

```

*****
M I X U T I L . P A S
*****
}
{ Task      : Provides functions for programming the
              Sound Blaster mixer.
}
*****
{ Author      : Michael Tischer / Bruno Jennrich
{ Developed on : 03/20/1994
{ Last update  : 04/06/1995
}
*****
{ Info      : mix_??? - Applies to all mixer versions
              mix3_??? - ... for mixer CT1345 ( DSP V3.00 and higher )
              mix4_??? - ... for mixer CT1745 ( DSP V4.00 and higher )
}
*****
Unit MIXUTIL;

Interface

Uses SBUTIL;

Const
  ERROR      = $ffff;
  NO_ERROR   = 0;

  { Mixer port offsets }
  MIX_REGISTERPORT = $04; { Setting register... }
  MIX_DATAPORT     = $05; { Read/write data }

  { CT1335 mixer (in DSP 2.xx) }
  MIX_RSET         = $00; { Reset mixer }
  MIX2_MASTERVOL   = $02; { Total-Volume (BITS 3,2,1) }
  MIX2_MIDIVOL     = $06; { MIDI(FM) Volume (BITS 3,2,1) }
  MIX2_CDVOL       = $08; { CD Volume (BITS 3,2,1) }
  MIX2_VOICEVOL    = $0A; { VOICE(DSP) Volume (BITS 2,1) }

  { CT1345 mixer (in DSP 3.xx ) }
  MIX3_VOICEVOL    = $04; { VOICE-Vol. (Bits 7,6,5 3,2,1) }
  MIX3_MICVOL      = $0A; { Microphone Volume (BITS 2,1) }

  { DSP input and filter }
  MIX3_ADCSTATE    = $0C; { Input filter }
  MIX3_ADCFILTEROFF= $20; { Bit 5: Low pass filter on }
  MIX3_LOWPASS88   = $08; { Bit 3: 8.8 kHz Low pass filter }
  { Input source }
  MIX3_MICSRC      = $00; { BIT 2,1 = 00: Microphone }
  MIX3_MICSRC_     = $04; { BIT 2,1 = 10: also Microphone }
  MIX3_CDSRC       = $02; { BIT 2,1 = 01: CD }
  MIX3_LINESRC     = $06; { BIT 2,1 = 11: LINE IN }
  MIX3_SRCMSK      = $06; { Bit mask of source bits }

  { Output filter and stereo switch }
  MIX3_DACSTATE    = $0E; { Bit 5: Low-Pass Filter an (s.o.) }
  MIX3_DACFILTEROFF= $20; { Bit 1: Stereo output }
  MIX3_STEREOON    = $02;

  { Totalvol. (BITS 7,6,5 and 3,2,1) }
  MIX3_MASTERVOL   = $22; { MIDI-Vol. (BITS 7,6,5 and 3,2,1) }
  MIX3_MIDIVOL     = $26; { CD-Vol (BITS 7,6,5 and 3,2,1) }
  MIX3_CDVOL       = $28; { LINE-Vol. (BITS 7,6,5 and 3,2,1) }
  MIX3_LINEVOL     = $2E;

  { CT1745 Mixer (in DSP 4.xx) as well as ASP }
  { Volumes: }
  MIX4_MASTERVOL_L = $30; { Total left (BITS 7,6,5,4,3) }
  MIX4_MASTERVOL_R = $31; { Total right (BITS 7,6,5,4,3) }
  MIX4_VOICEVOL_L  = $32; { VOICE(DSP) left (BITS 7,6,5,4,3) }
  MIX4_VOICEVOL_R  = $33; { VOICE(DSP) right (BITS 7,6,5,4,3) }
  MIX4_MIDIVOL_L   = $34; { MIDI left (BITS 7,6,5,4,3) }
  MIX4_MIDIVOL_R   = $35; { MIDI right (BITS 7,6,5,4,3) }
  MIX4_CDVOL_L     = $36; { CD left (BITS 7,6,5,4,3) }
  MIX4_CDVOL_R     = $37; { CD right (BITS 7,6,5,4,3) }
  MIX4_LINEVOL_L   = $38; { LINE left (BITS 7,6,5,4,3) }
  MIX4_LINEVOL_R   = $39; { LINE right (BITS 7,6,5,4,3) }
  MIX4_MICVOL      = $3A; { Microphone (BITS 7,6,5,4,3 ) }
  MIX4_PCSPEAKERVOL= $3B; { PC-Speaker (BITS 7,6 ) }

  { Output sources }
  MIX4_OUTSOURCE   = $3C; { Sample sources L/R }
  MIX4_ADCSOURCE_L = $3D;
  MIX4_ADCSOURCE_R = $3E;

  { Active recording sources }
  MIX4_MIDI_L      = $40;
  MIX4_MIDI_R      = $20 ;
  MIX4_LINE_L      = $10;
  MIX4_LINE_R      = $08;
  MIX4_CD_L        = $04;
  MIX4_CD_R        = $02;
  MIX4_MIC          = $01;

```

```

{ Preamplifier for output(OUT) and sampling(ADC) (Bits 7 und 6 ) }
MIX4_ADCGAIN_L = $3F;
MIX4_ADCGAIN_R = $40;
MIX4_OUTGAIN_L = $41;
MIX4_OUTGAIN_R = $42;

MIX4_AGC = $43; { Microphone preamplifier (20dB) }
MIX4_AGCN = $01; { Microphone preamplifier on }

{ Treble and bass of preamplifier (BITS 7,6,5,4 ) }
MIX4_TREBLE_L = $44;
MIX4_TREBLE_R = $45;
MIX4_BASS_L = $46;
MIX4_BASS_R = $47;

{ Which interrupts and DMA lines are being used ? }
MIX4_IRQ = $80;
MIX4_IRQ2 = $01; { 4 possible interrupt lines }
MIX4_IRQ5 = $02;
MIX4_IRQ7 = $04;
MIX4_IRQ10 = $08;

MIX4_DMA = $81; {- Which DMA line is being used ? }
MIX4_DMA0 = $01;
MIX4_DMA1 = $02;
{MIX4_DMA2 = $04 disk drive }
MIX4_DMA3 = $08;
{MIX4_DMA4 = $10 Cascading }
MIX4_DMA5 = $20;
MIX4_DMA6 = $40;
MIX4_DMA7 = $80;

MIX4_IRQSOURCE = $82;
MIX4_IRQ8DMA = $01; { Interrupt of 8 bit DMA and Midi }
MIX4_IRQ16DMA = $02; { Interrupt of 16 bit DMA }
MIX4_IRQMPU = $04; { Interrupt of MPU }

{ MIXUTIL constants }
{ The program uses constants for accessing volume arrays. These constants }
{ specify whether a port address refers to the right channel, left channel }
{ or both channels. }
SBPORT = 0; { For array access: 1st element = port }
CHANNEL = 1; { 2nd element = access code }

L = 0; { left }
R = 1; { right }

CH_LEFT = 1; { Code for left channel }
CH_RIGHT = 2; { right channel }
CH_BOTH = 3; { both channels }
CH_NONE = 0; { no channel }
CH_MAX = 6; { use maximum of right and left }

{ Number available sources consecutively }
CD = 0;
LINE = 1;
VOICE = 2;
MASTER = 3;
MIDI = 4;
MIC = 5;
PCSPEAKER = 6;
NUM_SOURCES = 7;

CD_L = 7;
LINE_L = 8;
VOICE_L = 9;
MASTER_L = 10;
MIDI_L = 11;
CD_R = 12;
LINE_R = 13;
VOICE_R = 14;
MASTER_R = 15;
MIDI_R = 16;

MAX_SRC = 17;

DAC = TRUE; { for 'mix3_PrepereForStereo' }
ADC = FALSE;

{-- Globale Module Variables -----}
var
MIXBASE : SBBASE; { Port address of mixer }

bMix3DACStereo, { The program uses these variables }
bMix3DACFilter, { to save some mixer values. }
bMix3ADCFilter : Byte;

```

```

bMix4SourceL,
bMix4SourceR    : Byte;

{ The following tables are used for setting the volume.      }
{ The program notes the register and volume to be set (left, right, both) }
{ for each possible source.                                   }

Const
bMix3VolTable : Array[0..MAX_SRC - 1,0..1] of Byte =
    ( ( MIX3_CDVOL, CH_BOTH ),
      ( MIX3_LINEVOL, CH_BOTH ),
      ( MIX3_VOICEVOL, CH_BOTH ),
      ( MIX3_MASTERVOL, CH_BOTH ),
      ( MIX3_MIDIVOL, CH_BOTH ),
      ( MIX3_MICVOL, CH_MAX ),
      ( 0, 0 ), { Speaker }

      ( MIX3_CDVOL, CH_LEFT ),
      ( MIX3_LINEVOL, CH_LEFT ),
      ( MIX3_VOICEVOL, CH_LEFT ),
      ( MIX3_MASTERVOL, CH_LEFT ),
      ( MIX3_MIDIVOL, CH_LEFT ),

      ( MIX3_CDVOL, CH_RIGHT ),
      ( MIX3_LINEVOL, CH_RIGHT ),
      ( MIX3_VOICEVOL, CH_RIGHT ),
      ( MIX3_MASTERVOL, CH_RIGHT ),
      ( MIX3_MIDIVOL, CH_RIGHT ) );

bMix4VolTable : array[0..MAX_SRC - 1,0..1] of Byte =
    ( ( MIX4_CDVOL_L, CH_BOTH ),
      ( MIX4_LINEVOL_L, CH_BOTH ),
      ( MIX4_VOICEVOL_L, CH_BOTH ),
      ( MIX4_MASTERVOL_L, CH_BOTH ),
      ( MIX4_MIDIVOL_L, CH_BOTH ),
      ( MIX4_MICVOL, CH_LEFT ),
      ( MIX4_PCSPEAKERVOL, CH_LEFT ),

      ( MIX4_CDVOL_L, CH_LEFT ),
      ( MIX4_LINEVOL_L, CH_LEFT ),
      ( MIX4_VOICEVOL_L, CH_LEFT ),
      ( MIX4_MASTERVOL_L, CH_LEFT ),
      ( MIX4_MIDIVOL_L, CH_LEFT ),

      ( MIX4_CDVOL_L, CH_RIGHT ),
      ( MIX4_LINEVOL_L, CH_RIGHT ),
      ( MIX4_VOICEVOL_L, CH_RIGHT ),
      ( MIX4_MASTERVOL_L, CH_RIGHT ),
      ( MIX4_MIDIVOL_L, CH_RIGHT ) );

{ The CT1745 mixer allows users to enable and disable several      }
{ recording and playback sources. The following array notes      }
{ the appropriate switch bits for each potential source.          }

bMix4SourceBits : Array[0..MAX_SRC - 1] of Byte =
{ CD }                ( MIX4_CD_L or MIX4_CD_R,
{ LINE }              MIX4_LINE_L or MIX4_LINE_R,
{ VOICE ( not switchable ) } 0,
{ MASTER ( not switchable ) } 0,
{ MIDI }              MIX4_MIDI_L or MIX4_MIDI_R,
{ MIC }              MIX4_MIC,
{ PCSPEAKER ( not switchable ) } 0,

{ CD_L }              MIX4_CD_L,
{ LINE_L }            MIX4_LINE_L,
{ VOICE_L }           0,
{ MASTER_L }          0,
{ MIDI_L }            MIX4_MIDI_L,

{ CD_R }              MIX4_CD_R,
{ LINE_R }            MIX4_LINE_R,
{ VOICE_R }           0,
{ MASTER_R }          0,
{ MIDI_R }            MIX4_MIDI_R );

{-- Prototypes -----}

Procedure mix_Reset;

Function mix_SetBase( var sbbase : SBBASE; iReset:Boolean ) : Word;

Procedure mix_Write( iReg, iData : Word);

Function mix_Read( iReg : Word ) : Word;

```

```

Procedure mix3_SetADCFilter( iState : Boolean );
Function mix3_GetADCFilter : Boolean;
Procedure mix3_SetDACFilter( iState : Boolean );
Function mix3_GetDACFilter : Boolean;
Procedure mix3_SetDACStereo( iState : Boolean );
Function mix3_GetDACStereo : Boolean;
Procedure mix3_SetADDACLowPass( iState : Boolean );
Function mix3_GetADDACLowPass : Boolean;
Procedure mix3_PrepareForStereo( iMode : Boolean );
Procedure mix3_RestoreFromStereo;
Procedure mix3_SetVolume( iSource, iVolL, iVolR : Word );
Function mix3_GetVolume( iSource : Word ) : Word;
Procedure mix3_SetADCSource( iSource : Word );
Function mix3_GetADCSource : Word;
Procedure mix4_PrepareForMonoADC;
Procedure mix4_RestoreFromMonoADC;
Procedure mix4_SetVolume( iSource, iVolL, iVolR : Word );
Function mix4_GetVolume( iSource : Word ) : Word;
Procedure mix4_SetADCSourceL( iSource : Word; iState : Boolean );
Procedure mix4_SetADCSourceR( iSource : Word; iState : Boolean );
Function mix4_GetADCSourceL( iSource : Word ) : Word;
Function mix4_GetADCSourceR( iSource : Word ) : Word;
Procedure mix4_SetOUTSource( iSource : Word; iState : Boolean );
Function mix4_GetOUTSource( iSource : Word ) : Word;
Procedure mix4_SetADCGain( iGainL, iGainR : Word);
Function mix4_GetADCGain( iChannel : Word ) : Word;
Procedure mix4_SetOUTGain( iGainL, iGainR : Word);
Function mix4_GetOUTGain( iChannel : Word ) : Word;
Procedure mix4_SetAGC( iState : Word );
Function mix4_GetAGC : Boolean;
Procedure mix4_SetTreble( iTrebleL, iTrebleR : Word);
Function mix4_GetTreble( iChannel : Word ) : Word;
Procedure mix4_SetBass( iBassL, iBassR : Word );
Function mix4_GetBass( iChannel : Word ) : Word;

```

Implementation

```

{*****}
{ mix_Reset : Reset mixer }
{*****}
Procedure mix_Reset;

Begin
    mix_Write( MIX_RSET, 0 );
End;

{*****}
{ mix_SetBase : Sets port address }
{*****}
{ *-----* }
{ Input : SBBase - an initialised SB base structure }
{ iReset - <> 0 : Mixer to be reset }
{ *-----* }
{ Info : - Before any of the following functions can be called, }

```

```

}
    it's necessary to set an SBBASE structure
    with information about the installed
    sound card with the help of this function!
}
*****
Function mix_SetBase( var sbbase : SBBASE; iReset : Boolean ) : Word;

Begin
    if sbbase.iMixPort <> -1 then
        Begin
            MIXBASE := sbbase; { Note data in TPU structure }
            if iReset then mix_Reset;
            mix_SetBase := NO_ERROR;
        End
    else
        mix_SetBase := ERROR;
End;

{ *****
{ mix_Write : Send byte to mixer register
{ -----*
{ Input : iReg - Number of register to be changed
{ iData - Value to be written in the register
{ *****
Procedure mix_Write( iReg, iData : Word);
Begin
    port[MIXBASE.iMixPort + { Specify register to be written }
        MIX_REGISTERPORT] := Byte ( iReg );

    port[MIXBASE.iMixPort + { Send new register value }
        MIX_DATAPORT] := Byte ( iData );
End;

{ *****
{ mix_Read : Read byte from mixer register
{ -----*
{ Input : iReg - Number of register to be read
{ Output : Value of the mixer register
{ *****
Function mix_Read( iReg : Word ) : Word;
Begin
    port[MIXBASE.iMixPort + { Specify register to be read }
        MIX_REGISTERPORT] := Byte ( iReg );

    mix_Read := port[MIXBASE.iMixPort + { Get register value }
        MIX_DATAPORT];
End;

{ *****
{ mix3_SetADCFilter : Switch ADC filter on or off
{ -----*
{ Input : iState = FALSE: Filter off
{ = TRUE: Filter on
{ -----*
{ Info : - The bit that controls the filter is an
{ OFF switch. Therefore, if the bit is set, the filter
{ is off!
{ ADC = Analog/Digital Converter (recording)
{ *****
Procedure mix3_SetADCFilter( iState : Boolean );

var bIn : Byte;

Begin
    bIn := Byte ( mix_Read( MIX3_ADCSTATE ) ); { read current register value }

    {-- Set / clear filter bit (LOW-Active) -----}
    if iState then
        mix_Write( MIX3_ADCSTATE, bIn and not MIX3_ADCFILTEROFF )
    else
        mix_Write( MIX3_ADCSTATE, bIn or MIX3_ADCFILTEROFF );
End;

{ *****
{ mix3_GetADCFilter : Get status of ADC filter
{ -----*
{ Output := FALSE - Filter off
{ TRUE - Filter on
{ *****
Function mix3_GetADCFilter : Boolean;

Begin
    { Warning! Filter bit is LOW active }
    mix3_GetADCFilter := ( ( mix_Read( MIX3_ADCSTATE ) and
        MIX3_ADCFILTEROFF ) = 0 );
End;

```

```

*****
{ mix3_SetDACFilter : Switch DAC filter on or off
}
-----*
{ Input : iState - FALSE : Filter off
               TRUE  : Filter on
}
-----*
{ Info :      DAC : Digital/Analog Converter (playback)
}
*****
Procedure mix3_SetDACFilter( iState : Boolean );

var Bin : Byte;

Begin
    bIn := Byte ( mix_Read( MIX3_DACSTATE ) );{ read current register value }

    {-- Set/clear filter bit (Filter bit is LOW active) -----}
    if iState then
        mix_Write( MIX3_DACSTATE, bIn and not MIX3_DACFILTEROFF )
    else
        mix_Write( MIX3_DACSTATE, bIn or MIX3_DACFILTEROFF );
End;

*****
{ mix3_GetDACFilter : Get status of DAC filter
}
-----*
{ Output := FALSE - Filter off
               TRUE  - Filter on
}
*****
Function mix3_GetDACFilter : Boolean;

Begin
    { Warning! Filter bit is LOW active }
    mix3_GetDACFilter := ( ( mix_Read( MIX3_DACSTATE ) and
                           MIX3_DACFILTEROFF ) = 0 );
End;

*****
{ mix3_SetDACStereo : Stereo playback (DAC) on or off
}
-----*
{ Input : iState - FALSE: Stereo playback off => Mono
               TRUE  : Stereo playback on
}
*****
Procedure mix3_SetDACStereo( iState : Boolean );

var bIn : Byte;

Begin
    bIn := Byte ( mix_Read( MIX3_DACSTATE ) );{ read current register value }
    if iState then
        mix_Write( MIX3_DACSTATE,bIn or MIX3_STEREOON )
    else
        mix_Write( MIX3_DACSTATE,bIn and not MIX3_STEREOON );
End;

*****
{ mix3_GetDACStereo : Get status of stereo playback
}
-----*
{ Output := FALSE - Stereo playback off
               TRUE  - Stereo playback on
}
*****
Function mix3_GetDACStereo : Boolean;

Begin
    mix3_GetDACStereo := ( ( mix_Read( MIX3_DACSTATE ) and
                           MIX3_STEREOON ) <> 0 );
End;

*****
{ mix3_SetADDACLowPass : Set low pass filter
}
-----*
{ Input : iState = FALSE: 3.2 kHz low pass filter
               TRUE:   8.8 kHz low pass filter
}
-----*
{ Info : - The low pass filter is only used when the ADC
           or DAC filter is enabled. Otherwise the
           low pass filter is ignored during recording and playback.
}
*****
Procedure mix3_SetADDACLowPass( iState : Boolean );

var Bin : Byte;

Begin
    bIn := Byte ( mix_Read( MIX3_ADCSTATE ) );{ read current register value }
    if iState then
        mix_Write( MIX3_ADCSTATE, bIn or MIX3_LOWPASS88 )
    else
        mix_Write( MIX3_ADCSTATE, bIn and not MIX3_LOWPASS88 );

```

```

End;

{*****}
{ mix3_GetADDACLowPass : Get low pass filter being used }
{*****}
{-----*}
{ Output := FALSE: 3.2 kHz low pass filter being used }
{         TRUE:  8.8 kHz low pass filter being used }
{*****}
Function mix3_GetADDACLowPass : Boolean;

Begin
    mix3_GetADDACLowPass := ( ( mix_Read( MIX3_ADCSTATE ) and
                               MIX3_LOWPASS88 ) <> 0 );
End;

{*****}
{ mix3_PrepereForStereo : Prepare mixer for stereo }
{                      recording/playback }
{-----*}
{ Input : iMode = FALSE : Output }
{         = TRUE  : Recording }
{-----*}
{ Info : - With stereo recording/playback, all filters must be }
{         disabled. However, for proper output }
{         the stereo switch must be enabled. }
{*****}
Procedure mix3_PrepereForStereo( iMode : Boolean );

Begin
    {-- Note current mixer status -----}
    bMix3DACStereo := Byte (mix3_GetDACStereo);
    bMix3DACFilter := Byte (mix3_GetDACFilter);
    bMix3ADCFilter := Byte (mix3_GetADCFilter);

    {-- Disable filter and stereo playback -----}
    mix3_SetDACStereo( iMode );
    mix3_SetDACFilter( FALSE );
    mix3_SetADCFilter( FALSE );
End;

{*****}
{ mix3_RestoreFromStereo : Restore mixer to old status }
{-----*}
{ Info : - The status of the mixer must be saved beforehand }
{         by the 'mix3_PrepereForStereo' function in the }
{         appropriate global variables. }
{*****}
Procedure mix3_RestoreFromStereo;

Begin
    {-- Restore filter and stereo playback -----}
    mix3_SetDACStereo( Boolean (bMix3DACStereo) );
    mix3_SetDACFilter( Boolean (bMix3DACFilter) );
    mix3_SetADCFilter( Boolean (bMix3ADCFilter) );
End;

{*****}
{ mix3_SetVolume : Set output volume }
{-----*}
{ Input : iSource - Source ( Microphone, DSP, CD, LINE, MIDI ) and }
{         TotalVolume ( Master ) }
{         iLeft   - Volume for left channel }
{         iRight  - Volume for right channel }
{-----*}
{ Info : - The total volume must be greater than 0, so that the }
{         other sources can be heard. The volume is specified }
{         separately for the left and right channels. Since a }
{         microphone is always monophone, its volume is determined }
{         by the maximum of the left and right channels. }
{         The program uses values between 0 and 255 to specify }
{         volumes, the values are mapped to the }
{         range that is valid for the sound card }
{*****}
Procedure mix3_SetVolume( iSource, iVolL, iVolR : Word );

var bIn, bVolR, bVolL : Byte;

Begin
    {-- Map values from 0 - 255 to values of 0 - 7 -----}
    bVolL := Byte ( iVolL shr 5 );
    bVolR := Byte ( iVolR shr 5 );

    case iSource of
        MIC,VOICE_L,VOICE_R,LINE,LINE_L,LINE_R,
        CD,CD_L,CD_R,MIDI,MIDI_L,MIDI_R,MASTER,MASTER_L,MASTER_R:
    
```

```

Begin { read current volume value }
bIn := Byte ( mix_Read( bMix3VolTable[iSource,SBPORT] ) );

{-- set new volume value ---}
case bMix3VolTable[iSource, CHANNEL] of
  CH_LEFT: { Change only left channel }
    bIn := Byte ( ( bIn and $0F ) or ( bVolL shl 5 ) );

  CH_RIGHT: { Change only right channel }
    bIn := Byte ( ( bIn and $F0 ) or ( bVolR shl 1 ) );

  CH_BOTH: { Change left and right }
    bIn := Byte ( ( bVolL shl 5 ) or ( bVolR shl 1 ) );

  CH_MAX: { Set maximum of left and right }
    Begin
      if (bVolL shl 1) > (bVolR shl 1) then
        bIn := Byte ( bVolL shl 1 )
      else
        bIn := Byte ( bVolR shl 1 );
      End;
    End;

    { ... and write back }
    mix_Write( bMix3VolTable[iSource , SBPORT], bIn );
  End;
End;

```

```

{*****}
{ mix3_GetVolume : Get output volume }
{*****}
{-----*}
{ Input : iSource - source ( Microphone, DSP, CD, LINE, MIDI ) and }
{           TotalVolume ( Master ) }
{ Output : Volume of source between 0 and 255, with stereo as }
{           LO and HIByte }
{*****}
Function mix3_GetVolume( iSource : Word ) : Word;

```

```

var bIn, bLeft, bRight : Byte;

```

```

Begin
  case iSource of
    MIC,VOICE,VOICE_L,VOICE_R,LINE,LINE_L,LINE_R,
    CD,CD_L,CD_R,MIDI,MIDI_L,MIDI_R,MASTER,MASTER_L,MASTER_R:
      Begin { read current volume value }
        bIn := Byte (mix_Read( bMix3VolTable[iSource,SBPORT] ) );
        bLeft := Byte ( ( ( bIn and $F0 ) shr 5 ) shl 5 );
        bRight := Byte ( ( ( bIn and $0F ) shr 1 ) shl 5 );

        case bMix3VolTable[iSource , CHANNEL] of
          CH_LEFT: mix3_GetVolume := bLeft;
          CH_MAX,
          CH_RIGHT: mix3_GetVolume := bRight;
          CH_BOTH: mix3_GetVolume := bLeft * 256 + bRight;
        End;
      End
    else mix3_GetVolume:= $FFFF;
  End;
End;

```

```

{*****}
{ mix3_SetADCSource : Set recording source }
{*****}
{-----*}
{ Input : iSource - Source for recording }
{           Allowed: MIC, CD, LINE }
{-----*}
{ Info : Der A/D-converter requires a source. The program specifies }
{           this source here. }
{*****}
Procedure mix3_SetADCSource( iSource : Word );

```

```

var bIn : Byte;

```

```

Begin
  bIn := Byte (mix_Read( MIX3_ADCSTATE ) );
  { always clear old source before setting new one }

  case iSource of
    MIC:
      Begin
        bIn := bin and not MIX3_SRCMSK;
        bIn := bin or MIX3_MICSRC;
      End;

    CD:
      Begin

```



```

bIn := bIn and not MIX3_SRCMSK;
bIn := bIn or MIX3_CDSRC;
End;

LINE:
Begin
    bIn := bIn and not MIX3_SRCMSK;
    bIn := bIn or MIX3_LINESRC;
End;
End;
mix_Write( MIX3_ADCSTATE, bIn );
End;

{*****}
{ mix3_GetADCSrc : Get current recording source }
{*****}
{-----*}
{ Input : iSource - Source for recording }
{           Allowed: MIC, CD, LINE }
{-----*}
{ Info : The A/D-converter requires a source. The program }
{ specifies this source here. }
{*****}
Function mix3_GetADCSrc : Word;

var bIn : Byte;

Begin
    bIn := Byte ( mix_Read( MIX3_ADCSTATE ) and MIX3_SRCMSK );

    case bIn of
        MIX3_MICSRC_,
        MIX3_MICSRC:  mix3_GetADCSrc := MIC;
        MIX3_CDSRC:   mix3_GetADCSrc := CD;
        MIX3_LINESRC: mix3_GetADCSrc := LINE;
        else          mix3_GetADCSrc := $FFFF;
    End;
End;

{*****}
{ mix4_PrepareForMonoADC : Prepare mixer for mono recording }
{-----*}
{ Info : - This command puts recording sources of the right }
{         A/D channel in the left channel as well. In mono }
{         recording, the DSP4.xx uses only the left channel. }
{-----*}
Procedure mix4_PrepareForMonoADC;

Begin
    bMix4SourceL := Byte ( mix_Read( MIX4_ADCSOURCE_L ) );
    bMix4SourceR := Byte ( mix_Read( MIX4_ADCSOURCE_R ) );
    mix_Write( MIX4_ADCSOURCE_L, bMix4SourceL or bMix4SourceR );
End;

{*****}
{ mix4_RestoreFromMonoADC : Restore mixer status. }
{-----*}
{ Info : This function can only be called after the }
{         'mix4_PrepareForMonoADC' function has already been called. }
{-----*}
Procedure mix4_RestoreFromMonoADC;

Begin
    mix_Write( MIX4_ADCSOURCE_L, bMix4SourceL );
    mix_Write( MIX4_ADCSOURCE_R, bMix4SourceR );
End;

{*****}
{ mix4_SetVolume : Set output volume of a mixer source }
{-----*}
{ Input : iSource - source ( Microphone, DSP, CD, LINE, MIDI, }
{           PCSPEAKER ) and TotalVolume ( Master ) }
{           iLeft   - Volume for left channel }
{           iRight  - Volume for right channel }
{-----*}
{ Info : - The total volume must be greater than 0, so that the }
{         other sources can be heard. The volume is specified }
{         separately for the left and right channels. Since a }
{         microphone is always monophone, its volume is determined }
{         by the left and right channels. }
{         The program uses values between 0 and 255 to specify the }
{         volume, the values are mapped to the range that is }
{         valid for the sound card. }
{         This function takes advantage of the fact that the }
{         registers for the left and right channels of a source }
{         are located at sequential port addresses. }
{-----*}

```

```

procedure mix4_SetVolume( iSource, iVolL, iVolR : Word);
Begin
  iVolL := iVolL and not $07;
  iVolR := iVolR and not $07;

  case iSource of
    MIC, PCSPEAKER, VOICE,VOICE_L,VOICE_R,LINE,LINE_L,LINE_R,
    CD,CD_L,CD_R,MIDI,MIDI_L,MIDI_R,MASTER,MASTER_L,MASTER_R:

      case bMix4VolTable[iSource, CHANNEL] of
        CH_LEFT:
          mix_Write( bMix4VolTable[iSource, SBPORT], iVolL );

        CH_RIGHT:
          mix_Write( bMix4VolTable[iSource, SBPORT] + 1, iVolR );

        CH_BOTH:
          Begin
            mix_Write( bMix4VolTable[iSource, SBPORT], iVolL );
            mix_Write( bMix4VolTable[iSource, SBPORT] + 1, iVolR );
          End;

        CH_MAX:
          Begin
            if iVolL > iVolR then
              mix_Write( bMix4VolTable[iSource,SBPORT], iVolL)
            else
              mix_Write( bMix4VolTable[iSource,SBPORT], iVolR)
            End;
          End;
        End;
      End;
    End;
  End;

  {*****}
  { mix4_GetVolume : Get output volume }
  {*****}
  {-----*}
  { Input : iSource : source ( Microphone, DSP, CD, LINE, MIDI, }
  {          PCSPEAKER ) and TotalVolume ( Master ) }
  { Output : Volume of source between 0 and 255, with stereo as }
  {          LO and HIByte }
  {*****}
Function mix4_GetVolume( iSource : Word ) : Word;

var bLeft, bRight : Byte;

Begin
  case iSource of
    MIC,PCSPEAKER,VOICE,VOICE_L,VOICE_R,LINE,LINE_L,LINE_R,
    CD,CD_L,CD_R,MIDI,MIDI_L,MIDI_R,MASTER,MASTER_L,MASTER_R:

      {-- read current volume value -----}
      Begin
        bLeft := Byte ( mix_Read( bMix4VolTable[iSource, SBPORT] ) );
        bLeft := bLeft and not $07;
        bRight := Byte ( mix_Read( bMix4VolTable[iSource , SBPORT] + 1 ) );
        bRight := bRight and not $07;

        case bMix4VolTable[iSource , CHANNEL] of
          CH_LEFT:
            mix4_GetVolume := bLeft;

          CH_MAX:
            if( bLeft > bRight ) then
              mix4_GetVolume := bLeft
            else mix4_GetVolume := bRight;

          CH_RIGHT:
            mix4_GetVolume := bRight;

          CH_BOTH:
            mix4_GetVolume := bLeft * 256 + bRight;
        End;
      end;
    else mix4_GetVolume := $FFFF;
  End;
End;

{*****}
{ mix4_SetADCSOURCEL : Set recording source for left ADC }
{*****}
{-----*}
{ Input : iSource - Recording source }
{          iState - Switch source on or off }
{          == 0: off }
{          <> 0: on }
{*****}

```

```

procedure mix4_SetADCSOURCEL(iSource : Word; iState : Boolean );
var bInL : Byte;

Begin
  bInL := Byte ( mix_Read( MIX4_ADCSOURCE_L ) );

  case iSource of
    MIDI,MIDI_L,MIDI_R,CD,CD_L,CD_R,
    LINE,LINE_L,LINE_R,MIC:
      if iState then                                { set input channel }
        bInL := bInL or bMix4SourceBits[iSource]
      else                                           { clear input channel }
        bInL := bInL and not bMix4SourceBits[iSource];
      End;
    mix_Write( MIX4_ADCSOURCE_L, bInL );
  End;

  {*****}
  { mix4_SetADCSOURCE_R : Set recording source for right ADC }
  {*****}
  { Input : iSource - Recording source }
  { iState - Switch source on or off }
  { = FALSE: off }
  { = TRUE: on }
  {*****}
procedure mix4_SetADCSOURCE_R( iSource : Word; iState : Boolean );

var bInR : Byte;

Begin
  bInR := Byte ( mix_Read( MIX4_ADCSOURCE_R ) );
  case iSource of
    MIDI,MIDI_L,MIDI_R,CD,CD_L,CD_R,
    LINE,LINE_L,LINE_R,MIC:
      if iState then                                { set input channel }
        bInR := bInR or bMix4SourceBits[iSource]
      else                                           { clear input channel }
        bInR := bInR and not bMix4SourceBits[iSource];
      End;
    mix_Write( MIX4_ADCSOURCE_R, bInR );
  End;

  {*****}
  { mix4_GetADCSOURCE_L : Get recording source for left ADC }
  {*****}
  { Input : iSource - Recording source }
  { Output : ID for which ADC channels the specified recording }
  { source is set. }
  {*****}
function mix4_GetADCSOURCE_L(iSource : Word) : Word;

var bInL : Byte;

Begin
  bInL := Byte ( mix_Read( MIX4_ADCSOURCE_L ) );
  case iSource of
    MIDI,CD,LINE,MIC:
      if ( bInL and bMix4SourceBits[iSource] ) = bMix4SourceBits[iSource]
        then mix4_GetADCSOURCE_L := CH_BOTH
        else Mix4_GetADCSOURCE_L := CH_None;

    MIDI_L,CD_L,LINE_L:
      if ( bInL and bMix4SourceBits[iSource] <> 0 )
        then mix4_GetADCSOURCE_L := CH_LEFT
        else Mix4_GetADCSOURCE_L := CH_None;

    MIDI_R,CD_R,LINE_R:
      if( bInL and bMix4SourceBits[iSource] <> 0 )
        then mix4_GetADCSOURCE_L := CH_RIGHT
        else Mix4_GetADCSOURCE_L := CH_None;

    else { case else }
      Mix4_GetADCSOURCE_L := CH_None;
  End;
End;

  {*****}
  { mix4_GetADCSOURCE_R : Get recording source for right ADC }
  {*****}
  { Input : iSource : Recording source }
  { Output : ID for which ADC channels the specified recording }
  { source is set. }
  { Possible output: CH_BOTH, CH_LEFT, CH_RIGHT, CH_NONE }
  {*****}
function mix4_GetADCSOURCE_R( iSource : Word ) : Word;

```

```

var bInR : Byte;

Begin
  bInR := Byte ( mix_Read( MIX4_ADCSOURCE_R ) );

  case iSource of
    MIDI,CD,LINE,MIC:
      if ( bInR and bMix4SourceBits[iSource] ) = bMix4SourceBits[iSource]
      then mix4_GetADCSourcer := CH_BOTH
      else mix4_GetADCSourcer := CH_NONE;

    MIDI_L,CD_L,LINE_L:
      if ( bInR and bMix4SourceBits[iSource] ) <> 0
      then mix4_GetADCSourcer := CH_LEFT
      else mix4_GetADCSourcer := CH_NONE;

    MIDI_R,CD_R,LINE_R:
      if( bInR and bMix4SourceBits[iSource] ) <> 0
      then mix4_GetADCSourcer := CH_RIGHT
      else mix4_GetADCSourcer := CH_NONE;

  else { case else }
    mix4_GetADCSourcer := CH_NONE;
  End;
End;

```

```

{*****}
{ mix4_SetOUTSource : Set output(playback) source }
{-----*}
{ Input : iSource - source }
{         iState - Switch source on or off }
{*****}
Procedure mix4_SetOUTSource( iSource : Word; iState : Boolean );

```

```

var bIn : Byte;

Begin
  bIn := Byte ( mix_Read( MIX4_OUTSOURCE ) );
  case iSource of
    CD,CD_L,CD_R,LINE,LINE_L,LINE_R,MIC:
      if iState then
        bIn := bIn or bMix4SourceBits[iSource] { Set }
      else
        bIn := bIn and not bMix4SourceBits[iSource]; { Clear }
      End;
  mix_Write( MIX4_OUTSOURCE, bIn );
End;

```

```

{*****}
{ mix4_GetOUTSource : Get output source }
{-----*}
{ Input : iSource - Recording source }
{ Output : ID for which DAC channels the specified recording }
{           source is set. }
{           Possible output: CH_BOTH, CH_LEFT, CH_RIGHT, CH_NONE }
{*****}
Function mix4_GetOUTSource( iSource : Word ) : Word;

```

```

var bIn : Byte;

Begin
  bIn := Byte ( mix_Read( MIX4_OUTSOURCE ) );

  case iSource of
    CD,LINE,MIC:
      if ( bIn and bMix4SourceBits[iSource] ) <> 0
      then mix4_GetOUTSource := CH_BOTH
      else mix4_GetOUTSource := CH_NONE;

    CD_L,LINE_L:
      if ( bIn and bMix4SourceBits[iSource] ) <> 0
      then mix4_GetOUTSource := CH_LEFT
      else mix4_GetOUTSource := CH_NONE;

    CD_R,LINE_R:
      if ( bIn and bMix4SourceBits[iSource] ) <> 0
      then mix4_GetOUTSource := CH_RIGHT
      else mix4_GetOUTSource := CH_NONE;

  else { case else }
    mix4_GetOUTSource := CH_NONE;
  End;
End;

```

```

{*****}

```

```

{mix4_SetADCGain : Set recording preamplifier
}-----*
{ Input : iGainL, iGainR - 0 : 0 dB
{                               1 : 6 dB
{                               2 : 12 dB
{                               3 : 18 dB
}-----*
*****
Procedure mix4_SetADCGain( iGainL, iGainR : Word);

var bIn, bGainL, bGainR : Byte;

Begin
  bGainL := Byte ( iGainL and $03 );
  bGainL := bGainL shl 6;
  bGainR := Byte ( iGainR and $03 );
  bGainR := bGainR shl 6;

  bIn := Byte ( ( mix_Read( MIX4_ADCGAIN_L ) and $3F ) or bGainL);
  mix_Write( MIX4_ADCGAIN_L, bIn );

  bIn := Byte ( ( mix_Read( MIX4_ADCGAIN_R ) and $3F ) or bGainR );
  mix_Write( MIX4_ADCGAIN_R, bIn );
End;

*****
{mix4_GetADCGain : Get values of recording preamplifier
}-----*
{ Input : iChannel - CH_LEFT : Gain for left channel
{                  CH_RIGHT : Gain for right channel
{                  other : Gain for both channels
{ Output : Current preamplifier values
{          In determining the value of both channels, the value
{          for the left channel is supplied in the LO-Byte, while
{          the value for the right channel is supplied in the
{          HI-Byte. Otherwise, the requested value is always
{          returned in the LO-Byte!
}-----*
*****
Function mix4_GetADCGain( iChannel : Word ) : Word;

var bGainL, bGainR : Byte;

Begin
  bGainL := Byte ( mix_Read( MIX4_ADCGAIN_L ) );
  bGainL := bGainL and $C0;
  bGainL := bGainL shr 6;

  bGainR := Byte ( mix_Read( MIX4_ADCGAIN_R ) );
  bGainR := bGainR and $C0;
  bGainR := bGainR shr 6;

  case iChannel of
    CH_LEFT:
      mix4_GetADCGain := bGainL;
    CH_RIGHT:
      mix4_GetADCGain := bGainR;
    else { case else }
      mix4_GetADCGain := bGainL * 256 + bGainR;
  End;
End;

*****
{mix4_SetOUTGain : Set output preamplifier
}-----*
{ Input : iGainL, iGainR - 0 : 0 dB
{                               1 : 6 dB
{                               2 : 12 dB
{                               3 : 18 dB
}-----*
*****
Procedure mix4_SetOUTGain( iGainL, iGainR : Word);

var bIn, bGainL, bGainR : Byte;

Begin
  bGainL := Byte ( ( iGainL and $03 ) shl 6 );
  bGainR := Byte ( ( iGainR and $03 ) shl 6 );

  bIn := Byte ( ( mix_Read( MIX4_OUTGAIN_L ) and $3F ) or bGainL );
  mix_Write( MIX4_OUTGAIN_L, bIn );

  bIn := Byte ( ( mix_Read( MIX4_OUTGAIN_R ) and $3F ) or bGainR );
  mix_Write( MIX4_OUTGAIN_R, bIn );
End;

*****

```

```

mix4_GetOUTGain : Get output preamplifier
}-----*
{ Input : iChannel - CH_LEFT : Gain for left channel
                  CH_RIGHT : Gain for right channel
                  other : Gain for both channels
}
{ Output : Current preamplifier values
          In determining the value of both channels,
          the value for the left channel is supplied in the
          LO-Byte, while the value for the right channel is
          supplied in the HI-Byte. Otherwise the requested
          value is always specified in the LO-Byte!
}
*****
Function mix4_GetOUTGain( iChannel:Word ) : Word;

var bGainL, bGainR : Byte;
Begin
  bGainL := Byte ( mix_Read( MIX4_OUTGAIN_L ) );
  bGainL := bGainL and $C0;
  bGainL := bGainL shr 6;
  bGainR := Byte ( mix_Read( MIX4_OUTGAIN_R ) );
  bGainR := bGainR and $C0;
  bGainR := bGainR shr 6;

  case iChannel of
    CH_LEFT:
      mix4_GetOUTGain := bGainL;

    CH_RIGHT:
      mix4_GetOUTGain := bGainR;

    CH_BOTH:
      mix4_GetOUTGain := bGainL * 256 + bGainR ;

    else { case else }
      mix4_GetOUTGain := 0;
  End;
End;

*****
{ mix4_SetAGC : Enable / disable Automatic Gain Control
}-----*
{ Input : iState - FALSE: Disable AGC ( 0 dB )
                  TRUE: Enable AGC ( 20dB )
}
}-----*
{ Info : - AGC is used for recordings with a microphone and
          permits the addition of a preamplifier, which
          amplifies the microphone signal by 20dB.
}
*****
Procedure mix4_SetAGC( iState : Word );

var bIn : Byte;

Begin
  bIn := Byte ( mix_Read( MIX4_AGC ) );
  if iState = 0 then
    mix_Write( MIX4_AGC, bIn and not MIX4_AGCON )
  else
    mix_Write( MIX4_AGC, bIn or MIX4_AGCON );
End;

*****
{ mix4_GetAGC : Get Automatic Gain Control
}-----*
{ Output: TRUE : AGC enabled
          FALSE : AGC disabled
}
*****
Function mix4_GetAGC : Boolean;

Begin
  mix4_GetAGC := ( ( mix_Read( MIX4_AGC ) and MIX4_AGCON ) <> 0 );
End;

*****
{ mix4_SetTreble : Set treble
}-----*
{ Input : iTrebleL, iTrebleR : Set treble for left and right
                              channels. Values from 0 to 15
                              equal -14 dB to 14 dB in
                              increments of 2 dB.
}
}-----*
{ Info : The Sound Blaster16 card has a preamplifier
          whose sound attributes can be modified.
}
*****
Procedure mix4_SetTreble( iTrebleL, iTrebleR : Word);

Begin

```

```

mix_Write( MIX4_TREBLE_L, Byte( iTrebleL shl 4 ) );
mix_Write( MIX4_TREBLE_R, Byte( iTrebleR shl 4 ) );
End;

{*****}
{ mix4_GetTreble : Get current treble setting }
{-----*}
{ Input : iChannel : CH_LEFT : for left channel }
{          CH_RIGHT : for right channel }
{          CH_BOTH : for both channels }
{ Output : Current preamplifier values }
{          In determining the value of both channels, }
{          the value for the left channel is furnished }
{          in the LO-Byte, while the value for the right }
{          channel is furnished in the HI-Byte. Otherwise }
{          the requested value is always returned in the LO-Byte. }
{*****}
Function mix4_GetTreble( iChannel : Word ) : Word;

var bTrebleL, bTrebleR : Byte;

Begin
  bTrebleL := mix_Read( MIX4_TREBLE_L ) shr 4;
  bTrebleR := mix_Read( MIX4_TREBLE_R ) shr 4;

  case iChannel of
    CH_LEFT: mix4_GetTreble := bTrebleL;
    CH_RIGHT: mix4_GetTreble := bTrebleR;
    CH_BOTH: mix4_GetTreble := bTrebleL * 256 + bTrebleR ;
    else     mix4_GetTreble := 0
  end;
End;

{*****}
{ mix4_SetBass : Set Bass }
{-----*}
{ Input : iBassL, iBassR : Set bass for left and right output }
{          channels. Values from 0 to 15 }
{          are the equivalent of -14 dB to 14 dB }
{          in increments of 2 dB. }
{-----*}
{ Info - The Sound Blaster 16 card has a preamplifier }
{          whose sound attributes can be modified. }
{*****}
Procedure mix4_SetBass( iBassL, iBassR : Word );

Begin
  mix_Write( MIX4_BASS_L, iBassL shl 4 );
  mix_Write( MIX4_BASS_R, iBassR shl 4 );
End;

{*****}
{ mix4_GetBass : Get current bass setting }
{-----*}
{ Input : iChannel : CH_LEFT : for left channel }
{          CH_RIGHT : for right channel }
{          CH_BOTH : for both channels }
{ Output : Current preamplifier values }
{          In determining the value of both channels, }
{          the value for the left channel is supplied }
{          in the LO-Byte, while the value for the right }
{          channel is provided in the HI-Byte. Otherwise, }
{          the requested value is always specified in the LO-Byte! }
{*****}
Function mix4_GetBass( iChannel : Word ) : Word;

var bBassL, bBassR : Byte;

Begin
  bBassL := Byte( mix_Read( MIX4_BASS_L ) shr 4 );
  bBassR := Byte( mix_Read( MIX4_BASS_R ) shr 4 );

  case iChannel of
    CH_LEFT: mix4_GetBass := bBassL;
    CH_RIGHT: mix4_GetBass := bBassR;
    CH_BOTH: mix4_GetBass := bBassL * 256 + bBassR ;
    else     mix4_GetBass := 0;
  end;
End;

End.

```

