

FMUTIL.PAS

```
{*****}
{                                     F M U T I L . P A S                                     }
{-----}
Task           : Utilities for accessing FM synthesis of
                Sound Blaster card.
{-----}
Author          : Michael Tischer / Bruno Jennrich
Developed on   : 02/06/1994
Last update    : 04/5/1995
{*****}
```

{ \$X+ } { Function results optional }

Unit FMUTIL;

Interface

Uses SBUTIL;

Const

```
ADLIB_PORT      = $388;
AL_TIMER1       = $02;
AL_TIMER2       = $03;
AL_TISTATE      = $04;
```

```
AL_TI1STARTSTOP = $01;
AL_TI2STARTSTOP = $02;
AL_TI2MASK      = $20;
AL_TI1MASK      = $40;
AL_TIRESET      = $80;
```

```
AL_STAT2OVRFLW  = $20;
{ Bits of status register }
{ Timer 2 overflow }
```

```

AL_STATIOVRFLW    =  $40;                { Timer 1 overflow }
AL_STATIRQ        =  $80;

SB_REGPORT        =  0;                   { Register port }
SB_STATUSPORT     =  0;                   { Status port }
SB_DATAPORT       =  1;                   { Data port }

                                { Oscillator numbers of percussion instruments }
BASSDRUM_A        = 12;                   { takes frequency from Channel 6 }
HIHAT              = 13;                   { takes frequency from Channel 7 }
TOMTOM            = 14;                   { takes frequency from Channel 8 }
BASSDRUM_B        = 15;                   { takes frequency from Channel 6 }
SNAREDRUM         = 16;                   { takes frequency from Channel 7 }
TOPCYMBAL         = 17;                   { takes frequency from Channel 8 }

BASS_CHANNEL      =  6;
HIHATSNARE_CHANNEL =  7;
TOMTOMCYMBAL_CHANNEL = 8;

FM_FIRSTOPL2     =  0;
FM_SECONDOPL2    =  1;

                                { Frequency parameter of standard scale }
                                { FrqParam = 1.13 * Frq          (0-1024) }
_c      = 343;                  { 262 Hz }
_cis    = 363;                  { 277 Hz }
_d      = 385;                  { 294 Hz }
_dis    = 408;                  { 311 Hz }
_e      = 432;                  { 330 Hz }
_f      = 458;                  { 349 Hz }
_fis    = 485;                  { 370 Hz }
_g      = 514;                  { 392 Hz }
_gis    = 544;                  { 415 Hz }

```

```

_a      = 577;                                { 440 Hz }
_ais    = 611;                                { 466 Hz }
_h      = 647;                                { 494 Hz }
_c2     = 686;                                { 523 Hz }

{ Tone duration as ten-thousandth of a time constant }
_l_1    = 10000;                             { whole tone      ( 10000 / 1 ) }
_l_2    = 5000;                              { half tone      ( 10000 / 2 ) }
_l_4    = 2500;                              { quarter tone   ( 10000 / 4 ) }
_l_8    = 1250;                              { eighth tone    ( 10000 / 8 ) }
_l_16   = 625;                               { sixteenth tone ( 10000 / 16 ) }
_l_32   = 312;                               { thirty-second tone ( 10000 / 32 ) }

NO_ERROR    = 0;
ERROR       = -1;

{--- Global Variables -----}
var
{ Mirroring of Sound Blaster registers }
OPL2Mirror : Array[0..1,0..255] of Byte;

FMBASE      : SBBASE; { Base data of sound card }

Const                               { Offsets of oscillators }
{-- Numbers of modulator/carrier oscillators -----}
Oscillator  : Array[0..17] of Byte =
    ( $00,$01,$02,$03,$04,$05,$08,$09,$0a,
      $0b,$0c,$0d,$10,$11,$12,$13,$14,$15 );

{-- Modulators, carriers and channels -----}
Modulator   : Array[0..8] of Byte = (0,1,2,6,7,8,12,13,14 );
Carrier     : Array[0..8] of Byte = (3,4,5,9,10,11,15,16,17 );

```

```

Channel      : Array[0..17] of Integer = ( 0, 1, 2, 0, 1, 2, 3, 4, 5,
                                           3, 4, 5, 6, 7, 8, 6, 7, 8 );

{-- Function prototypes -----}

Function  fm_Write( iOpl : integer;
                   iReg  : Integer;
                   iVal  : Byte ) : Boolean;

Function  fm_WriteBit( iOpl : integer;
                      iReg  : integer;
                      iBit  : Integer;
                      bSet  : Boolean ) : Boolean;

Procedure fm_Reset;

Procedure fm_SetBase( var SBBASE : SBBASE; iReset : Boolean);

Function  fm_GetAdLib( var SBBASE : SBBASE ) : Integer;

Function  fm_GetChannel( o_nr : Integer ) : Integer;

Function  fm_GetModulator( ch_nr : Integer ) : Integer;

Function  fm_GetCarrier( ch_nr : Integer ) : Integer;

Procedure fm_SetOscillator( iOpl      : integer;
                           iOsc      : integer;
                           bA        : byte;
                           bD        : byte;
                           bS        : byte;
                           bR        : byte;

```

```

bShortADSR : byte;
bContADSR  : byte;
bVibrato   : byte;
bTremolo    : byte;
bMute      : byte;
bHiMute    : byte;
bFrqFactor : byte;
bWave      : Byte );

```

```

Procedure fm_SetModulator( iOpl      : integer;
                           iChn      : Integer;
                           bA        : byte;
                           bD        : byte;
                           bS        : byte;
                           bR        : byte;
                           bShortADSR : byte;
                           bContADSR  : byte;
                           bVibrato   : byte;
                           bTremolo    : byte;
                           bMute      : byte;
                           bHiMute    : byte;
                           bFrqFactor : byte;
                           bWave      : byte );

```

```

Procedure fm_SetCarrier( iOpl      : integer;
                          iChn      : Integer;
                          bA        : byte;
                          bD        : byte;
                          bS        : byte;
                          bR        : byte;
                          bShortADSR : byte;
                          bContADSR  : byte;

```

```
    bVibrato    : byte;  
    bTremolo    : byte;  
    bMute       : byte;  
    bHiMute     : byte;  
    bFrqFactor  : byte;  
    bWave       : byte );
```

```
Procedure fm_SetChannel( iOpl      : integer;  
                        iChn       : Integer;  
                        bOct       : Byte;  
                        iFrq       : Integer;  
                        bFM        : byte;  
                        bFeedBack  : Byte);
```

```
Procedure fm_SetCard( iOpl : Integer; bVibrato, bTremolo : Boolean);
```

```
Procedure fm_PlayChannel( iOpl, iChn : Integer; bOn : Boolean );
```

```
Procedure fm_PlayHiHat( iOpl : Integer; bOn : Boolean );
```

```
Procedure fm_PlayTopCymbal ( iOpl : Integer; bOn : Boolean );
```

```
Procedure fm_PlayTomTom    ( iOpl : Integer; bOn : Boolean );
```

```
Procedure fm_PlaySnareDrum ( iOpl : Integer; bOn : Boolean );
```

```
Procedure fm_PlayBassDrum  ( iOpl : Integer; bOn : Boolean );
```

```
Procedure fm_PercussionMode( iOpl : Integer; bOn : Boolean );
```

```
Procedure fm_PollTime( lMilli : Longint );
```

```

Function  fm_QuadroOn( iOn : Boolean ) : Boolean;

Procedure fm_ChannelLR( iOpl, iChn : Integer; iL, iR : Boolean);

Procedure fm_QuadroChannel( iOpl : Integer; iCH0, iCH1, iCH2 : Boolean );

Procedure fm_QuadroMode( iOpl, iChn, iMode : Integer );

```

Implementation

Uses DSPUTIL;

```

{*****}
{  fm_Write: Write value to Sound Blaster registers and mirror  }
{-----}
{  Input: iOpl      -   Number of OPL2 chip (0,1) (only for OPL3)   }
{           iReg    -   Number of registers to be written         }
{           iVal    -   New register value                         }
{  Output: Success of operation                                     }
{           TRUE    -   Able to set register                       }
{           FALSE   -   Register not set (no second OPL2!)        }
{*****}
Function fm_Write( iOpl, iReg : Integer; iVal : Byte ) : Boolean;

```

```

var iOplBase : Integer;
    b         : Byte;

```

Begin

```

  if ( ( iOpl <> 0 ) and ( FMBASE.uDspVersion < DSP_3XX ) ) then
    fm_write := FALSE
  else
    { transfer register to be written }
  Begin

```

```

iOplBase := FMBASE.iDspPort;
if iOpl <> 0 then
  Inc( iOplBase, 2 );
  port[iOplBase + SB_REGPORT] := iReg;
  {-- Wait to prevent card from being "swallowed" -----}
  b := port[iOplBase + SB_REGPORT];
  b := port[iOplBase + SB_REGPORT];
  b := port[iOplBase + SB_REGPORT];
  b := port[iOplBase + SB_REGPORT];
  b := port[iOplBase + SB_REGPORT];
  b := port[iOplBase + SB_REGPORT];

  {-- transfer register contents to be written -----}
  port[iOplBase + SB_DATAPORT] := iVal;

  {-- Write register value for read accesses to memory -----}
  if iOpl <> 0 then OPL2Mirror[FM_SECNDOP2, iReg] := iVal
    else OPL2Mirror[FM_FIRSTOP2, iReg] := iVal;
  fm_write := TRUE;

```

End;

End;

```

{*****}
{  fm_WriteBit: Set/clear single bit in Sound Blaster register  }
{-----}
{  Input: iOpl      -  Number of OPL2 chip (0,1) (1: only for OPL3)  }
{          iReg     -  Number of registers to be written           }
{          iBit     -  Number of bits 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!)      }
{*****}
Function fm_WriteBit( iOpl, iReg, iBit : Integer;
                    bSet                : Boolean ) : Boolean;

var s : Byte;

Begin
    if iOpl <> 0 then iOpl := 1;
    if bSet then s := 1 else s := 0;
    fm_WriteBit := fm_Write( iOpl, iReg,
                            Byte( ( OPL2Mirror[iOpl, iReg] and
                                    ( not( 1 shl iBit ) ) ) or
                                    ( s shl iBit ) ) );
End;

{*****}
{ Reset_Card: Reset all card registers. Well defined }
{          exit state!                               }
{*****}
Procedure fm_Reset;

var i : Byte;

Begin
    for i := 0 to 255 do
        Begin
            fm_Write( FM_FIRSTOPL2, i, 0);           { 1. OPL2 }
            fm_Write( FM_SECNDOPL2, i, 0);           { 2. OPL2 }
        End;
    End;
End;

```

```

{*****}
{  fm_SetBase : Provide routines with initialized SB base          }
{                structure                                         }
{*-----*}
{  Input: SBBASE - an initialized SB base structure                }
{        iReset - if <> 0 - Reset Sound Blaster                    }
{*****}
Procedure fm_SetBase( var SBBASE : SBBASE; iReset : Boolean);

```

```

Begin
  dsp_SetBase( SBBASE, iReset );
  FMBASE := SBBASE;
  if iReset then fm_Reset;
End;

```

```

{*****}
{  fm_GetAdLib: Look for ports on AdLib compatible card           }
{*-----*}
{  Input: SBBASE - an SB base structure to be initialized         }
{  Output: NO_ERROR - AdLib compatible card found                 }
{         else      - not an AdLib card                           }
{*****}
Function fm_GetAdLib( var SBBASE : SBBASE ) : Integer;

```

```

var cx : word;

```

```

Begin
  SBBASE.iDspPort := ADLIB_PORT;           { AdLib Port $388/$389 }
  SBBASE.iMixPort := -1;
  SBBASE.iMixPort := -1;
  SBBASE.uDspVersion := $FFFF;             { will be initialized later }
  SBBASE.iDspIrq  := -1;

```

```

SBBASE.iDspDmaB := -1;
SBBASE.iDspDmaW := -1;

fm_SetBase( SBBASE, TRUE ); { Initialize base port and mirror }

fm_Write( FM_FIRSTOPL2, AL_TIMER1, $FF );      { Initial value Timer 1 }
fm_Write( FM_FIRSTOPL2, AL_TISTATE, AL_TIRESET ); { Reset timer }
fm_Write( FM_FIRSTOPL2, AL_TISTATE, AL_T11STARTSTOP ); { start }

cx := $FFFF;
while ( not(port[SBBASE.iDspPort] and AL_STAT1OVRFLW) and cx <> 0 )
do Dec( cx );

if cx <> 0 then
    fm_GetAdLib := NO_ERROR
else
    fm_GetAdLib := ERROR;
End;

{ ***** }
{ fm_GetChannel : Get channel number of an oscillator }
{ ----- }
{ Input: o_nr - Number (not offset) of oscillator }
{ Output: Number of matching channel }
{ ***** }
Function fm_GetChannel( o_nr : Integer ) : Integer;

Begin
    fm_GetChannel := Channel[o_nr];
End;

{ ***** }

```

```

{ fm_GetModulator : Get oscillator number of a channel }
-----}
{ Input: ch_nr - Number of channels }
{ Output: Oscillator number of matching modulator }
{ ***** }
Function fm_GetModulator( ch_nr : Integer ) : Integer;

```

```

Begin
    fm_GetModulator := Modulator[ch_nr];
End;

```

```

{ ***** }
{ fm_GetCarrier : Get oscillator number of a channel }
-----}
{ Input: ch_nr - Number of channels }
{ Output: Oscillator number of matching carrier }
{ ***** }
Function fm_GetCarrier( ch_nr : Integer ) : Integer;

```

```

Begin
    fm_GetCarrier := Carrier[ch_nr];
End;

```

```

{ ***** }
{ fm_SetOscillator: Set Oscillator parameter }
-----}
{ 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 parameter for frequency      }
{      bWave      - Wave form ( 0 - 3 )                         }
{      Output: none                                           }
{-----}
{      Info: Oscillator numbers <> Oscillator offsets          }
{      Frequency parameter is not a linear factor              }
{      *****}

```

```

Procedure fm_SetOscillator( iOpl      : integer;
                           iOsc      : integer;
                           bA         : byte;
                           bD         : byte;
                           bS         : byte;
                           bR         : byte;
                           bShortADSR : byte;
                           bContADSR  : byte;
                           bVibrato   : byte;
                           bTremolo   : byte;
                           bMute      : byte;
                           bHiMute    : byte;
                           bFrqFactor : byte;
                           bWave      : Byte );

```

```

begin
    { Set all single bits }
    fm_Write( iOpl, $20 + Oscillator[iOsc],
        Byte ( ( OPL2Mirror[iOpl,$20+Oscillator[iOsc]] and $F0 ) or
            bFrgFactor ) );
    fm_WriteBit( iOpl, $20 + Oscillator[iOsc], 4, Boolean ( bShortADSR ) );
    fm_WriteBit( iOpl, $20 + Oscillator[iOsc], 5, Boolean ( bContADSR ) );
    fm_WriteBit( iOpl, $20 + Oscillator[iOsc], 6, Boolean ( bVibrato ) );
    fm_WriteBit( iOpl, $20 + Oscillator[iOsc], 7, Boolean ( bTremolo ) );

    { Attack (Hi-Nibble) and Decay (Lo-Nibble) in byte }
    fm_Write( iOpl, $60 + Oscillator[iOsc],
        Byte ( bD or ( bA shl 4 ) ) );

    { Set mute }
    fm_Write( iOpl, $40 + Oscillator[iOsc],
        Byte ( bMute or ( bHiMute shl 5 ) ) );
    { Sustain (Hi-Nibble) and Release (Lo-Nibble) in one byte }
    fm_Write( iOpl, $80 + Oscillator[iOsc],
        Byte ( bR or ( bS shl 4 ) ) );

    fm_WriteBit( iOpl, $01, 5, TRUE ); { Allow wave form changes }
    fm_Write( iOpl, $E0 + Oscillator[iOsc], bWave ); { Write wave form}
    fm_WriteBit( iOpl, $01, 5, FALSE ); { Forbid wave form changes }
End;

```

```

{ ***** }
{ 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 }
-----}

```

```

Procedure fm_SetModulator( iOpl      : integer;
                          iChn      : Integer;
                          bA        : byte;
                          bD        : byte;
                          bS        : byte;
                          bR        : byte;
                          bShortADSR : byte;
                          bContADSR  : byte;
                          bVibrato   : byte;
                          bTremolo   : byte;
                          bMute      : byte;
                          bHiMute    : byte;
                          bFrqFactor : byte;
                          bWave      : Byte );

```

Begin

```

fm_SetOscillator( iOpl, Modulator[iChn], bA, bD, bS, bR,
                  bShortADSR, bContADSR, bVibrato, bTremolo,
                  bMute, bHiMute, bFrgFactor, bWave );

```

End;

```

{*****}
{ fm_SetCarrier : 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 }
      bFrgFactor - Multiplication parameter for frequency }
      bWave      - Wave form ( 0 - 3 ) }
Output: none }
{-----}
Info: Oscillator numbers != Oscillator offsets }
      Frequency parameter is not a linear factor }
{*****}

```



```

Procedure fm_SetCarrier( iOpl      : integer;
                        iChn      : Integer;
                        bA        : byte;
                        bD        : byte;
                        bS        : byte;
                        bR        : byte;
                        bShortADSR : byte;
                        bContADSR  : byte;
                        bVibrato   : byte;
                        bTremolo   : byte;
                        bMute      : byte;
                        bHiMute    : byte;
                        bFrqFactor : byte;
                        bWave      : Byte );

Begin
    fm_SetOscillator( iOpl, Carrier[iChn], bA, bD, bS, bR,
                      bShortADSR, bContADSR, bVibrato, bTremolo,
                      bMute, bHiMute, bFrqFactor, bWave );
End;

```

```

{ ***** }
{ 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 }
{ ***** }
Procedure fm_SetChannel( iOpl      : integer;

```

```

iChn      : Integer;
bOct      : Byte;
iFrq      : Integer;
bFM       : byte;
bFeedBack : Byte);

```

```
var fm : Byte;
```

```

Begin
    { Set LoByte of frequency }
    fm_Write( iOpl, $A0 + iChn,      { 3 set upper frequency bits }
              Byte ( iFrq and $FF ) ); { and octave }

    fm_Write( iOpl,      { Oscillator linking and modulator feedback }
              $B0 + iChn, Byte (((iFrq shr 8) and $3) or (bOct shl 2)));

    if( OPL2Mirror[1,5] and 1 ) <> 0 then      { OPL3 stereo mode }
        Begin
            fm_Write( iOpl, $C0 + iChn, Byte ( (OPL2Mirror[iOpl,$C0] and $F1)
                                                or (bFeedBack shl 1)));
            fm_QuadroMode( iOpl, iChn, bFM );
        End
    else
        Begin
            if bfm <> 0 then fm := 0 else fm := 1;
            fm_Write( iOpl, $C0 + iChn,
                      Byte((OPL2Mirror[iOpl, $C0] and $F0) or
                           (fm or (bFeedBack shl 1))) );
        End;
    End;
End;

{ ***** }
{ 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!
*****}
Procedure fm_SetCard( iOpl : Integer; bVibrato, bTremolo : Boolean );

Begin
  fm_WriteBit( iOpl, $BD, 6, bVibrato );
  fm_WriteBit( iOpl, $BD, 7, bTremolo );
End;

{-----}
fm_PlayChannel: Switch channel on and off
{-----}
Input: iOpl - Number of OPL2 chip (only for OPL3!)      }
      iChn   - Number of channel                        }
      bOn    - TRUE: Switch sound on, FALSE: Switch sound off }
Output: none
*****}
Procedure fm_PlayChannel( iOpl, iChn : Integer; bOn : Boolean );

Begin
  fm_WriteBit( iOpl, $B0 + iChn, 5, bOn );
End;

```

```

{ ***** }
{ Helper Functions }
{ ----- }
{ Switch rhythm mode or rhythm instrument on and off }
{ ***** }
Procedure fm_PlayHiHat( iOpl : Integer; bOn : Boolean );
Begin
    fm_WriteBit( iOpl, $BD, 0, bOn );
End;

Procedure fm_PlayTopCymbal ( iOpl : Integer; bOn : Boolean );
Begin
    fm_WriteBit( iOpl, $BD, 1, bOn );
End;

Procedure fm_PlayTomTom ( iOpl : Integer; bOn : Boolean );
Begin
    fm_WriteBit( iOpl, $BD, 2, bOn );
End;

Procedure fm_PlaySnareDrum ( iOpl : Integer; bOn : Boolean );
Begin
    fm_WriteBit( iOpl, $BD, 3, bOn );
End;

Procedure fm_PlayBassDrum( iOpl : Integer; bOn : Boolean );
Begin
    fm_WriteBit( iOpl, $BD, 4, bOn );
End;

Procedure fm_PercussionMode( iOpl : Integer; bOn : Boolean );
Begin

```

```

    fm_WriteBit( iOpl, $BD, 5, bOn );
End;

```

```

{*****}
{ fm_PollTime : Waits a specified number of thousandths of a second      }
{-----*}
{ Input : lMilli : Number of thousands to wait                          }
{*****}

```

```

Procedure fm_PollTime( lMilli : Longint );

```

```

var creg, dreg : word;

```

```

Begin

```

```

    creg := Word( lmilli shr 16 );
    dreg := Word( lmilli and $ffff );

```

```

    asm

```

```

        mov bx,1000
    @waitloop:
        mov ah,$86
        mov cx,creg
        mov dx,dreg
        int $15
        dec bx
        jne @waitloop

```

```

    End;

```

```

End;

```

```

{-- OPL3 - 4 Operator synthesis -----}

```

```

{*****}
{ fm_QuadroOn : Switch/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.          }
*****}
Function fm_QuadroOn( iOn : Boolean ) : Boolean;

```

```

Begin
    { Bit 0 in Register 5' set: second OPL2 active }
    fm_QuadroOn := fm_WriteBit( FM_SECNDOP2, $05, 0, iOn );
End;

```

```

*****}
fm_ChannelLR : Set channel to left and/or right output      }
*-----*
Input: iOpl - Number of OPL2 chip (only for OPL3!)          }
      iChn - Number of channel                              }
      iL   - != 0 : Set channel to left output              }
      iR   - != 0 : Set channel to right output             }
*-----*
Note: This function only works with an OPL3                  }
*****}
Procedure fm_ChannelLR( iOpl, iChn : Integer; iL, iR : Boolean );

```

```

Begin
    fm_WriteBit( iOpl, $C0 + iChn, 4, iL );
    fm_WriteBit( iOpl, $C0 + iChn, 5, iR );

```

End;

```
{*****}
{ 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 }
{   iCH1 != 0 : Channel 2 of OPL has 4 operators }
{   == 0 : Channel 2 of OPL has 2 operators }
{*****}
Procedure fm_QuadroChannel( iOpl : Integer; iCH0, iCH1, iCH2 : Boolean );
```

var o : Integer;

Begin

```
  if iOpl <> 0 then o := 3 else o := 0 ;
  fm_WriteBit( FM_SECNDOP2, $04, o , iCH0 );

  if iOpl <> 0 then o := 4 else o := 1;
  fm_WriteBit( FM_SECNDOP2, $04, o , iCH1 );

  if iOpl <> 0 then o := 5 else o := 2;
  fm_WriteBit( FM_SECNDOP2, $04, o , iCH2 );
```

End;

```
{*****}
{ 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!                                              }
{ ***** }

```

```

Procedure fm_QuadroMode( iOpl, iChn, iMode : Integer );

```

```

Begin

```

```

    if iChn < 3 then

```

```

        Begin

```

```

            fm_WriteBit( iOpl, $C0 + iChn, 0, Boolean ( iMode and $01 ) );

```

```

            fm_WriteBit( iOpl, $C0 + iChn + 3, 0, Boolean ( iMode and $02 ) );

```

```

        End;

```

```

End;

```

```

End.

```