

```
/*  
WEFAX, FAX, SSTV, RTTY & CW software for Sound Blaster sound cards.
```

```
Written by Brian E. Cauchi 9H1JS, 1-Apr-1994.
```

```
SCOPE.CPP - Handle tuning oscilloscope display  
*/
```

```
#include"global.h"
```

```
struct SS { unsigned char Y, C; };
```

```
char *OffOn[2] = { "Off", "On" },
```

```
    *FilterMsg[4] = { "Off", "Low", "Medium", "High" },
```

```
    ScopeDirect=0, ScopeFilter=1, ScopeHoldoff=0;
```

```
int x1,x2,x3,x4,y;
```

```
void ControlPanels(void);
```

```
void ShowSettings(void);
```

```
void ScopeScales(void);
```

```
void ScopeScreen(struct SS *S);
```

```
int ScopeKeys(void);
```

```
void ScopeBackground(void);
```

```
int ScaleFreq(int f);
```

```
//-----  
// TUNING SCOPE
```

```
void TuningScope(void)
```

```
{  
    int a,b,c, z, level, clipping=0, memory, analyzer[256];
```

```
    char trend[256]; unsigned char far *wb; long n; float f, Fc=1900;
```

```
    if(Wave.Status!=WAVE_INPUT)  
        return;
```

```
    SetView(&Rx);
```

```
    ControlPanels();
```

```
    ScopeScales();
```

```
// clear analyzer data
```

```
for(a=0;a<256;a++)  
{  
    analyzer[a]=0;  
    trend[a]=-60;  
}
```

```
// main loop
```

```
for(;;)  
{  
    if(kbhit())  
    {  
        if(ScopeKeys())  
            break;  
    }  
    else  
        if(ScopeHoldoff)  
            delay(250 << (ScopeHoldoff-1)); // 0,250,500,1000ms
```

```
// wait for a bank swap
```

```
for(z=Wave.Bank;z==Wave.Bank;)  
    if(kbhit())  
        break;
```

```
    ScopeBackground();
```

```

// direct waveform display

if(ScopeDirect)
{
    wb = (Wave.Bank) ? Wave.Buffer+Wave.Size : Wave.Buffer;

    a = *wb++;

    setcolor(brGreen); setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
    moveto(x1, y - a);

    for(z=x1+1; z<=x2; z+=2)
        lineto(z, y - *wb++);
}

// prepare for plot, using most recent data

n = FdPtr->XmsFinish - (x2-x1+1);

if(n<0)
    continue; // not enough for complete trace

a = XmsGetByte(n);

setcolor(brWhite); setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
moveto(x1, y - a);

// process demodulator output trace & get analyzer data

for(z=x1+1; z<=x2; n++, z++)
{
    b=XmsGetNextByte();

    if(dem_Mode==dem_FFT)
    {
        for(a=0; a<8; a++, b>>=1) // process 8 frequency bands
            if(b&1)
            {
                if(! ScopeDirect)
                    line(z, y- (a*32+12), z, y- (a*32+20));

                for(c= 12; c<=20; c++)
                    ++analyzer[a*32+c];
            }
    }
    else
    {
        a = ( (a << ScopeFilter) - a + b ) >> ScopeFilter;

        if(! ScopeDirect)
            lineto(z, y - a);

        ++analyzer[a];
    }
}
« end for z=x1+1; z<=x2; n++, z++ »

// convert analyzer data to logarithmic scale & display

moveto(x3,y);

for(a=0; a<256; a++)
{
    f = 10 * log( ((float)analyzer[a]+0.1) / (x2-x1+1)); //dB

    if(dem_Mode==dem_FM)
        f+=10; // boost gain

// update waterfall data, before rescaling 'f'

if(dem_Mode!=dem_FFT)
{
    trend[a] -= 2; // 2dB decay per sweep

    if(f > trend[a])
        trend[a] = (trend[a] + f) / 2;
}

if(dem_Mode==dem_FM)

```

```

        f+=5;           // raise analyzer data above waterfall

f = ClipRange(f, -60, 0);

// display analyzer data

f *= (float)(x4-x3) / 60;

line(x3, y-a, x4+f, y-a);

analyzer[a] = 0;
} « end for a=0;a<256;a++ »

// smooth & display waterfall data

if(dem_Mode!=dem_FFT)
{
    setcolor(brCyan);
    setlinestyle(SOLID_LINE, 0, THICK_WIDTH);

    for(z=-60,a=1; a<255; a+=3)
    {
        for(b=-1,c=-60; b<2; b++)
            if(c < trend[a+b])
                c = trend[a+b];

        z = (z + c) / 2 - 2;    // shown 2dB less
        z = ClipRange(z, -60, 0);

        c = z * (float)(x4-x3) / 60;

        line(x3, y-a, x4+c, y-a);
    }
}

// baudot graticule

if((Mode==Baudot)&&(!ScopeDirect))
{
    setcolor(brYellow);
    setlinestyle(SOLID_LINE, 0, NORM_WIDTH);

    line(x1, y-128, x2, y-128);
    line(x3, y-128, x4, y-128);

    for(a=0,f=x1;f<x2;a++)
    {
        if(a%10==0)
            line(f,y-136,f,y-120);
        else
            line(f,y-132,f,y-124);

        f += (0.01/SamplingWindow);
    }
}

// get instantaneous audio level, frequency & clipping

clipping = GetInstantAFC(&level, &f);

if(dem_Mode==dem_FFT)
    Fc = Fc*0.67 + f*0.33;
else
    Fc = f;

// highlight Fc on analyzer

c = ScaleFreq(Fc);

if(dem_Mode!=dem_AM)
{
    setcolor(brRed);
    setlinestyle(SOLID_LINE, 0, THICK_WIDTH);
    line(x3, y-c, x4, y-c);
}

TextPanelAt(-10, 40, brWhite, Green, "%.0f Hz ", Fc);

```

```
#pragma warn -sig
memory = (100*(1+Rx.XmsFinish-Rx.XmsStart)) / (Xms.Pages * Xms.PageSize);
#pragma warn +sig
```

```
TextPanelAt(- 6, 35, brWhite, Green, "%i%% ", level);

c = (100.0 * clipping) / Wave.Size;
clipping /= 2;

if(c) TextPanelAt(- 6, 43, brWhite, Red, "Clip %i ", c);
else TextPanelAt(- 6, 43, brWhite, Green, " ");

TextPanelAt(- 2, 36, brWhite, Green, "%i%% ", memory);
TextPanelAt(- 2, 42, brWhite, Green, "Rx %lik, Tx %lik ",
(Rx.XmsFinish-Rx.XmsStart)>>10, (Tx.XmsFinish-Tx.XmsStart)>>10);
} « end for ;; »
} « end TuningScope »
```

//-----

```
void ControlPanels(void)
```

```
{
char *p;

ClearDesktop();

InitTextPanel(RxStatus_Clr, "ø TUNING SCOPE ø");
```

```
// scope controls
```

```
TextPanelAt(- 10, brWhite, Green, "\
ø Scope Controls ø Centre frequency:");

TextPanelAt(- 9, brYellow, Blue, "\
Direct Ü Filter Ü Holdoff Ü Restart Ü Ü"); /*
012345678--012345678--012345678--012345678--012345678--012345678*/
```

```
// mixer controls
```

```
TextPanelAt(- 6, brWhite, Green, "\
ø Mixer Controls ø Input level:");

TextPanelAt(- 5, brYellow, Blue, "\
Mic i/p Ü Line i/p Ü Tx level Ü Monitor Ü Mute Ü Reset"); /*
012345678--012345678--012345678--012345678--012345678--012345678*/
```

```
// mode settings
```

```
TextPanelAt(- 2, brWhite, Green, "\
ø Mode Settings ø Memory usage:");

p = (ModeIsVideo()) ? StdModes[StdMode].Desc : "\0";

TextPanelAt(- 1, brYellow, Blue, "\
Demodulator: %- 4s Mode: %s %s",
ModeMsg[dem_Mode], ModeMsg[Mode], p);
```

```
ShowSettings();
} « end ControlPanels »
```

//-----

```
void ShowSettings(void)
```

```
{
TextPanelAt(- 8, brYellow, Blue, "\
%- 3s Ü %- 6s Ü %- 6s Ü Ü",
OffOn[ScopeDirect], FilterMsg[ScopeFilter], FilterMsg[ScopeHoldoff]);

if(ExternalMixerControl)
TextPanelAt(- 4, brYellow, Blue, "\
F1----F2 Ü F3----F4 Ü F5----F6 Ü F7----F8 Ü F10---- Ü F12");
else
TextPanelAt(- 4, brYellow, Blue, "\
F1-%02i-F2 Ü F3-%02i-F4 Ü F5-%02i-F6 Ü F7-%02i-F8 Ü F10 %- 3s Ü F12",
MixerAdjust(MIC_IN, ''),
MixerAdjust(LINE_IN, ''),
MixerAdjust(TX_OUT, ''));
```

```

        MixerAdjust(MONITOR_OUT, 1),
        OffOn[Mixer.Mute>=0] );
    }
}

//-----

void ScopeScales(void)
{
    int a,b; float f;

// left & right margins, vertical centering

    a = 16 + textwidth(" ") * 5; b = getmaxx() - (8 + a);
    x1 = 8; x2 = (b*3)/4;

    x3 = x2 + a; x4 = getmaxx()-8;    // analyzer

    a = (3*textheight(" ") / 2;
    y = getmaxy() - 11*a;            // less text panels
    y = 256 + (y-256) / 2;

// clear rectangle

    setfillstyle(SOLID_FILL, Black);
    bar(x1-8, y-272, x4+8, y+48+textheight(" "));

// greyscale bar

    a=Mode;
    Mode=FaxBw;
    InitPalette();
    Mode=a;

    for(a=0;a<264;a++)
    {
        if(a>239)
            b=239;
        else
            b=a;

        setcolor(b);
        line(x2+8, y-a, x2+8 + textwidth(" ") * 5, y-a);
    }

// modulation scale

    setcolor(Cyan); a=- textheight(" ")/2;

    switch(dem_Mode)
    {
        case dem_AM:

            outtextxy(x2+8,y+a-251,"- 100-");
            outtextxy(x2+8,y+a-140," % ");
            outtextxy(x2+8,y+a-16,"- 0 -");
            break;

        case dem_FM:

            outtextxy(x2+8,y+a-237,"- 2.3-");
            outtextxy(x2+8,y+a-140," kHz ");
            outtextxy(x2+8,y+a-37,"- 1.5-");
            outtextxy(x2+8,y+a-4,"- 1.2-");
            break;

        case dem_FFT:

            if(Mode==Baudot)
            {
                outtextxy(x2+8,y+a-128,"- 1.9-");
                outtextxy(x2+8,y+a-64," kHz ");
            }

            if(Mode==Morse)
            {
                for(b=0;b<8;b++)
                {

```

```

    char s[8];

    sprintf(s, "-%i-", (Wave.Rate/32)*(b+1)); s[5]='\0';

    outtextxy(x2+8,y+a-(32*b+16),s);
}
outtextxy(x2+8,y-a, "- Hz-");
}
break;
} « end switch dem_Mode »

```

```
// rectangular frame
```

```

setcolor(brYellow); setlinestyle(SOLID_LINE, 0, THICK_WIDTH);

rectangle(x1-5, y+5, x2+5, y-260);

rectangle(x3-5, y+5, x4+5, y-260); // analyzer

setcolor(Black); rectangle(x1-2, y+2, x2+2, y-257);

setcolor(Black); rectangle(x3-2, y+2, x4+2, y-257); // analyzer

```

```
// draw X scales
```

```

setcolor(brWhite); setlinestyle(SOLID_LINE, 0, NORM_WIDTH);

for(f=x1;f<x2;)
{
    line(f,y+16,f,y+31);

    if(ScopeDirect)
        f += 0.002 * Wave.Rate; // 1ms/div
    else
        f += (((dem_Mode==dem_FFT) ? 0.1 : 0.01) / SamplingWindow);
}

if(ScopeDirect)
    outtextxy(x1, y+40, "Time 1ms/div ->");
else
{
    if(dem_Mode==dem_FFT)
        outtextxy(x1, y+40, "Time 100ms/div ->");
    else
        outtextxy(x1, y+40, "Time 10ms/div ->");
}

for(f=x3;f<x4;)
{
    line(f,y+16,f,y+31);
    f += 25;
}

outtextxy(x3, y+40, "Level/ dB ->");
} « end ScopeScales »

```

```
//-----
```

```

void ScopeScreen(struct SS *S)
{
    int i,j;

    if(ScopeDirect)
    {
        setfillstyle(SOLID_FILL, Black);
        bar(x1, y, x2, y-255);
    }

    for(i=j=0; S[i].Y!=0; i++)
    {
        setfillstyle(SOLID_FILL, S[i].C);

        if(! ScopeDirect)
            bar(x1, y-j, x2, y-S[i].Y); // scope

        bar(x3, y-j, x4, y-S[i].Y); // analyzer

        j = S[i].Y+1;
    }
}

```

```

} « end ScopeScreen »

//-----

int ScopeKeys(void)
{
    int a;

    a=GetKey();

    if(a>0)
        a=toupper(a);

    switch(a)
    {

        // scope controls

        case 'D': ScopeDirect ^= 1; ScopeScales();      break;

        case 'F': ScopeFilter = (++ScopeFilter)&3;      break;

        case 'H': ScopeHoldoff = (++ScopeHoldoff)&3;    break;

        case 'R': disable();
                    Rx.XmsStart=Rx.XmsFinish=Rx.FrameStart=Rx.FrameFinish=0;
                    enable();

                    Blip();
                    break;

        case Escape: return 1;

        // mixer controls

        case F1:
        case F2: MixerAdjust(MIC_IN, a- F1); break;

        case F3:
        case F4: MixerAdjust(LINE_IN, a- F3); break;

        case F5:
        case F6: MixerAdjust(TX_OUT, a- F5); break;

        case F7:
        case F8: MixerAdjust(MONITOR_OUT, a- F7); break;

        case F10: MixerMuteToggle(); break;

        case F12: MixerReset(); Blip(); break;
    } « end switch a »

    ShowSettings();

    return 0;
} « end ScopeKeys »

//-----

void ScopeBackground(void)
{
    switch(Mode)
    {
        case Wefax:
            {
                struct SS S[] = {
                    { 16,  Blue  },
                    { 251, brBlack },
                    { 255,  Red   },
                    { 0,0 }      };

                ScopeScreen(S);
            }
            break;

        case Baudot:
            {

```

```

    struct SS S[] = {
        { 255,   brBlack },
        { 0,0 }
    };

    ScopeScreen(S);
}
break;

case Morse:
{
    struct SS S[] = {
        { mfb*32,   Blue },
        { (mfb+1)*32, brBlack },
        { 255,   Blue },
        { 0,0 }
    };

    ScopeScreen(S);
}
break;

default:

    struct SS S[] = {
        { 3,   Red },
        { 36,  Blue },
        { 237, brBlack },
        { 255, Red },
        { 0,0 }
    };

    ScopeScreen(S);
} « end switch Mode »
} « end ScopeBackground »

```

//-----

```

int ClipRange(int val, int min, int max)
{
    if(val < min)
        val = min;
    else
        if(val > max)
            val = max;

    return val;
}

```

//-----

```

int ScaleFreq(int f)
{
    int a,b,c;

    switch(Mode) // a=f_min, b=f_max
    {
        case Morse:  a=125; b=2125; break;

        case Baudot: a=1400; b=2400; break;

        default: a=1350; b=2375;
    }

    c = (256.0 / (b-a)) * (f-a);

    c = ClipRange(c, 1, 254);

    return c;
} « end ScaleFreq »

```

/*-----
Calculates instantaneous amplitude, frequency & clipping
*/

```

int GetInstantAFC(int *level, float *freq)
{
    int a,b,c, z, clipping=0;

    unsigned char far *wb; long n,o;

    // calculate audio level & frequency

```

```

wb = (Wave.Bank) ? Wave.Buffer+Wave.Size : Wave.Buffer;
for(b=c=z=0,n=o=0; z<Wave.Size; z++)
{
a = *wb++;
c = abs( (int)a - 0x80);
if(c>=127)
++clipping;

n += c;           // sum amplitudes for level

if((a^b)&0x80)    // count edges for frequency
{
++o;
b=a;
}
}

// convert number of edges to frequency
*freq = (o * Wave.Rate) / (2.0 * Wave.Size);

#pragma warn -sig
*level = (100L * n) / (Wave.Size * 0x80L);
*level *= 1.625;
#pragma warn +sig

return clipping;
} « end GetInstantAFC »

```