

# MIXUTIL.C

```

/*****
/*          M I X U T I L . C          */
/**-----**/
/* 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_??? - Valid for all mixer versions          */
/*          mix3_??? - ... for mixer CT1345 ( DSP V3.00 and higher ) */
/*          mix4_??? - ... for mixer CT1745 ( DSP V4.00 and higher ) */
/**-----**/
/* COMPILER      : Borland C++ 3.1, Microsoft Visual C++ 1.5          */
/*****
#ifdef __MIXUTIL_C          /* Can also be #Included          */
#define __MIXUTIL_H

/*-- Add include files -----*/

#include <dos.h>
#include <stdlib.h>
#include <conio.h>

#include "sbutil.h"
#include "mixutil.h"

/*-- Global Module Variables -----*/

SBBASE _FP MIXBASE;          /* Port Address of mixer */

```

```

BYTE bMix3DACStereo,      /* These variables are used to save */
    bMix3DACFilter,      /* some mixer values.                */
    bMix3ADCFilter;

BYTE bMix4SourceL,
    bMix4SourceR;

/* The following tables are used for setting the volume. */
/* Program notes the register and volume (left, right, both) */
/* to be set for each potential source. */
BYTE bMix3VolTable[ MAX_SRC ][ 2 ] = { { 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 } };

BYTE bMix4VolTable[ MAX_SRC ][ 2 ] = { { 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_MAX },
{ MIX4_PCSPEAKERVOL, CH_MAX },

{ 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 you to enable and disable several */
/* recording and playback sources. The following array notes */
/* the appropriate switch bits for each potential source. */
BYTE bMix4SourceBits[ MAX_SRC ] = {
/* CD */ MIX4_CD_L | MIX4_CD_R,
/* LINE */ MIX4_LINE_L | MIX4_LINE_R,
/* VOICE ( non-switchable ) */ 0,
/* MASTER ( non-switchable ) */ 0,
/* MIDI */ MIX4_MIDI_L | MIX4_MIDI_R,
/* MIC */ MIX4_MIC,
/* PCSPEAKER ( non-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 };

/*****
/* mix_SetBase : Set Port Address */
**-----**
/* Input : pSBBASE - Address of an initialized SB base structure */
/*          iReset   - <> 0 : Mixer is to be reset */
**-----**
/* Info      : - Before one of the following functions can be called */
/*              the program must use this function to specify */
/*              the mixer address! */
*****/
WORD mix_SetBase( PSBBASE pSBBASE, WORD iReset )
{
    if( pSBBASE )
        if( pSBBASE->iMixPort != -1 )
        {
            MIXBASE = *pSBBASE;
            if( iReset ) mix_Reset();
            return NO_ERROR;
        }
    return ERROR;
}

```

```

/*****
/* mix_Write : Send byte to mixer register */
/**-----*/
/* Input : iReg - Number of register to be modified */
/*          iData - Value to be written to the register */
/*****
VOID mix_Write( WORD iReg, WORD iData )
{
    /* Specify register to be written */
    outp( MIXBASE.iMixPort + MIX_REGISTERPORT, ( BYTE ) iReg );
    /* Send new register value */
    outp( MIXBASE.iMixPort + MIX_DATAPORT, ( BYTE ) iData );
}

/*****
/* mix_Read : Read byte from mixer register */
/**-----*/
/* Input : iReg - Number of register to be read */
/* Output : Value of mixer register */
/*****
WORD mix_Read( WORD iReg )
{
    /* Specify register to be read */
    outp( MIXBASE.iMixPort + MIX_REGISTERPORT, ( BYTE ) iReg );
    /* Send register value to caller */
    return inp( MIXBASE.iMixPort + MIX_DATAPORT );
}

/*****
/* mix_Reset : Reset mixer */
/*****
VOID mix_Reset( VOID )
{

```

```

    mix_Write( MIX_RESET, 0 );
}

/*****
/* mix3_SetADCFilter : Switch ADC filter on or off */
/**-----*/
/* Input : iState - == 0: Switch filter off */
/*          <> 0: Switch filter on */
/**-----*/
/* Info : - The bit that controls the filter is an */
/*          OFF switch. Therefore, when the bit is set, */
/*          the filter is switched off! */
/*          ADC = Analog/Digital Converter (recording) */
*****/
VOID mix3_SetADCFilter( WORD iState )
{
    BYTE bIn;

    bIn = ( BYTE )mix_Read( MIX3_ADCSTATE ); /* Read old reg. value */
    /* Set / clear filter bit (LOW active) */
    mix_Write( MIX3_ADCSTATE,
               iState ? ( bIn & ~MIX3_ADCFILTEROFF ) :
                   ( bIn | MIX3_ADCFILTEROFF ) );
}

/*****
/* mix3_GetADCFilter : Get status of ADC filter */
/**-----*/
/* Output : == 0 - Filter disabled */
/*          <> 0 - Filter enabled */
*****/
WORD mix3_GetADCFilter( VOID )
{
    /* Warning! Filter bit is LOW active */

```

```

    return ( ( mix_Read( MIX3_ADCSTATE ) & MIX3_ADCFILTEROFF ) == 0 );
}

/*****
/* mix3_SetDACFilter : Switch DAC filter on or off */
/**-----**/
/* Input : iState - == 0 : Switch off filter */
/*          <> 0 : Switch on filter */
/**-----**/
/* Info :      DAC : Digital/Analog Converter (playback) */
/*****
VOID mix3_SetDACFilter( WORD iState )
{
    BYTE bIn;

    bIn = ( BYTE )mix_Read( MIX3_DACSTATE ); /* Read old reg. value */
        /* Set/clear filter bit (Filter bit is LOW active) */
    mix_Write( MIX3_DACSTATE,
        iState ? ( bIn & ~MIX3_DACFILTEROFF ) :
            ( bIn | MIX3_DACFILTEROFF );
}

/*****
/* mix3_GetDACFilter : Get status of DAC filter */
/**-----**/
/* Output : == 0 - Filter off */
/*           <> 0 - Filter on */
/*****
WORD mix3_GetDACFilter( VOID )
{
    /* Warning! Filter bit is LOW active */
    return ( ( mix_Read( MIX3_DACSTATE ) & MIX3_DACFILTEROFF ) == 0 );
}

```

```

/*****
/* mix3_SetDACStereo : Switch stereo playback (DAC) on or off */
/**-----**/
/* Input : iState - == 0: Disable stereo playback => Mono */
/*          <> 0: Enable stereo playback */
/*****
VOID mix3_SetDACStereo( WORD iState )
{
    BYTE bIn;

    bIn = ( BYTE )mix_Read( MIX3_DACSTATE ); /* Read old Reg. value */
    mix_Write( MIX3_DACSTATE,
               iState ? ( bIn | MIX3_STEREOON ) :
                       ( bIn & ~MIX3_STEREOON ) );
}

/*****
/* mix3_GetDACStereo : Get status of stereo playback */
/**-----**/
/* Output : == 0 - Stereo playback off */
/*          <> 0 - Stereo playback on */
/*****
WORD mix3_GetDACStereo( VOID )
{
    return ( ( mix_Read( MIX3_DACSTATE ) & MIX3_STEREOON ) != 0 );
}

/*****
/* mix3_SetADDACLowPass : Set low pass filter for recording */
/**-----**/
/* Input : iState : == 0: 3.2 kHz LowPass Filter */
/*          <> 0: 8.8 kHz LowPass 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. */
/*-----*/
VOID mix3_SetADDACLowPass( WORD iState )
{
    BYTE bIn;

    bIn = ( BYTE )mix_Read( MIX3_ADCSTATE ); /* Read old Reg. value */
    mix_Write( MIX3_ADCSTATE,
               iState ? ( bIn | MIX3_LOWPASS88 ) :
                   ( bIn & ~MIX3_LOWPASS88 ) );
}

/*-----*/
/* mix3_GetADDACLowPass : Get low pass filter being used */
/*-----*/
/* Output : == 0: 3.2 kHz low pass filter in use */
/*          <> 0: 8.8 kHz low pass filter in use */
/*-----*/
WORD mix3_GetADDACLowPass( VOID )
{
    return ( ( mix_Read( MIX3_ADCSTATE ) & MIX3_LOWPASS88 ) != 0 );
}

/*-----*/
/* mix3_PrepereForStereo : Prepare mixer for stereo */
/*          recording/playback */
/*-----*/
/* Input : iMode : == DAC : Output */
/*          <> DAC : Recording */
/*-----*/

```

```

/* Info : - With stereo recording/playback all filters must be      */
/*          off. However, for regular output, the stereo switch      */
/*          must be enabled.                                         */
/*****
VOID mix3_PrepStereo( WORD iMode )
{
    /* Note old mixer states */
    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 );
}

/*****
/* mix3_RestoreFromStereo : Restore mixer to old state              */
/*-----*/
/* Info : - The state of the mixer must have been saved beforehand */
/*          in the appropriate global variables                      */
/*          by the 'mix3_PrepStereo' function.                      */
/*****
VOID mix3_RestoreFromStereo( VOID )
{
    /* Restore filter and stereo playback */
    mix3_SetDACStereo( bMix3DACStereo );
    mix3_SetDACFilter( bMix3DACFilter );
    mix3_SetADCFilter( bMix3ADCFilter );
}

/*****
/* 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 TotalVolume 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 */
/*           the volumes, converting them into the appropriate values */
/*           for the sound card. */
/*****
VOID mix3_SetVolume( WORD iSource, WORD iVolL, WORD iVolR )
{ BYTE bIn, bVolR, bVolL;
    /* Map values from 0 - 255 to values from 0 - 7 */
    bVolL = ( BYTE )( iVolL >> 5 );
    bVolR = ( BYTE )( iVolR >> 5 );
    switch( iSource )
    {
        case MIC:                                /* PC speaker is not used! */
        case VOICE:    case VOICE_L:    case VOICE_R:
        case LINE:     case LINE_L:     case LINE_R:
        case CD:       case CD_L:       case CD_R:
        case MIDI:     case MIDI_L:     case MIDI_R:
        case MASTER:   case MASTER_L:   case MASTER_R:
            /* Read old volume value... */
            bIn = ( BYTE )mix_Read( bMix3VolTable[ iSource ][ PORT ] );
            /* ...and change... */
            switch( bMix3VolTable[ iSource ][ CHANNEL ] )

```

```

{
    case CH_LEFT:
        /* Change only left channel */
        bIn = ( BYTE )( ( bIn & 0x0F ) | ( bVolL << 5 ) );
        break;
    case CH_RIGHT:
        /* Change only right channel */
        bIn = ( BYTE )( ( bIn & 0xF0 ) | ( bVolR << 1 ) );
        break;
    case CH_BOTH:
        /* Change left and right channels */
        bIn = ( BYTE )( ( bVolL << 5 ) | ( bVolR << 1 ) );
        break;
    case CH_MAX:
        /* Set maximum for left and right channels */
        bIn = ( BYTE ) (max( bVolL << 1, bVolR << 1 ) );
        break;
}

/* ... and write back */
mix_Write( bMixVolTable[ iSource ][ PORT ], bIn );
}

/*****
/* 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-Byte and HI-Byte */
*****/
WORD mix3_GetVolume( WORD iSource )
{
    BYTE bIn, bLeft, bRight;

    switch( iSource )
    {

```

```

case MIC:                                /* PC speaker is not used! */
case VOICE:    case VOICE_L:    case VOICE_R:
case LINE:     case LINE_L:     case LINE_R:
case CD:       case CD_L:       case CD_R:
case MIDI:     case MIDI_L:     case MIDI_R:
case MASTER:   case MASTER_L:   case MASTER_R:
                                /* Read old volume value */
    bIn = ( BYTE )mix_Read( bMix3VolTable[ iSource ][ PORT ] );
    bLeft  = ( BYTE )( ( ( bIn & 0xF0 ) >> 5 ) << 5 );
    bRight = ( BYTE )( ( ( bIn & 0x0F ) >> 1 ) << 5 );

    switch( bMix3VolTable[ iSource ][ CHANNEL ] )
    {
        case CH_LEFT:    return bLeft;
        case CH_MAX:
        case CH_RIGHT:   return bRight;
        case CH_BOTH:    return MAKEWORD( bLeft, bRight );
    }
}
return 0xFFFF;
}

/*****
/* mix3_SetADCSource : Set recording source                                */
/**-----**/
/* Input : iSource - Source for recording                                */
/*                               Allowed: MIC, CD, LINE                    */
/**-----**/
/* Info : The A/D converter requires a source. This source is            */
/*         specified here.                                                */
*****/
VOID mix3_SetADCSource( WORD iSource )

```

```

{  BYTE bIn;

    bIn = ( BYTE )mix_Read( MIX3_ADCSTATE );
    /* Always delete old source before setting new source */
    switch( iSource )
    {
        case MIC:  bIn &= ~MIX3_SRCMSK; bIn |= MIX3_MICSRC;  break;
        case CD:   bIn &= ~MIX3_SRCMSK; bIn |= MIX3_CDSRC;   break;
        case LINE: bIn &= ~MIX3_SRCMSK; bIn |= MIX3_LINESRC; break;
    }
    mix_Write( MIX3_ADCSTATE, bIn );
}

/*****
/* mix3_GetADCSource : Get current recording source          */
/**-----**/
/* Input : iSource - Source for recording                    */
/*                               Allowed: MIC, CD, LINE       */
/**-----**/
/* Info : The A/D converter requires a source. This source is */
/*          specified here.                                     */
/*****
WORD mix3_GetADCSource( VOID )
{  BYTE bIn;

    bIn = ( BYTE )( mix_Read( MIX3_ADCSTATE ) & MIX3_SRCMSK );

    switch( bIn )
    {
        case MIX3_MICSRC_:
        case MIX3_MICSRC:  return MIC;
        case MIX3_CDSRC:   return CD;

```

```

        case MIX3_LINESRC: return LINE;
        default:           return 0xFFFF;
    }
}

/*****
/* mix4_PrepereForMonoADC : Prepare mixer for mono recording          */
/**-----*/
/* Info : - This command sets 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.                    */
/*****
VOID mix4_PrepereForMonoADC( VOID )
{
    bMix4SourceL = ( BYTE )mix_Read( MIX4_ADCSOURCE_L );
    bMix4SourceR = ( BYTE )mix_Read( MIX4_ADCSOURCE_R );
    mix_Write( MIX4_ADCSOURCE_L, bMix4SourceL | bMix4SourceR );
}

/*****
/* mix4_RestoreFromMonoADC : Restore mixer status.                  */
/**-----*/
/* Info : This function can only be called if the                    */
/*          'mix4_PrepereForMonoADC' function was called beforehand.   */
/*****
VOID mix4_RestoreFromMonoADC( VOID )
{
    mix_Write( MIX4_ADCSOURCE_L, bMix4SourceL );
    mix_Write( MIX4_ADCSOURCE_R, bMix4SourceR );
}

/*****

```

```

/* 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 TotalVolume 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, converting them into the appropriate values for */
/*           the sound card. */
/*           This function makes use of the fact that the registers */
/*           for the left and right channels of a source are located */
/*           at sequential port addresses. */
/*****/
VOID mix4_SetVolume( WORD iSource, WORD iVolL, WORD iVolR )
{
    iVolL &= ~0x07;
    iVolR &= ~0x07;

    switch( iSource )
    {
        case MIC:
        case PCSPEAKER:
        case VOICE:      case VOICE_L:      case VOICE_R:
        case LINE:       case LINE_L:       case LINE_R:
        case CD:         case CD_L:         case CD_R:
    }

```



```

case MIDI:      case MIDI_L:      case MIDI_R:
case MASTER:   case MASTER_L:    case MASTER_R:
switch( bMix4VolTable[ iSource ][ CHANNEL ] )
{
    case CH_LEFT:
        mix_Write( bMix4VolTable[ iSource ][ PORT ], iVolL );
        break;
    case CH_RIGHT:
        mix_Write( bMix4VolTable[ iSource ][ PORT ] + 1, iVolR );
        break;
    case CH_BOTH:
        mix_Write( bMix4VolTable[ iSource ][ PORT ], iVolL );
        mix_Write( bMix4VolTable[ iSource ][ PORT ] + 1, iVolR );
        break;

    case CH_MAX:
        mix_Write( bMix4VolTable[ iSource ][ PORT ],
                    max( iVolL, iVolR ) );
        break;
}
}
}

/*****
/* mix4_GetVolume : Get output volume */
/**-----**/
/* Input : iSource : Source ( Microphone, DSP, CD, LINE, MIDI, */
/*          PCSPEAKER ) and TotalVolume ( Master ) */
/* Output : Volume of the source between 0 and 255, with stereo as */
/*          LO-Byte and HI-Byte */
/*****
WORD mix4_GetVolume( WORD iSource )

```

```

{ BYTE bLeft, bRight;

switch( iSource )
{
    case MIC:
    case PCSPEAKER:
    case VOICE:    case VOICE_L:    case VOICE_R:
    case LINE:    case LINE_L:    case LINE_R:
    case CD:      case CD_L:      case CD_R:
    case MIDI:    case MIDI_L:    case MIDI_R:
    case MASTER:  case MASTER_L:  case MASTER_R:
                                /* Read old volume value */
    bLeft = ( BYTE )mix_Read( bMix4VolTable[ iSource ][ PORT ] );
    bLeft &= ~0x07;
    bRight = ( BYTE )mix_Read( bMix4VolTable[ iSource ][ PORT ] + 1 );
    bRight &= ~0x07;

    switch( bMix4VolTable[ iSource ][ CHANNEL ] )
    {
        case CH_LEFT:  return bLeft;
        case CH_MAX:
        case CH_RIGHT: return bRight;
        case CH_BOTH:  return MAKEWORD( bLeft, bRight );
    }
}
return 0xFFFF;
}

/*****
/* mix4_SetADCSOURCEL : Set recording source for left ADC          */
/*-----**/
/* Input : iSource - Recording source                                */

```

```

/*          iState  - Switch source on or off                      */
/*          == 0: off                                           */
/*          <> 0: on                                             */
/*****
VOID mix4_SetADCSOURCEL( WORD iSource, WORD iState )
{ BYTE bInL;

    bInL = ( BYTE )mix_Read( MIX4_ADCSOURCE_L );
    switch( iSource )
    {
        case MIDI: case MIDI_L: case MIDI_R:
        case CD:   case CD_L:   case CD_R:
        case LINE: case LINE_L: case LINE_R:
        case MIC:
            if( iState )                /* Set input channel */
                bInL |= bMix4SourceBits[ iSource ];
            else                        /* Clear input channel */
                bInL &= ~bMix4SourceBits[ iSource ];
    }
    mix_Write( MIX4_ADCSOURCE_L, bInL );
}

/*****
/* mix4_SetADCSOURCER : Set recording source for right ADC      */
/*-----*/
/* Input : iSource - Recording source                            */
/*          iState  - Switch source on or off                      */
/*          == 0: off                                           */
/*          <> 0: on                                             */
/*****
VOID mix4_SetADCSOURCER( WORD iSource, WORD iState )
{ BYTE bInR;

```

```

bInR = ( BYTE )mix_Read( MIX4_ADCSOURCE_R );
switch( iSource )
{
    case MIDI: case MIDI_L: case MIDI_R:
    case CD:    case CD_L:   case CD_R:
    case LINE: case LINE_L: case LINE_R:
    case MIC:
        if( iState )                /* Set input channel */
            bInR |= bMix4SourceBits[ iSource ];
        else                        /* Clear input channel */
            bInR &= ~bMix4SourceBits[ iSource ];
}
mix_Write( MIX4_ADCSOURCE_R, bInR );
}

/*****
/* mix4_GetADCSOURCE_L : Get recording source for left ADC          */
/*-----*/
/* Input : iSource - Recording source                               */
/* Output : ID for which ADC channel the specified recording        */
/*           source is set.                                          */
*****/
WORD mix4_GetADCSOURCE_L( WORD iSource )
{
    BYTE bInL;

    bInL = ( BYTE )mix_Read( MIX4_ADCSOURCE_L );

    switch( iSource )
    {
        case MIDI:
        case CD:

```

```

case LINE:
case MIC:
    if( ( bInL & bMix4SourceBits[ iSource ] ) ==
        bMix4SourceBits[ iSource ] ) return CH_BOTH;

break;
case MIDI_L:
case CD_L:
case LINE_L:
    if( bInL & bMix4SourceBits[ iSource ] ) return CH_LEFT;
break;
case MIDI_R:
case CD_R:
case LINE_R:
    if( bInL & bMix4SourceBits[ iSource ] ) return CH_RIGHT;
break;
}
return CH_NONE;
}

/*****
/* 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
*****/
WORD mix4_GetADCSOURCE_R( WORD iSource )
{ BYTE bInR;

    bInR = ( BYTE )mix_Read( MIX4_ADCSOURCE_R );

```

```

switch( iSource )
{
    case MIDI:
    case CD:
    case LINE:
    case MIC:
        if( ( bInR & bMix4SourceBits[ iSource ] ) ==
            bMix4SourceBits[ iSource ] ) return CH_BOTH;

        break;
    case MIDI_L:
    case CD_L:
    case LINE_L:
        if( bInR & bMix4SourceBits[ iSource ] ) return CH_LEFT;
        break;
    case MIDI_R:
    case CD_R:
    case LINE_R:
        if( bInR & bMix4SourceBits[ iSource ] ) return CH_RIGHT;
        break;
    }
    return CH_NONE;
}

/*****
/* mix4_SetOUTSource : Set output(playback) source */
/*-----**
/* Input : iSource - Source */
/*          iState - Switch source on or off */
*****/
VOID mix4_SetOUTSource( WORD iSource, WORD iState )

```

```

{
    BYTE bIn;
    bIn = ( BYTE )mix_Read( MIX4_OUTSOURCE );
    switch( iSource )
    {
        case CD:      case CD_L:      case CD_R:
        case LINE:    case LINE_L:    case LINE_R:
        case MIC:
            if( iState ) bIn |= bMix4SourceBits[ iSource ]; /* Set */
            else         bIn &= ~bMix4SourceBits[ iSource ]; /* Clear */
    }
    mix_Write( MIX4_OUTSOURCE, bIn );
}

/*****
/* 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 */
*****/
WORD mix4_GetOUTSource( WORD iSource )
{
    BYTE bIn;

    bIn = ( BYTE )mix_Read( MIX4_OUTSOURCE );

    switch( iSource )
    {
        case CD:
        case LINE:
        case MIC:

```

```

        if( ( bIn & bMix4SourceBits[ iSource ] ) )
            return CH_BOTH;
        break;
    case CD_L:
    case LINE_L:
        if( ( bIn & bMix4SourceBits[ iSource ] ) )
            return CH_LEFT;
        break;
    case CD_R:
    case LINE_R:
        if( ( bIn & bMix4SourceBits[ iSource ] ) )
            return CH_RIGHT;
        break;
    }
    return CH_NONE;
}

```

```

/*****
/* mix4_SetADCGain : Set recording preamplifier */
/**-----**/
/* Input : iGainL, iGainR - 0 : 0 dB */
/*          1 : 6 dB */
/*          2 : 12 dB */
/*          3 : 18 dB */
/*****

```

```

VOID mix4_SetADCGain( WORD iGainL, WORD iGainR )
{ BYTE bIn, bGainL, bGainR;

```

```

    bGainL = ( BYTE )( iGainL &= 0x03 );
    bGainL <<= 6;

```

```

    bGainR = ( BYTE )( iGainR &= 0x03 );

```



```

bGainR <= 6;

bIn = ( BYTE )( ( mix_Read( MIX4_ADCGAIN_L ) & 0x3F ) | bGainL);
mix_Write( MIX4_ADCGAIN_L, bIn );

bIn = ( BYTE )( ( mix_Read( MIX4_ADCGAIN_R ) & 0x3F ) | bGainR );
mix_Write( MIX4_ADCGAIN_R, bIn );
}

/*****
/* 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 values 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! */
*****/
WORD mix4_GetADCGain( WORD iChannel )
{
    BYTE bGainL, bGainR;

    bGainL = ( BYTE )mix_Read( MIX4_ADCGAIN_L );
    bGainL &= 0xC0;
    bGainL >= 6;

    bGainR = ( BYTE )mix_Read( MIX4_ADCGAIN_R );
    bGainR &= 0xC0;

```

```

bGainR >= 6;

switch (iChannel )
{
    case CH_LEFT:  return bGainL;
    case CH_RIGHT: return bGainR;
    default:  return MAKEWORD( bGainL, bGainR );
}
}

/*****
/* mix4_SetOUTGain : Set output preamplifier */
/**-----**/
/* Input : iGainL, iGainR - 0 : 0 dB */
/*          1 : 6 dB */
/*          2 : 12 dB */
/*          3 : 18 dB */
/*****/
VOID mix4_SetOUTGain( WORD iGainL, WORD iGainR )
{ BYTE bIn, bGainL, bGainR;

    bGainL = ( BYTE )( ( iGainL & 0x03 ) << 6 );
    bGainR = ( BYTE )( ( iGainR & 0x03 ) << 6 );

    bIn = ( BYTE )( ( mix_Read( MIX4_OUTGAIN_L ) & 0x3F ) | bGainL );
    mix_Write( MIX4_OUTGAIN_L, bIn );

    bIn = ( BYTE )( ( mix_Read( MIX4_OUTGAIN_R ) & 0x3F ) | bGainR );
    mix_Write( MIX4_OUTGAIN_R, bIn );
}

/*****/

```

```

/* mix4_GetOUTGain : Get output preamplifier */
/**-----**/
/* Input : iChannel - CH_LEFT : Gain for left channel */
/*          CH_RIGHT : Gain for right channel */
/*          andere : Gain for both channels */
/* Output : Current preamplifier value */
/*          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! */
/*****/
WORD mix4_GetOUTGain( WORD iChannel )
{
    BYTE bGainL, bGainR;

    bGainL = ( BYTE )mix_Read( MIX4_OUTGAIN_L );
    bGainL &= 0xC0;
    bGainL >>= 6;
    bGainR = ( BYTE )mix_Read( MIX4_OUTGAIN_R );
    bGainR &= 0xC0;
    bGainR >>= 6;

    switch ( iChannel )
    {
        case CH_LEFT: return bGainL;
        case CH_RIGHT: return bGainR;
        case CH_BOTH: return MAKEWORD( bGainL, bGainR );
    }
    return 0;
}

/*****/

```

```

/* mix4_SetAGC : Set / clear Automatic Gain Control */
/**-----**/
/* Input : iState - == 0 : Clear AGC ( 0 dB ) */
/*          <> 0 : Set AGC ( 20dB ) */
/**-----**/
/* Info : - AGC is used for recordings with a microphone and */
/*          permits the addition of a preamplifier that */
/*          amplifies the microphone signal by 20 decibels. */
/******/
VOID mix4_SetAGC( WORD iState )
{
    BYTE bIn;
    bIn = ( BYTE )mix_Read( MIX4_AGC );
    mix_Write( MIX4_AGC, iState ?
                ( bIn | MIX4_AGCON ) :
                ( bIn & ~MIX4_AGCON ) );
}

/******/
/* mix4_GetAGC : Get Automatic Gain Control */
/**-----**/
/* Output: TRUE : AGC set */
/*          FALSE : AGC cleared */
/******/
WORD mix4_GetAGC( VOID )
{
    return ( ( mix_Read( MIX4_AGC ) & MIX4_AGCON ) != 0 );
}

/******/
/* mix4_SetTreble : Set treble */
/**-----**/

```

```

/* Input : iTrebleL, iTrebleR : Set treble for left and right output- */
/*                                     channels. Values from 0 to 15      */
/*                                     correspond to -14 decibels to 14    */
/*                                     decibels in 2 decibel increments.    */
/*-----*/
/* Info : The Sound Blaster 16 card has a preamplifier                  */
/*         whose sound attributes can be modified.                      */
/*-----*/
VOID mix4_SetTreble( WORD iTrebleL, WORD iTrebleR )
{ BYTE bIn, bTrebleL, bTrebleR;

bTrebleL = ( BYTE )(iTrebleL << 4);
  bIn = ( BYTE )( ( mix_Read( MIX4_TREBLE_L ) & 0xF0 ) | bTrebleL );
  mix_Write( MIX4_TREBLE_L, bIn );

  bTrebleR = ( BYTE )(iTrebleR << 4);
  bIn = ( BYTE )( ( mix_Read( MIX4_TREBLE_R ) & 0xF0 ) | bTrebleR );
  mix_Write( MIX4_TREBLE_R, bIn );
}

/*****
/* mix4_GetTreble : Get current treble setting                          */
/*-----*/
/* Input : iChannel : CH_LEFT : Gain for left channel                  */
/*                                     CH_RIGHT : Gain for right channel */
/*                                     CH_BOTH : 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!                             */

```

```

/*****
WORD mix4_GetTreble( WORD iChannel )
{ BYTE bTrebleL, bTrebleR;

    bTrebleL = ( BYTE )mix_Read( MIX4_TREBLE_L );
    bTrebleL &= 0xF0;
    bTrebleL >>= 4;

    bTrebleR = ( BYTE )mix_Read( MIX4_TREBLE_R );
    bTrebleR &= 0xF0;
    bTrebleR >>= 4;

    switch ( iChannel )
    {
        case CH_LEFT:  return bTrebleL;
        case CH_RIGHT: return bTrebleR;
        case CH_BOTH:  return MAKEWORD( bTrebleL, bTrebleR );
    }
    return 0;
}

/*****
/* mix4_SetBass : Set bass */
/*-----*/
/* Input : iBassL, iBassR : Set bass for left and right output */
/*                      channels. Values from 0 to 15 */
/*                      equal -14 to 14 decibels in */
/*                      increments of 2 decibels. */
/*-----*/
/* Info - The Sound Blaster 16 card has a preamplifier */
/*         whose sound attributes can be modified. */
/*****

```

```

VOID mix4_SetBass( WORD iBassL, WORD iBassR )
{ BYTE bIn, bBassL, bBassR;

    bBassL = ( BYTE )(iBassL << 4);
    bIn = ( BYTE )( ( mix_Read( MIX4_BASS_L ) & 0xF0 ) | bBassL );
    mix_Write( MIX4_BASS_L, bIn );
    bBassR = ( BYTE )(iBassR << 4);
    bIn = ( BYTE )( ( mix_Read( MIX4_BASS_R ) & 0xF0 ) | bBassR );
    mix_Write( MIX4_BASS_R, bIn );
}

/*****
/* mix4_GetBass : Get current bass setting */
/*-----*/
/* Input : iChannel : CH_LEFT : Gain for left channel */
/*          CH_RIGHT : Gain for right channel */
/*          CH_BOTH : 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! */
*****/
WORD mix4_GetBass( WORD iChannel )
{ BYTE bBassL, bBassR;

    bBassL = ( BYTE )mix_Read( MIX4_BASS_L );
    bBassL &= 0xF0;
    bBassL >= 4;
    bBassR = ( BYTE )mix_Read( MIX4_BASS_R );
    bBassR &= 0xF0;

```

```
bBassR >= 4;
switch (iChannel )
{
    case CH_LEFT:  return bBassL;
    case CH_RIGHT: return bBassR;
    case CH_BOTH:  return MAKEWORD( bBassL, bBassR );
}
return 0;
}
#endif
```