inc( y );

MAXOBJ := 0;
win_GetScreenSettings( screen );
screenbuf := win_Save( screen );
win_Clr( screen );

win_Init( wind, 1, 1, 30, 22, $1F, $71, WIN_ACTIVE );
win_Clr( wind );

win_Init( key, 32, 1, 44, 22, $1F, $17, WIN_ACTIVE );
win_Clr( key );

win_print( key, '      Ý Ý Ò Ý Ý Ý Ò      Ý Ý Ò Ý Ý Ý Ý Ò'+#10 );
win_print( key, '      Ý Ý Ò Ý Ý Ý Ò      Ý Ý Ò Ý Ý Ý Ý Ò'+#10 );
win_print( key, '      Ò Ò Ò Ò Ò Ò Ò      Ò Ò Ò Ò Ò Ò Ò'+#10 );
win_print( key, '      Ò Ò Ò Ò Ò Ò Ò      Ò Ò Ò Ò Ò Ò Ò'+#10 );
win_print( key, '      q w e r t y u   a s d e f g h'+#10 );
win_print( key, '      1. Octave      2. Octave'+#10 );
win_print( key, '#10 );
win_print( key, 'F1 - BassDrum  F2 - HiHat  F3 - TomTom'+#10 );
win_print( key, 'F4 - SnareDrum  F5 - TopCymbal'+#10#10 );
win_print( key, 'Navigation: '+#10);
win_print( key, '* <CURSOR UP> & <CURSOR DOWN>'+#10);
win_print( key, ' Choice of setting'+#10#10 );
win_print( key, '* <+> & <->'+#10);
win_print( key, ' different value'+#10#10 );
win_print( key, '<ESC> for Exit'+#10#10 );
win_print( key, 'Micro Piano (c)'+#10+' Michael Tischer &'+#10 );

```

```
win_print( key, ' Bruno Jennrich');  
                                     { Enable percussion mode }  
fm_PercussionMode( 0, TRUE );  
fm_PercussionMode( 1, TRUE );  
  
win_ObjectProcessArray( wind, @Objects, MAXOBJ, Func );  
  
fm_Reset;                           { complete silence }  
win_Restore( screenbuf, TRUE );  
End.
```