

```

***** F M U T I L . P A S *****
-----
Aufgabe      : Hilfsfunktionen f•r Zugriff auf FM-Synthese der
               SoundBlaster-Karte.
-----
Autor        : Michael Tischer / Bruno Jennrich
entwickelt am : 06.02.1994
letztes Update : 8.10.1994
*****
{$X+}                                     { Funktionsergebnisse 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;
AL_STAT1OVRFLW  = $40;
AL_STATIRQ      = $80;

SB_REGPORT      = 0;
SB_STATUSPORT   = 0;
SB_DATAPORT     = 1;

{ Bits des Statusregisters }
{ Timer 2 •berlauf }
{ Timer 1 •berlauf }

{ Registerport }
{ Statusport }
{ Datenport }

{ Oszilatornummern der Percussioninstrumente }
BASSDRUM_A      = 12;
HIHAT           = 13;
TOMTOM          = 14;
BASSDRUM_B      = 15;
SNAREDRUM       = 16;
TOPCYMBAL       = 17;

{ bezieht Frequenz aus Kanal 6 }
{ bezieht Frequenz aus Kanal 7 }
{ bezieht Frequenz aus Kanal 8 }
{ bezieht Frequenz aus Kanal 6 }
{ bezieht Frequenz aus Kanal 7 }
{ bezieht Frequenz aus Kanal 8 }

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

FM_FIRSTOPL2 = 0;
FM_SECONDOPL2 = 1;

{ Frequenzparameter der Standard-Tonleiter }
{ FrqParam = 1.13 * Frq (0-1024) }
_c = 343;
_cis = 363;
_d = 385;
_dis = 408;
_e = 432;
_f = 458;
_fis = 485;
_g = 514;
_gis = 544;
_a = 577;
_ais = 611;
_h = 647;
_c2 = 686;

{ 262 Hz }
{ 277 Hz }
{ 294 Hz }
{ 311 Hz }
{ 330 Hz }
{ 349 Hz }
{ 370 Hz }
{ 392 Hz }
{ 415 Hz }
{ 440 Hz }
{ 466 Hz }
{ 494 Hz }
{ 523 Hz }

{ Tondauern als zehntausendstel einer Zeitkonstanten }
_l_1 = 10000;
_l_2 = 5000;
_l_4 = 2500;
_l_8 = 1250;
_l_16 = 625;
_l_32 = 312;

{ ganzer Ton ( 10000 / 1 ) }
{ halber Ton ( 10000 / 2 ) }
{ viertel Ton ( 10000 / 4 ) }
{ achtel Ton ( 10000 / 8 ) }
{ sechzehntel Ton ( 10000 / 16 ) }
{ zweiunddreissigstel Ton ( 10000 / 32 ) }

NO_ERROR = 0;
ERROR = -1;

{--- Globale Variablen -----}
var
{ Spiegelung der Soundblaster-Register }
OPL2Mirror : Array[0..1,0..255] of Byte;

FMBASE      : SBBASE; { Basisdaten der Soundkarte }

```

```

const { Offsets der Oszilatoren }
{-- Nummern der Modulator/Carrier Oszilatoren -----}
Oszillator : Array[0..17] of Byte =
    ( $00,$01,$02,$03,$04,$05,$08,$09,$0a,
      $0b,$0c,$0d,$10,$11,$12,$13,$14,$15 );

{-- Modulatoren, Tr„ger und Kan„le -----}
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 );

{-- Funktionsprototypen -----}

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_SetOszillator( iOpl      : integer;
                          iOsz      : 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: Schreibe Wert in Soundblaster-Register und Mirror }
{-----}
{ Eingabe: iOpl      - Nummer des OPL2-Chips (0,1) (nur bei OPL3) }
{              iReg  - Nummer des zu beschreibenden Registers }
{              iVal   - Neuer Registerwert }
{ Ausgabe: Erfolg der Operation }
{              TRUE   - Register konnte gesetzt werden }
{              FALSE  - Register nicht gesetzt (kein zweiter 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
    { zu beschreibendes Register •bermitteln }
    Begin
      iOplBase := FMBASE.iDspPort;
      if iOpl <> 0 then
        Inc( iOplBase, 2 );
        port[iOplBase + SB_REGPORT] := iReg;
        {-- Warten um "verschlucken" der Karte zu verhindern -----}
        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];

        {-- zu schreibenden Registerinhalt •bermitteln -----}
        port[iOplBase + SB_DATAPORT] := iVal;

        {-- Registerwert f•r Lesezugriffe in Speicher schreiben -----}
        if iOpl <> 0 then OPL2Mirror[FM_SECNDOPPL2, iReg] := iVal
          else OPL2Mirror[FM_FIRSTOPL2, iReg] := iVal;
        fm_write := TRUE;
      End;
    End;

```

```

{*****}
{ fm_WriteBit: Einzelnes Bit in Soundblaster-Register setzen/l"schen }
{-----}
{ Eingabe: iOpl      - Nummer des OPL2-Chips (0,1) (1: nur bei OPL3) }
{              iReg  - Nummer des zu beschreibenden Registers }
{              iBit   - Nummer des zu ndernden Bits (0-7) }
{              bSet   - TRUE: Bit Setzen }
{                   FALSE: Bit L"schen }
{ Ausgabe: Erfolg der Operation }
{              TRUE   - Register konnte gesetzt werden }
{              FALSE  - Register nicht gesetzt (kein zweiter 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: Setze alle Kartenregister zur•ck. Wohldefinierter aus- }
{ gangszustand! }
{*****}
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;

{*****}
{ fm_SetBase : Routinen mit initialisierter SBBASE-Struktur }
{ versorgen }
{*****}
{-----*}
{ Eingabe: SBBASE - eine initialisierte SBBASE-Struktur }
{ iReset - wenn <> 0 - Soundblaster resettet }
{*****}
Procedure fm_SetBase( var SBBASE : SBBASE; iReset : Boolean);

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

{*****}
{ fm_GetAdLib: Ports auf AdLib kompatible Karte durchsuchen }
{-----*}
{ Eingabe: SBBASE - eine zu initialisierenden SBBASE-Struktur }
{ Ausgabe: NO_ERROR - AdLib kompatible Karte gefunden }
{ sonst - keine AdLib Karte }
{*****}
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;                 { wird sp„ter initialisiert }
  SBBASE.iDspIrq := -1;
  SBBASE.iDspDmaB := -1;
  SBBASE.iDspDmaW := -1;

  fm_SetBase( SBBASE, TRUE ); { Basisport und Mirror initialisieren }

  fm_Write( FM_FIRSTOPL2, AL_TIMER1, $FF );    { Startwert Timer 1 }
  fm_Write( FM_FIRSTOPL2, AL_TISTATE, AL_TIRESET ); { Timer resettet }
  fm_Write( FM_FIRSTOPL2, AL_TISTATE, AL_TI1STARTSTOP ); { starten }

  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 : Kanalnummer eines Oszillators ermitteln }
{-----*}
{ Eingabe: o_nr - Nummer (nicht Offset) des Oszillators }
{ Ausgabe: Nummer des zugeh„rigen Kanals }
{*****}

```

```

Function fm_GetChannel( o_nr : Integer ) : Integer;
Begin
    fm_GetChannel := Channel[o_nr];
End;

{*****}
{ fm_GetModulator : Oszillatornummer eines Kanals ermitteln }
{-----}
{ Eingabe: ch_nr - Nummer des Kanals }
{ Ausgabe: Oszillator-Nummer des zugehörigen Modulators }
{*****}
Function fm_GetModulator( ch_nr : Integer ) : Integer;
Begin
    fm_GetModulator := Modulator[ch_nr];
End;

{*****}
{ fm_GetCarrier : Oszillatornummer eines Kanals ermitteln }
{-----}
{ Eingabe: ch_nr - Nummer des Kanals }
{ Ausgabe: Oszillator-Nummer des zugehörigen Carriers }
{*****}
Function fm_GetCarrier( ch_nr : Integer ) : Integer;
Begin
    fm_GetCarrier := Carrier[ch_nr];
End;

{*****}
{ fm_SetOszillator: Setze Oszillatorparameter }
{-----}
{ Eingabe: iOpl      - Nummer des OPL2-Chips (nur bei OPL3!) }
{ iOsz              - Nummer (nicht Offset) des Oszillators }
{ bA                - Attack / Anschwellen ( 0 - 15 ) }
{ bD                - Decay / Abschwellen ( 0 - 15 ) }
{ bS                - Sustain / Halten ( 0 - 15 ) }
{ bR                - Release / Ausklingen ( 0 - 15 ) }
{ bShortADSR        - Hüllkurvenverkürzung mit zunehmender Oktave }
{ bContADSR         - kontinuierliche Hüllkurve an/aus }
{ bVibrato          - Vibrato an/aus }
{ bTremolo          - Tremolo an/aus }
{ bMute             - Dämpfung ( 0: laut, 63: leise ) }
{ bHiMute           - Hochtondämpfung }
{                   0: 0dB pro Oktave }
{                   1: 3dB pro Oktave }
{                   2: 1.5dB pro Oktave }
{                   3: 6dB pro Oktave }
{ bFrgFactor        - Multiplikationsparameter für Frequenz }
{ bWave            - Wellenform ( 0 - 3 ) }
{ Ausgabe: keine }
{-----}
{ Info: Oszillatornummern <> Oszillatoroffsets }
{       Frequenzparameter ist kein linearer Faktor }
{*****}

Procedure fm_SetOszillator( iOpl      : integer;
                           iOsz      : integer;
                           bA        : byte;
                           bD        : byte;
                           bS        : byte;
                           bR        : byte;
                           bShortADSR : byte;
                           bContADSR  : byte;
                           bVibrato   : byte;
                           bTremolo   : byte;
                           bMute      : byte;
                           bHiMute    : byte;
                           bFrgFactor : byte;
                           bWave      : Byte );
begin
    { Alle Einzel-Bits setzen }
    fm_Write( iOpl, $20 + Oszillator[iOsz],
              Byte ( ( OPL2Mirror[iOpl,$20+Oszillator[iOsz]] and $F0 ) or
                    bFrgFactor ) );
    fm_WriteBit( iOpl, $20 + Oszillator[iOsz], 4, Boolean ( bShortADSR ) );
    fm_WriteBit( iOpl, $20 + Oszillator[iOsz], 5, Boolean ( bContADSR ) );
    fm_WriteBit( iOpl, $20 + Oszillator[iOsz], 6, Boolean ( bVibrato ) );
    fm_WriteBit( iOpl, $20 + Oszillator[iOsz], 7, Boolean ( bTremolo ) );

    { Attack (Hi-Nibble) und Decay (Lo-Nibble) in Byte }
    fm_Write( iOpl, $60 + Oszillator[iOsz],
              Byte ( bD or ( bA shl 4 ) ) );

    { Dämpfung setzen }
    fm_Write( iOpl, $40 + Oszillator[iOsz],

```

```

        Byte ( bMute or bHiMute shl 5 ) );
        { Sustain (Hi-Nibble) und Release (Lo-Nibble) in ein Byte }
fm_Write( iOpl, $80 + Oszillator[iOsz],
        Byte ( bR or ( bS shl 4 ) ) );

fm_WriteBit( iOpl, $01, 5, TRUE ); { Wellenform •nderungen erlauben }
fm_Write( iOpl, $E0 + Oszillator[iOsz], bWave ); { Wellenform schreiben }
fm_WriteBit( iOpl, $01, 5, FALSE ); { Wellenform •nderungen verbieten }
End;

{*****}
{ fm_SetModulator : Modulator eines Kanals programmieren }
{-----}
{ Eingabe: iOpl      - Nummer des OPL2-Chips (nur bei OPL3!) }
{ iChn             - Kanalnummer }
{ bA               - Attack / Anschwellen ( 0 - 15 ) }
{ bD               - Decay / Abschwellen ( 0 - 15 ) }
{ bS               - Sustain / Halten ( 0 - 15 ) }
{ bR               - Release / Ausklingen ( 0 - 15 ) }
{ bShortADSR       - H•llkurvenverk•rzung mit zunehmender Oktave }
{ bContADSR        - kontinuierliche H•llkurve an/aus }
{ bVibrato         - Vibrato an/aus }
{ bTremolo         - Tremolo an/aus }
{ bMute            - D•mpfung ( 0: laut, 63: leise ) }
{ bHiMute          - Hochtond•mpfung }
{                  0: 0dB pro Oktave }
{                  1: 3dB pro Oktave }
{                  2: 1.5dB pro Oktave }
{                  1: 6dB pro Oktave }
{ bFrqFactor       - Multiplikationsparameter f•r Frequenz }
{ bWave            - Wellenform ( 0 - 3 ) }
{ Ausgabe: keine }
{-----}

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

{*****}
{ fm_SetCarrier : Carrier / Tr•ger eines Kanals programmieren }
{-----}
{ Eingabe: iOpl      - Nummer des OPL2-Chips (nur bei OPL3!) }
{ iChn             - Kanalnummer }
{ bA               - Attack / Anschwellen ( 0 - 15 ) }
{ bD               - Decay / Abschwellen ( 0 - 15 ) }
{ bS               - Sustain / Halten ( 0 - 15 ) }
{ bR               - Release / Ausklingen ( 0 - 15 ) }
{ bShortADSR       - H•llkurvenverk•rzung mit zunehmender Oktave }
{ bContADSR        - kontinuierliche H•llkurve an/aus }
{ bVibrato         - Vibrato an/aus }
{ bTremolo         - Tremolo an/aus }
{ bMute            - D•mpfung ( 0: laut, 63: leise ) }
{ bHiMute          - Hochtond•mpfung }
{                  0: 0dB pro Oktave }
{                  1: 3dB pro Oktave }
{                  2: 1.5dB pro Oktave }
{                  1: 6dB pro Oktave }
{ bFrqFactor       - Multiplikationsparameter f•r Frequenz }
{ bWave            - Wellenform ( 0 - 3 ) }
{ Ausgabe: keine }
{-----}
{ Info: Oszilatornummern != Oszilatoroffsets }
{       Frequenzparameter ist kein linearer Faktor }
{*****}

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

{ ***** }
{ fm_SetChannel: Setze Kanal }
{ ----- }
{ Eingabe: iOpl      - Nummer des OPL2-Chips (nur bei OPL3!) }
{           iChn     - Nummer (UND Offset) des Kanals }
{           bOct      - Oktave (0 - 8) }
{           iFrq      - Frequenz (0 - 1023) }
{           bFM       - FM-Synthese oder addierte Oszilatoren }
{           bFeedBack - Modulator-Rückkopplung }
{ Ausgabe: keine }
{ ***** }
Procedure fm_SetChannel( iOpl      : integer;
  iChn      : Integer;
  bOct      : Byte;
  iFrq      : Integer;
  bFM       : byte;
  bFeedBack : Byte);

var fm : Byte;

Begin
  { LoByte der Frequenz setzen }
  fm_Write( iOpl, $A0 + iChn, { 3 obere Frequenz-Bits }
    Byte ( iFrq and $FF ) ); { und Oktave setzen }

  fm_Write( iOpl, { Oszillator-Verknüpfung und Modulator-Feedback }
    $B0 + iChn, Byte ( ((iFrq shr 8) and $3) or (bOct shl 2) ));

  if( OPL2Mirror[1,5] and 1 ) <> 0 then { OPL3-Stereomodus }
    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;

{ ***** }
{ fm_SetCard: Soundkarten Parameter setzen }
{ ----- }
{ Eingabe: iOpl      - Nummer des OPL2-Chips (nur bei OPL3!) }
{           bVibrato - TRUE: Vibratotiefe = 14/100 der Kanalfrequenz }
{           FALSE: Tremolotiefe = 7/100 der Kanalfrequenz }
{           bTremolo - TRUE: Tremolotiefe = 4.8 dB }
{           FALSE: Tremolotiefe = 1 dB }
{ Ausgabe: keine }
{ ----- }
{ Info: Tremolo- und Vibratotiefe nur f•r alle Oszilatoren einstell- }
{       bar! }
{ ***** }
Procedure fm_SetCard( iOpl : Integer; bVibrato, bTremolo : Boolean );

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

{ ***** }
{ fm_PlayChannel: Kanal an- und abschalten }
{ ----- }
{ Eingabe: iOpl - Nummer des OPL2-Chips (nur bei OPL3!) }
{           iChn - Nummer des Kanals }
{           bOn  - TRUE: Ton anschalten, FALSE: Ton abschalten }
{ Ausgabe: keine }
{ ***** }
Procedure fm_PlayChannel( iOpl, iChn : Integer; bOn : Boolean );

```

```

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

{*****}
{  Helper-Functions  }
{-----}
{  Schalten Rythmusmodus bzw. Rythmusinstrument an und ab  }
{*****}
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 : Wartet angebene Anzahl von tausendstel Sekunden  }
{-----*}
{  Eingabe : lMilli : Anzahl abzuwartender Tausendstel  }
{*****}
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-Synthese -----}

{*****}
{  fm_QuadroOn : Stereobetrieb der OLPs umschalten  }
{-----*}
{  Eingabe : iOn - TRUE : zweiten OPL2 aktivieren (OPL3-Modus)  }
{             FALSE : zweiten OPL2 deaktivieren (Kompatibili-  }
{                   t„tsmodus)  }
{  Ausgabe : TRUE - zweiter OPL2 vorhanden  }
{             FALSE - kein zweiter OPL2  }
{-----*}
{  Hinweis: Innerhalb der Funktion fm_Write(Bit) wird festgestellt,  }
{           ob der zweite OPL2 •berhaupt angesprochen wertden kann,  }
{           bzw. ob die Soundblasterkarte •ber einen OPL3 verf•gt.  }
{*****}
Function fm_QuadroOn( iOn : Boolean ) : Boolean;

Begin
    { Bit 0 in Register 5' gesetzt: zweiter OPL2 aktiv }
    fm_QuadroOn := fm_WriteBit( FM_SECNDOP2, $05, 0, iOn );
End;

```



```

{*****}
{ fm_ChannelLR : Kanal auf linken und/oder rechten Ausgang legen }
{*****}
{
  Eingabe: iOpl - Nummer des OPL2-Chips (nur bei OPL3!)
           iChn - Nummer des Kanals
           iL   - != 0 : Kanal auf linken Ausgang legen
           iR   - != 0 : Kanal auf rechten Ausgang legen
}
{*****}
{ Hinweis: Diese Funktion funktioniert nur mit einem 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: Anzahl der Operatoren eines Kanals festlegen }
{*****}
{
  Eingabe: iOpl - Nummer des OPL2-Chips (nur bei OPL3!)
           iCH0 != 0 : Kanal 0 des OPL's hat 4 Operatoren
           == 0 : Kanal 0 des OPL's hat 2 Operatoren
           iCH1 != 0 : Kanal 1 des OPL's hat 4 Operatoren
           == 0 : Kanal 1 des OPL's hat 2 Operatoren
           iCH2 != 0 : Kanal 2 des OPL's hat 4 Operatoren
           == 0 : Kanal 2 des OPL's hat 2 Operatoren
}
{*****}
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 : Zellenverknüpfung für 4 Operatoren Kanäle fest- }
{ legen. }
{*****}
{
  Eingabe: iOpl - Nummer des OPL2-Chips (nur bei OPL3!)
           iChn - Nummer des Kanals (0-2)
           iMode - Verknüpfungsmodus (0-3)
}
{*****}
{ Hinweis: Der Modus wird auf die beiden "alten" Verknüpfungsbits }
{ der beiden 2 Operator-Kanäle eines 4-Operator Kanals ver- }
{ teilt! }
{*****}
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.

```