

```
/* This file is hand-coded. It provides the headers to the  
   UI simulation in korgsim.c.code.pdf
```

```
*/
```

```
void initDisplay(void);  
void ledLowOn(void);  
void ledLowOff(void);  
void ledExactOn(void);  
void ledExactOff(void);  
void ledHighOn(void);  
void ledHighOff(void);  
void showNote(char * note);  
void showFreq(char * freq);  
void showDev(float dev);  
void showFlat(void);  
void showSharp(void);  
void showNone(void);  
void spkrOn(void);  
void spkrOff(void);  
void spkFreq(float freq);  
void showInputFreq(short freq);
```

```
/* This file is hand-coded. It provides a rudimentary text-based
   interface and is for the target system replaced by the driver
   routines for the LCD display, the LEDs and the input keys. It
   further provides a simulation of the frequency detection con-
   sisting of a loop providing a continuous sweep between 400
   and 700 Hz.
*/
```

```
#include <string.h>
#include <mem.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <bios.h>
#include <conio.h>
#include <string.h>
#include <dos.h>
#include "korgsim.h"
#include "korg.h"
```

```
static short spkrFreq = 1000;
static short spkrlsOn = 0;
static short displayIsOn = 0;
static char * freqStr = " ";
void showF(void);
```

```
void main(void)
{
  int key, oldkey, freq, freqadd;
  freq = 4000;
  freqadd = 1;
  oldkey = 0;
  korg_boot();
  while(1){
    key = bioskey(2);
    if((key & 1) != (oldkey & 1))
      {if(key & 1)krg_ONOFFFPressed();}
    if((key & 2) != (oldkey & 2))
      {if(key & 2)krg_CALIBupPressed(); else krg_CALIBupReleased();}
    if((key & 4) != (oldkey & 4))
      {if(key & 4)krg_CALIBdownPressed(); else krg_CALIBdownReleased();}
    if((key & 8) != (oldkey & 8))
      {if(key & 8)krg_SOUNDPressed(); else krg_SOUNDReleased();}
```

```
oldkey = key;
delay(5000);
if(freq > 7000) freqadd = -1;
if(freq < 4000) freqadd = 1;
freq += freqadd;
Krg_HorizontalSliderValue(freq/ 10);
processTimers();
}
}
```

```
void spkrOn(void)
{extern short spkrFreq, spkrlsOn;
 sound(spkrFreq);
 spkrlsOn = 1;
}
```

```
void spkrOff(void)
{extern short spkrlsOn;
 nosound();
 spkrlsOn = 0;
}
```

```
void showInputFreq(short freq)
{if(displayIsOn)
 {textcolor(LIGHTGRAY);
 gotoxy(15,11);
 cprintf("Measured Frequency: %d Hz", freq);
 }}
```

```
void spkFreq(float freq)
{extern short spkrFreq, spkrlsOn;
 if(spkrlsOn)nosound();
 spkrFreq = (short)(freq + 0.5);
 if(spkrlsOn)sound(spkrFreq);
}
```

```
void ledLowOff(void)
{gotoxy(20,3);
 textcolor(DARKGRAY);
 cputs("\333\333");
```

```
}
void ledLowOn(void)
{gotoxy(20,3);
 textcolor(LIGHTRED);
 cputs("\33\333");
}
void ledExactOff(void)
{gotoxy(40,3);
 textcolor(DARKGRAY);
 cputs("\33\333");
}
void ledExactOn(void)
{gotoxy(40,3);
 textcolor(LIGHTGREEN);
 cputs("\33\333");
}
void ledHighOff(void)
{gotoxy(60,3);
 textcolor(DARKGRAY);
 cputs("\33\333");
}
void ledHighOn(void)
{gotoxy(60,3);
 textcolor(LIGHTRED);
 cputs("\33\333");
}

void showFreq(char * freq)
{
 strcpy(freqStr, freq);
 if(displayIsOn)showF();
}

void showF(void)
 {textcolor(WHITE);
 gotoxy(15,6);
 cputs(freqStr);}

void showDev(float fdev)
{static int oldval;
 int dev;
 if(displayIsOn)
 {dev = (int)(fdev + 0.5);
  if(dev != oldval){
```

```
    textcolor(BLACK);
    gotoxy(15 + (oldval + 50)/2, 9);
    cputs("\333");
    textcolor(YELLOW);
    gotoxy(15 + (dev + 50) /2, 9);
    cputs("\333");
    oldval = dev;
}
}
```

```
void showNote(char * note)
{textcolor(WHITE);
 gotoxy(60,6);
 cputs(note);
}
```

```
void showFlat(void)
{textcolor(WHITE);
 gotoxy(61,6);
 cputs("b");
}
```

```
void showSharp(void)
{textcolor(WHITE);
 gotoxy(61,6);
 cputs("#");
}
```

```
void showNone(void)
{textcolor(BLACK);
 gotoxy(61,6);
 cputs("\333");
}
```

```
void displayOn(void)
{textcolor(WHITE);
 gotoxy(17,3);
 cputs("b");
 gotoxy(40,2);
 cputs("V");
 gotoxy(63,3);
 cputs("#");
 gotoxy(20,6);
```



```
#ifndef _KORG_H
#define _KORG_H

#ifdef _korg_c
char convstr[] = " ";
/* global variable for counter RPitch */
unsigned short CI_value = 440;
/* global variables for calculation MCalc */
unsigned short cv_F;
unsigned short cv_R;
float cv_X;
/* bitmap indicating undefined input vars */
unsigned short cv_UNDEF = 255;
/* global variables for calculation Scalc */
unsigned short scl_R;
float scl_X;
/* bitmap indicating undefined input vars */
unsigned short scl_UNDEF = 7;
/* global variables for timer TI */
unsigned short TI_value;
unsigned short TI_count = 16;
#else
extern char convstr[];
/* global variable for counter RPitch */
extern unsigned short CI_value;
/* global variables for calculation MCalc */
extern unsigned short cv_F;
extern unsigned short cv_R;
extern float cv_X;
/* bitmap indicating undefined input vars */
extern unsigned short cv_UNDEF;
/* global variables for calculation Scalc */
extern unsigned short scl_R;
extern float scl_X;
/* bitmap indicating undefined input vars */
extern unsigned short scl_UNDEF;
/* global variables for timer TI */
extern unsigned short TI_value;
extern unsigned short TI_count;
#endif

void korg_boot(void);
```

```
long str2int(char * s);
float str2float(char * s);
void Krg_HorizontalSliderValue(unsigned short param);

void krg_CALIBdownPressed(void);

void krg_CALIBdownReleased(void);

void krg_CALIBupPressed(void);

void krg_CALIBupReleased(void);

void krg_ONOFFPressed(void);

void krg_SOUNDPressed(void);

void krg_SOUNDReleased(void);

void processTimers(void);

#endif
```

```
#define _korg_c
#include <korg.h>
#include <stdlib.h>
#include <cv.h>
#include <kr1.h>
#include <kr2.h>
#include <scl.h>
#include <snd.h>
#include <tl.h>
#include "korgsim.h"
```

```
#undef _korg_c
```

```
void korg_boot(void)
{
/* call boot function for State Machine korgmainctrl */
kr2_boot();
/* call boot function for State Machine korgcalctrl */
kr1_boot();
/* distribute the initial value of counter RPitch */
showFreq(itoa(CI_value, convstr, 10));
cv_UNDEF &= 191;
cv_R = CI_value;
scl_UNDEF &= 6;
scl_R = CI_value;
/* call boot function for State Machine KrogSndCtrl */
snd_boot();
}
```

```
void Krg_HorizontalSliderValue(unsigned short param)
{
kr2_processEventUnsignedShort(kr2_ev_freq_, param);
}
```

```
void krg_CALIBdownPressed(void)
{
kr2_processEvent(kr2_ev_CalDownPr);
}
```

```
void krg_CALIBdownReleased(void)
```

```
{
  krg2_processEvent(krg2_ev_CalDownRel);
}
```

```
void krg_CALIBupPressed(void)
{
  krg2_processEvent(krg2_ev_CalUpPr);
}
```

```
void krg_CALIBupReleased(void)
{
  krg2_processEvent(krg2_ev_CalUpRel);
}
```

```
void krg_ONOFFPressed(void)
{
  krg2_processEvent(krg2_ev_OnOff);
}
```

```
void krg_SOUNDPressed(void)
{
  krg2_processEvent(krg2_ev_SoundPressed);
}
```

```
void krg_SOUNDReleased(void)
{
  krg2_processEvent(krg2_ev_SoundReleased);
}
```

```
void processTimers(void)
{ /* timer processing should be called every 50 ms*/
  if(TI_value){ /* timer active? */
    TI_value--; /* decrement timer value */
    if(!TI_value){ /* timer due ?*/
      krg2_processEvent(krg2_ev_TimerTick);
      krg1_processEvent(krg1_ev_TimerTick);
    }
  }
}
```

```
long str2int(char * s)
{
    unsigned char i = 0;
    char factor = 1;
    unsigned long val = 0;
    while(1) {
        if(s[i] == '-') {
            factor = -1;
        } else
        if(s[i]>='0' & s[i]<='9') {
            val = val * 10 + s[i] - '0';
        } else
        if(s[i]==0) {
            return val*factor;
        }
        i++;
    }
}
```

```
float str2float(char * s)
{
    unsigned char i = 0;
    char factor = 1;
    float val = 0;
    float fact = 10;
    while(1) {
        if(s[i] == '-') {
            factor = -1;
        } else
        if (s[i] == '.') {
            fact = 0.1;
        } else
        if(s[i]>='0' & s[i]<='9') {
            if(fact > 1) {
                val = val * 10 + (s[i] - '0');
            } else {
                val = val + fact * (s[i] - '0');
                fact = fact * 0.1;
            }
        } else
        if(s[i]==0){
            return val*factor;
        }
    }
}
```

```
i++;  
}  
}
```

```
#ifndef _KRG1_H
#define _KRG1_H
typedef enum
    {
        krg1_ev_CalDwPr,
        krg1_ev_TimerTick,
        krg1_ev_CalDwRI,
        krg1_ev_CalUpPr,
        krg1_ev_CalUpRI,
        krg1_numberOfEvents
    } krg1_event_id ;

typedef enum
    {
        krg1_Up,
        krg1_DownDown,
        krg1_UpDown,
        krg1_numberOfStates
    } krg1_state_id ;

void krg1_boot(void);

void krg1_processEvent(krg1_event_id event);
#endif
```

```
#include <krg1.h>
#include <krg2.h>
#include <korg.h>
#include "korgsim.h"
```

```
static const krg1_state_id krg1_nextState[krg1_numberOfEvents *
krg1_numberOfStates] =
/*          krg1_ev_CalDwPr krg1_ev_TimerTick krg1_ev_CalDwRI
krg1_ev_CalUpPr krg1_ev_CalUpRI*/{
/*krg1_Up      */ krg1_DownDown,      krg1_Up,      krg1_Up,
krg1_UpDown,      krg1_Up,
/*krg1_DownDown */ krg1_DownDown, krg1_DownDown,      krg1_Up,
krg1_DownDown, krg1_DownDown,
/*krg1_UpDown   */ krg1_UpDown, krg1_UpDown, krg1_UpDown,
krg1_UpDown,      krg1_Up};
```

```
enum {krg1_a_no_action,
      krg1_a_DecCal,
      krg1_a_IncCal,
      krg1_a_TimerLong,
      krg1_a_TimerShort,
      krg1_a_TimerStart,
      krg1_a_TimerStop,
      krg1_a_numberOfActions};
```

```
static const short krg1_actions[krg1_numberOfEvents * krg1_numberOfStates] =
/*          krg1_ev_CalDwPr krg1_ev_TimerTick krg1_ev_CalDwRI
krg1_ev_CalUpPr krg1_ev_CalUpRI*/{
/*krg1_Up      */          1,          0,          0,          4,          0,
/*krg1_DownDown*/          0,          7,          11,          0,          0,
/*krg1_UpDown  */          0,          14,          0,          0,          11};
```

```
static const short actionSequences[18] =
{
  krg1_a_no_action,
/* 1*/ krg1_a_DecCal /* Up:CalDwPr->DownDown */,
  krg1_a_TimerStart,
  krg1_a_no_action,
/* 4*/ krg1_a_IncCal /* Up:CalUpPr->UpDown */,
  krg1_a_TimerStart,
  krg1_a_no_action,
/* 7*/ krg1_a_DecCal /* DownDown:TimerTick->DownDown */,
```

```
    krg1_a_TimerShort,
    krg1_a_TimerStart,
    krg1_a_no_action,
/* 11*/ krg1_a_TimerLong /* DownDown:CalDwRI->Up, UpDown:CalUpRI->Up */,
    krg1_a_TimerStop,
    krg1_a_no_action,
/* 14*/ krg1_a_IncCal /* UpDown:TimerTick->UpDown */,
    krg1_a_TimerShort,
    krg1_a_TimerStart,
    krg1_a_no_action
};
```

```
static krg1_state_id actualState, previousState;
```

```
void krg1_processActions(short actionIndex);
```

```
void krg1_boot(void)
{
    actualState = krg1_Up;
}
```

```
void krg1_processEvent(krg1_event_id event)
{
    previousState = actualState;
    actualState = krg1_nextState[previousState * krg1_numberOfEvents + event];
    krg1_processActions(krg1_actions[previousState * krg1_numberOfEvents +
event]);
}
```

```
void krg1_processActions(short actionIndex)
{while(actionSequences[actionIndex])
{switch(actionSequences[actionIndex]) {
    case krg1_a_DecCal:
        /* decrement counter RPitch */
        CI_value = --CI_value < 410 ? 480 : CI_value;
        showFreq(itoa(CI_value, convstr, 10));
        cv_UNDEF &= 191;
        cv_R = CI_value;
        scl_UNDEF &= 6;
        scl_R = CI_value;
        krg2_processEvent(krg2_ev_calc);
        break;
    case krg1_a_IncCal:
        /* increment counter RPitch */
```

```
    CI_value = ++CI_value > 480 ? 410 : CI_value;
    showFreq(itoa(CI_value, convstr, 10));
    cv_UNDEF &= 191;
    cv_R = CI_value;
    scl_UNDEF &= 6;
    scl_R = CI_value;
    krg2_processEvent(krg2_ev_calc);
    break;
case krg1_a_TimerLong:
    /* set timer TI interval to 800 ms */
    TI_count = 16;
    break;
case krg1_a_TimerShort:
    /* set timer TI interval to 150 ms */
    TI_count = 3;
    break;
case krg1_a_TimerStart:
    /* start timer TI */
    TI_value = TI_count;
    break;
case krg1_a_TimerStop:
    /* stop timer TI */
    TI_value = 0;
    break;
default: /* Error Handling not yet implemented */
    break;
}
actionIndex++;
}
}
```

```
#ifndef _KRG2_H
#define _KRG2_H
typedef enum
{
    krg2_ev_OnOff,
    krg2_ev_SoundPressed,
    krg2_ev_SoundReleased,
    krg2_ev_LastNote,
    krg2_ev_CalDownPr,
    krg2_ev_CalDownRel,
    krg2_ev_CalUpPr,
    krg2_ev_CalUpRel,
    krg2_ev_TimerTick,
    krg2_ev_dev_,
    krg2_ev_HiOn,
    krg2_ev_HiOff,
    krg2_ev_LoOn,
    krg2_ev_LoOff,
    krg2_ev_ExOn,
    krg2_ev_ExOff,
    krg2_ev_freq_,
    krg2_ev_calc,
    krg2_numberOfEvents
} krg2_event_id ;

typedef enum
{
    krg2_Off,
    krg2_Measure,
    krg2_SoundUp,
    krg2_SoundDwn,
    krg2_numberOfStates
} krg2_state_id ;

void krg2_boot(void);

void krg2_processEvent(krg2_event_id event);
void krg2_processEventFloat(krg2_event_id event, float param);
void krg2_processEventUnsignedShort(krg2_event_id event, unsigned short
param);
#endif
```

```
#include <krg2.h>
#include <cv.h>
#include <krg1.h>
#include <scl.h>
#include <snd.h>
#include <korg.h>
#include "korgsim.h"
```

```
static const krg2_state_id krg2_nextState[krg2_numberOfEvents *
krg2_numberOfStates] =
    {krg2_Measure, krg2_Off, krg2_Off, krg2_Off, krg2_Off, krg2_Off, krg2_Off,
krg2_Off, krg2_Off, krg2_Off, krg2_Off,
    krg2_Off, krg2_Off, krg2_Off, krg2_Off, krg2_Off, krg2_Off, krg2_Off, krg2_Off,
krg2_SoundDwn, krg2_Measure,
    krg2_Measure, krg2_Measure, krg2_Measure, krg2_Measure, krg2_Measure,
krg2_Measure, krg2_Measure, krg2_Measure,
    krg2_Measure, krg2_Measure, krg2_Measure, krg2_Measure, krg2_Measure,
krg2_Measure, krg2_Measure, krg2_Off,
    krg2_SoundDwn, krg2_SoundUp, krg2_SoundUp, krg2_SoundUp, krg2_SoundUp,
krg2_SoundUp, krg2_SoundUp, krg2_SoundUp,
    krg2_SoundUp, krg2_SoundUp, krg2_SoundUp, krg2_SoundUp, krg2_SoundUp,
krg2_SoundUp, krg2_SoundUp, krg2_SoundUp,
    krg2_SoundUp, krg2_SoundDwn, krg2_SoundDwn, krg2_SoundUp,
krg2_Measure, krg2_SoundDwn, krg2_SoundDwn, krg2_SoundDwn,
    krg2_SoundDwn, krg2_SoundDwn, krg2_SoundDwn, krg2_SoundDwn,
krg2_SoundDwn, krg2_SoundDwn, krg2_SoundDwn,
    krg2_SoundDwn, krg2_SoundDwn, krg2_SoundDwn, krg2_SoundDwn
};
enum {krg2_a_no_action,
    krg2_a_calcDev,
    krg2_a_calcFreq,
    krg2_a_CalDownPr,
    krg2_a_CalDownRel,
    krg2_a_CalUpPr,
    krg2_a_CalUpRel,
    krg2_a_dev_,
    krg2_a_displayOff,
    krg2_a_displayOn,
    krg2_a_ExOff,
    krg2_a_ExOn,
    krg2_a_FirstNote,
    krg2_a_freq_,
```

```
    krg2_a_HiOff,  
    krg2_a_HiOn,  
    krg2_a_LoOff,  
    krg2_a_LoOn,  
    krg2_a_NextNote,  
    krg2_a_Recalc,  
    krg2_a_SpkrOff,  
    krg2_a_SpkrOn,  
    krg2_a_TimerStart,  
    krg2_a_TimerStop,  
    krg2_a_ToggleSound,  
    krg2_a_numberOfActions};
```

```
static const short krg2_actions[krg2_numberOfEvents * krg2_numberOfStates] =  
    {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 8, 0, 0, 15, 17, 19, 21, 0, 23, 25,  
    27, 29, 31, 33, 35, 37,  
    39, 41, 47, 0, 0, 15, 17, 19, 21, 0, 0, 0, 0, 0, 0, 0, 0, 0, 50, 0, 0, 52, 54, 0, 0, 0, 0,  
    57, 0, 0, 0, 0, 0, 0, 0,  
    0, 0  
};
```

```
static const short actionSequences[59] =  
    {  
    krg2_a_no_action,  
/* 1*/ krg2_a_displayOn /* Off:OnOff->Measure */,  
    krg2_a_no_action,  
/* 3*/ krg2_a_ExOff /* Measure:OnOff->Off */,  
    krg2_a_HiOff,  
    krg2_a_LoOff,  
    krg2_a_displayOff,  
    krg2_a_no_action,  
/* 8*/ krg2_a_SpkrOn /* Measure:SoundPressed->SoundDwn */,  
    krg2_a_TimerStart,  
    krg2_a_FirstNote,  
    krg2_a_ExOff,  
    krg2_a_HiOff,  
    krg2_a_LoOff,  
    krg2_a_no_action,  
/* 15*/ krg2_a_CalDownPr /* Measure:CalDownPr->Measure,  
SoundUp:CalDownPr->SoundUp */,  
    krg2_a_no_action,  
/* 17*/ krg2_a_CalDownRel /* Measure:CalDownRel->Measure,  
SoundUp:CalDownRel->SoundUp */,  
    krg2_a_no_action,  
/* 19*/ krg2_a_CalUpPr /* Measure:CalUpPr->Measure,
```

```
SoundUp:CalUpPr->SoundUp */,  
    krg2_a_no_action,  
/* 21*/ krg2_a_CalUpRel /* Measure:CalUpRel->Measure,  
SoundUp:CalUpRel->SoundUp */,  
    krg2_a_no_action,  
/* 23*/ krg2_a_dev_ /* Measure:dev*->Measure */,  
    krg2_a_no_action,  
/* 25*/ krg2_a_HiOn /* Measure:HiOn->Measure */,  
    krg2_a_no_action,  
/* 27*/ krg2_a_HiOff /* Measure:HiOff->Measure */,  
    krg2_a_no_action,  
/* 29*/ krg2_a_LoOn /* Measure:LoOn->Measure */,  
    krg2_a_no_action,  
/* 31*/ krg2_a_LoOff /* Measure:LoOff->Measure */,  
    krg2_a_no_action,  
/* 33*/ krg2_a_ExOn /* Measure:ExOn->Measure */,  
    krg2_a_no_action,  
/* 35*/ krg2_a_ExOff /* Measure:ExOff->Measure */,  
    krg2_a_no_action,  
/* 37*/ krg2_a_freq_ /* Measure:freq*->Measure */,  
    krg2_a_no_action,  
/* 39*/ krg2_a_calcDev /* Measure:calc->Measure */,  
    krg2_a_no_action,  
/* 41*/ krg2_a_displayOff /* SoundUp:OnOff->Off */,  
    krg2_a_ExOff,  
    krg2_a_HiOff,  
    krg2_a_LoOff,  
    krg2_a_SpkrOff,  
    krg2_a_no_action,  
/* 47*/ krg2_a_TimerStart /* SoundUp:SoundPressed->SoundDwn */,  
    krg2_a_NextNote,  
    krg2_a_no_action,  
/* 50*/ krg2_a_calcFreq /* SoundUp:calc->SoundUp */,  
    krg2_a_no_action,  
/* 52*/ krg2_a_TimerStop /* SoundDwn:SoundReleased->SoundUp */,  
    krg2_a_no_action,  
/* 54*/ krg2_a_SpkrOff /* SoundDwn:LastNote->Measure */,  
    krg2_a_Recalc,  
    krg2_a_no_action,  
/* 57*/ krg2_a_ToggleSound /* SoundDwn:TimerTick->SoundDwn */,  
    krg2_a_no_action  
};
```

```
static float floatParameter;
```

```
static unsigned short unsignedShortParameter;
static krg2_state_id actualState, previousState;

void krg2_processActions(short actionIndex);

void krg2_boot(void)
{
    actualState = krg2_Off;
}

void krg2_processEventFloat(krg2_event_id event, float param)
{
    floatParameter = param;
    krg2_processEvent( event);
}

void krg2_processEventUnsignedShort(krg2_event_id event, unsigned short param)
{
    unsignedShortParameter = param;
    krg2_processEvent( event);
}

void krg2_processEvent(krg2_event_id event)
{
    previousState = actualState;
    actualState = krg2_nextState[previousState * krg2_numberOfEvents + event];
    krg2_processActions(krg2_actions[previousState * krg2_numberOfEvents +
event]);
}

void krg2_processActions(short actionIndex)
{while(actionSequences[actionIndex])
{switch(actionSequences[actionIndex]) {
    case krg2_a_calcDev:
        cv_calculate();
        break;
    case krg2_a_calcFreq:
        scl_calculate();
        break;
    case krg2_a_CalDownPr:
        krg1_processEvent(krg1_ev_CalDwPr);
        break;
    case krg2_a_CalDownRel:
        krg1_processEvent(krg1_ev_CalDwRl);
```

```
break;
case krg2_a_CalUpPr:
    krg1_processEvent(krg1_ev_CalUpPr);
break;
case krg2_a_CalUpRel:
    krg1_processEvent(krg1_ev_CalUpRI);
break;
case krg2_a_dev_:
    showDev(floatParameter);
break;
case krg2_a_displayOff:
    displayOff();
break;
case krg2_a_displayOn:
    displayOn();
break;
case krg2_a_ExOff:
    ledExactOff();
break;
case krg2_a_ExOn:
    ledExactOn();
break;
case krg2_a_FirstNote:
    snd_processEvent(snd_ev_first);
break;
case krg2_a_freq_:
    cv_UNDEF &= 251;
    if(cv_F != unsignedShortParameter) {
        cv_F = unsignedShortParameter;
        cv_calculate();
    }
break;
case krg2_a_HiOff:
    ledHighOff();
break;
case krg2_a_HiOn:
    ledHighOn();
break;
case krg2_a_LoOff:
    ledLowOff();
break;
case krg2_a_LoOn:
    ledLowOn();
break;
```

```
case krg2_a_NextNote:
    snd_processEvent(snd_ev_next);
break;
case krg2_a_Recalc:
break;
case krg2_a_SpkrOff:
    spkrOff();
break;
case krg2_a_SpkrOn:
    spkrOn();
break;
case krg2_a_TimerStart:
    /* start timer TI */
    TI_value = TI_count;
break;
case krg2_a_TimerStop:
    /* stop timer TI */
    TI_value = 0;
break;
case krg2_a_ToggleSound:
    snd_processEvent(snd_ev_toggle);
break;
default: /* Error Handling not yet implemented */
break;
}
actionIndex++;
}
}
```

```
#ifndef _SND_H
#define _SND_H
typedef enum
    {
        snd_ev_toggle,
        snd_ev_first,
        snd_ev_next,
        snd_numberOfEvents
    } snd_event_id ;

typedef enum
    {
        snd_chrom,
        snd_ABb,
        snd_numberOfStates
    } snd_state_id ;

void snd_boot(void);

void snd_processEvent(snd_event_id event);
#endif
```

```
#include <snd.h>
#include <scl.h>
#include <tl.h>
#include <korg.h>
#include "korgsim.h"
```

```
static const snd_state_id snd_nextState[snd_numberOfEvents *
snd_numberOfStates] =
/*      snd_ev_toggle  snd_ev_first  snd_ev_next*/{
/*snd_chrom */      snd_ABb,  snd_chrom,  snd_chrom,
/*snd_ABb */      snd_chrom,  snd_ABb,  snd_ABb};
```

```
enum {snd_a_no_action,
      snd_a_calculate,
      snd_a_firstABb,
      snd_a_fristChrom,
      snd_a_nextABb,
      snd_a_nextChrom,
      snd_a_numberOfActions};
```

```
static const short snd_actions[snd_numberOfEvents * snd_numberOfStates] =
/*      snd_ev_toggle  snd_ev_first  snd_ev_next*/{
/*snd_chrom*/      1,      4,      7,
/*snd_ABb */      4,      1,      10};
```

```
static const short actionSequences[13] =
{
  snd_a_no_action,
/* 1*/ snd_a_firstABb /* chrom:toggle->ABb, ABb:first->ABb */,
  snd_a_calculate,
  snd_a_no_action,
/* 4*/ snd_a_fristChrom /* chrom:first->chrom, ABb:toggle->chrom */,
  snd_a_calculate,
  snd_a_no_action,
/* 7*/ snd_a_nextChrom /* chrom:next->chrom */,
  snd_a_calculate,
  snd_a_no_action,
/* 10*/ snd_a_nextABb /* ABb:next->ABb */,
  snd_a_calculate,
  snd_a_no_action
};
```

```
static snd_state_id actualState, previousState;

void snd_processActions(short actionIndex);

void snd_boot(void)
{
    actualState = snd_chrom;
}

void snd_processEvent(snd_event_id event)
{
    previousState = actualState;
    actualState = snd_nextState[previousState * snd_numberOfEvents + event];
    snd_processActions(snd_actions[previousState * snd_numberOfEvents + event]);
}

void snd_processActions(short actionIndex)
{while(actionSequences[actionIndex])
{switch(actionSequences[actionIndex]) {
    case snd_a_calculate:
        scl_calculate();
        break;
    case snd_a_firstABb:
        tl_firstAb();
        break;
    case snd_a_fristChrom:
        tl_firstChrom();
        break;
    case snd_a_nextABb:
        tl_nextAb();
        break;
    case snd_a_nextChrom:
        tl_nextChrom();
        break;
    default: /* Error Handling not yet implemented */
        break;
}
    actionIndex++;
}
}
```

```
#ifndef _SCL_H  
#define _SCL_H  
void scl_calculate(void);  
#endif
```

```
#include <scl.h>
#include <korg.h>
#include "korgsim.h"
```

```
static float scl_S;
static float old_scl_S = 0;
```

```
void scl_calculate()
{
/* Calculate the real freq. for speaker */
if( !(scl_UNDEF & 5) ) {
    scl_S = scl_X * (float)scl_R / 440 ;
    /* clear undefined bit for scl_S */
    scl_UNDEF &= 5;
    /* scl_S has been calculated, process its new value */
    if(old_scl_S != scl_S) {
        old_scl_S = scl_S;
        spkFreq(scl_S);
    }
}
}
```

```
#ifndef _CV_H
#define _CV_H
void cv_calculate(void);
#endif
```

```
#include <cv.h>
#include <kr2.h>
#include <tl.h>
#include <korg.h>
#include "korgsim.h"
```

```
static float cv_N;
static float cv_D;
static unsigned char cv_L;
static unsigned char cv_E;
static unsigned char cv_H;
static float old_cv_N = 0;
static float old_cv_D = 0;
static unsigned char old_cv_L = 0;
static unsigned char old_cv_E = 0;
static unsigned char old_cv_H = 0;
```

```
void cv_calculate()
{
/* Normalize to 440 Hz */
if( !(cv_UNDEF & 68)) {
cv_N = (float)cv_F * 440 / (float)cv_R ;
/* clear undefined bit for cv_N */
cv_UNDEF &= 223;
/* cv_N has been calculated, process its new value */
if(old_cv_N != cv_N) {
old_cv_N = cv_N;
tl_greaterEqualLimit(cv_N);
}
}
/* deviation (Cent) */
if( !(cv_UNDEF & 160)) {
cv_D = ( cv_N - cv_X ) * 100 / ( 0.05946 * cv_X ) ;
/* clear undefined bit for cv_D */
cv_UNDEF &= 254;
/* cv_D has been calculated, process its new value */
if(old_cv_D != cv_D) {
old_cv_D = cv_D;
kr2_processEventFloat(kr2_ev_dev_, cv_D);
}
}
}
```

```
/* low ind. when more the 5 Cent too low */
if( !(cv_UNDEF & 1)) {
    cv_L = cv_D < -5 ;
    /* clear undefined bit for cv_L */
    cv_UNDEF &= 239;
    /* cv_L has been calculated, process its new value */
    if(old_cv_L != cv_L) {
        old_cv_L = cv_L;
        if(cv_L) {
            krg2_processEvent(krg2_ev_LoOn);
        } else {
            krg2_processEvent(krg2_ev_LoOff);
        }
    }
}
}
}
/* exact indicator: +/- 8 Cent */
if( !(cv_UNDEF & 1)) {
    cv_E = ( cv_D > -8 ) && ( cv_D < 8 ) ;
    /* clear undefined bit for cv_E */
    cv_UNDEF &= 253;
    /* cv_E has been calculated, process its new value */
    if(old_cv_E != cv_E) {
        old_cv_E = cv_E;
        if(cv_E) {
            krg2_processEvent(krg2_ev_ExOn);
        } else {
            krg2_processEvent(krg2_ev_ExOff);
        }
    }
}
}
}
/* hi ind. when more the 5 Cent too high */
if( !(cv_UNDEF & 1)) {
    cv_H = cv_D > 5 ;
    /* clear undefined bit for cv_H */
    cv_UNDEF &= 247;
    /* cv_H has been calculated, process its new value */
    if(old_cv_H != cv_H) {
        old_cv_H = cv_H;
        if(cv_H) {
            krg2_processEvent(krg2_ev_HiOn);
        } else {
            krg2_processEvent(krg2_ev_HiOff);
        }
    }
}
}
```

}
}

```
#ifndef _TL_H
#define _TL_H
void tl_firstAb(void);
void tl_firstChrom(void);
void tl_greaterEqualLimit(float param);
void tl_nextAb(void);
void tl_nextChrom(void);
#endif
```

```
#include <tl.h>
#include <cv.h>
#include <kr2.h>
#include <scl.h>
#include <korg.h>
#include <string.h>
#include "korgsim.h"
```

```
typedef void funcArr(void);
void shfl_flat(void);
void shfl_none(void);
void shfl_sharp(void);
static float freqData[37] =
    {220, 233.081881, 246.941651, 261.625565, 277.182631,
    293.664768, 311.126984, 329.627557, 349.228231, 369.994423,
    391.995436, 415.304698, 440, 466.163762, 493.883301,
    523.251131, 554.365262, 587.329536, 622.253967, 659.255114,
    698.456463, 739.988845, 783.990872, 830.609395, 880,
    932.327523, 987.766603, 1046.50226, 1108.73052, 1174.65907,
    1244.50794, 1318.51023, 1396.91293, 1479.97769, 1567.98174,
    1661.21879, 1760};
static float limitData[37] =
    {226.54094, 240.011766, 254.283608, 269.404098, 285.423699,
    302.395876, 320.37727, 339.427894, 359.611327, 380.994929,
    403.650067, 427.652349, 453.081881, 480.023531, 508.567216,
    538.808196, 570.847399, 604.791752, 640.754541, 678.855788,
    719.222654, 761.989859, 807.300134, 855.304698, 906.163762,
    960.047063, 1017.13443, 1077.61639, 1141.6948, 1209.5835,
    1281.50908, 1357.71158, 1438.44531, 1523.97972, 1614.60027,
    1710.6094, 1812.32752};
static char noteChar[] =
    "ABBCCDEEFFGGABBCCDDEFFGGABBCCDDEFFGGA";
static unsigned char notePos[38] =
    {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
    16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
    29, 30, 31, 32, 33, 34, 35, 36, 37};
funcArr * shfl_fnc[37] =
    {&shfl_none, &shfl_flat, &shfl_none, &shfl_none, &shfl_sharp, &shfl_none,
    &shfl_flat,
    &shfl_none, &shfl_none, &shfl_sharp, &shfl_none, &shfl_sharp, &shfl_none,
    &shfl_flat,
    &shfl_none, &shfl_none, &shfl_sharp, &shfl_none, &shfl_sharp, &shfl_none,
```

```
&shfl_none,
    &shfl_sharp, &shfl_none, &shfl_sharp, &shfl_none, &shfl_flat, &shfl_none,
&shfl_none,
    &shfl_sharp, &shfl_none, &shfl_sharp, &shfl_none, &shfl_none, &shfl_sharp,
    &shfl_none, &shfl_sharp, &shfl_none};
static unsigned char chromIndex[] =
    {3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
static unsigned char ablIndex[] =
    {12, 13};
```

```
static unsigned char cursorIndex;
static char * strvar[2];
```

```
void processIndex(unsigned char index);
```

```
void processIndex(unsigned char index)
{
    cv_UNDEF &= 127;
    if(cv_X != freqData[index]) {
        cv_X = freqData[index];
        cv_calculate();
    }
    scl_UNDEF &= 3;
    scl_X = freqData[index];
    strncpy(strvar, &noteChar[notePos[index]], notePos[index + 1] - notePos[index]);
    showNote(strvar);
    (shfl_fnc)[index]();
}
```

```
void shfl_flat(void)
{
    showFlat();
}
```

```
void shfl_none(void)
{
    showNone();
}
```

```
void shfl_sharp(void)
```

```
{
  showSharp();
}

void tl_firstAb(void)
{
  cursorIndex = 0;
  processIndex(abIndex[cursorIndex]);
}

void tl_firstChrom(void)
{
  cursorIndex = 0;
  processIndex(chromIndex[cursorIndex]);
}

void tl_greaterEqualLimit(float param)
{
  unsigned char ix;
  for(ix = 0; ix < 37; ix++) {
    if(limitData[ix] >= param) {
      processIndex(ix);
      return;
    }
  }
}

void tl_nextAb(void)
{
  if(++cursorIndex >=2){
    krg2_processEvent(krg2_ev_LastNote);
    return;
  }
  processIndex(abIndex[cursorIndex]);
}

void tl_nextChrom(void)
{
  if(++cursorIndex >=13){
    krg2_processEvent(krg2_ev_LastNote);
    return;
  }
  processIndex(chromIndex[cursorIndex]);
}
```

}