X-GUI 手册

目 录

修订记录

版本号	日 期	修订人	备 注

《 X-GUI 手册 》For【 iBoard 电子学堂】

版 本：V 1.0 日 期：07/29/2012

深入交流 QQ 群：

A: 204255896（500 人超级群，满员）
B: 165201798（500 人超级群，满员）
C: 215053598（200 人高级群）
D: 215054675（200 人高级群）
E: 215055211（200 人高级群）
F: 78538605 （500 人高级群）
G:158560047（500 人高级群）
YY 群 ：7182393
YY 频道 ：80518139（不定期语音群课）
论坛：http://www.heijin.org
博客：http://XiaomaGee.cnblogs.com

提示：请关注论坛和博客，以便浏览本文档最新版本

目 录

目 录

第一章 EVTFT 构架

EVTFT 构架

本章介绍了 EVTVT 的构架。

本章分为以下几个部分：

一、EVTFT 概述
二、寄存器描述
三、硬件连接形式
四、背光电路
五、EVTFT 驱动解析

一、 EVTFT

1、 EVTFT 概述

EVTFT 是一种优秀的真彩液晶显示构架，可广泛用于嵌入式仪器仪表、工业现场、智能家居等领域。通过内建驱动器及高速 Intel 8080 接口，使它具有使用简单、快速、显示效果好等诸多优点。通过应用本系统，可以使您的产品档次骤然提高，并能有效的缩短开发时间，提高产品的竞争力。

2、 EVTFT 构架特性

- 4.3 英寸 16:9 宽屏，480×272 分辨率；
- 65536 颜色显示，色彩逼真；
- 内建驱动器，16 位 8080 高速接口；
- 高亮白光 LED 背光，亮度 PWM 可调；
- X、Y 光标自增，方便高速读写；
- 可方便与 51 / DSP / ARM / FPGA 连接；
- μcGUI / ZLGGUI 等诸多软件库支持。

3、 EVTFT 构架框图

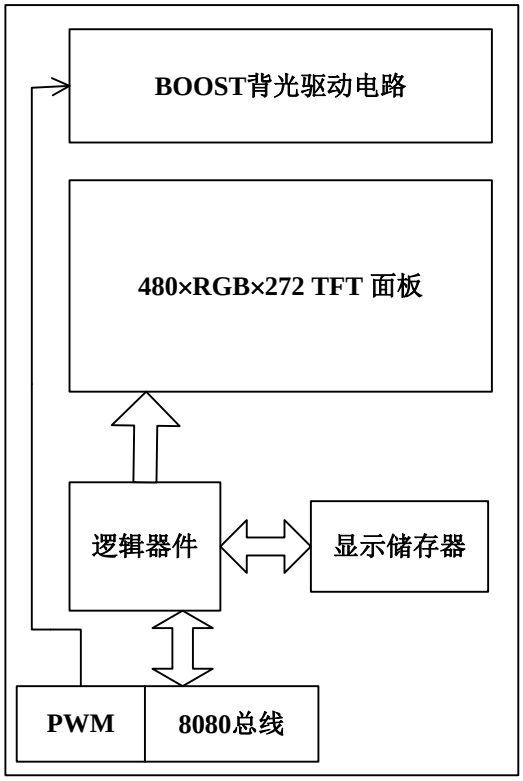

图 1-1 EVTFT 框图

4、 EVTFT 应用领域

- 高级仪器仪表；
- 工业现场；
- 远程控制；
- 楼宇自动化、门禁系统；
- 高端人机界面。

二、 寄存器描述

EVTFT 支持 16 位数据传输，其寄存器宽度均为 16 位。如表 1-1 所示。

表 1-1 EVTFT 的寄存器定义

地址	名称	默认值	属性 ※
0	DATA[15:0]	0X0000	R/W
2	CSRX[9:0]	0X000	WO
4	CSRY[9:0]	0X000	WO
6	CTL[15:0]	0X0000	WO

※ R/W 为可读写，WO 为只写

地址 0 为数据寄存器，传递当前光标下的液晶点 RGB 值，模式为 RGB 565 即 16 位色模式。映射关系如下：

DATA[15:11] = R[4:0]

DATA[10:5] = G[5:0]

DATA[4:0] = B[4:0]

地址 2 为光标行地址(X 地址)，范围为 0~479。地址 4 为光标列地址(Y 地址)，范围为 0~271。光标地址写入 CSRY 后生效。写入 CSRX 后不会立即生效。液晶坐标系如图 1-2 所示。

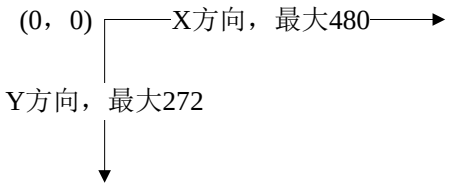

图 1-2 液晶坐标系

地址 6 为控制寄存器(CTL)，功能如图 1-3 所示：

位地址	CTL [15]	CTL [14:7]	CTL [6]	CTL [5:1]	CTL [0]
名称	LED	RESERVE	CSRMODE	TMODE	DISPEN

LED控制:
0 - LED亮
1 - LED灭

保留，不要向保留寄存器写1

地址自增模式:※
0 - 沿X轴增长
1 - 沿Y轴增长

显示开关:
0 - 显示关
1 - 显示开

测试模式 ：
0000 - 正常模式
其他 - 显示红色

图 1-3　控制寄存器功能图

注意：※沿 X 轴增长溢出时，会自动换行；沿 Y 轴溢出时，将发生错误。请确保不要发生沿 Y 轴溢出的情况

RGB 数据写入后，地址会递增，递增模式取决于 CTL 的 CSMODE 位。CSMODE=0 时，地址沿 X 轴增长；CSMODE=1 时，地址沿 Y 轴增长。

三、　硬件连接形式

1.　iHMI43 硬件连接

iHMI 的硬件连接电路如图 1-4 所示，STM32 与 EVTFT 之间通过总线 FSMC 相连。其中，总线由数据线和地址线组成，数据线由 16 根 DB[15：0]线组成，地址线为 2 根 AB[1：0]线。片选 CS 与 EVTFT 相连。

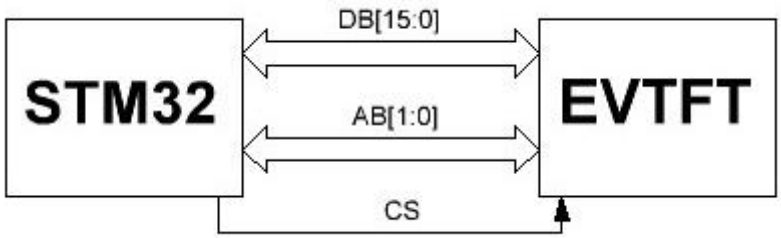

图 1-4 iHMI43 硬件连接图

由 STM32 芯片的官方手册可知，STM32 系列 100 脚封装芯片的 FSMC 总线静态存储器为 Bank1， 其物理寻址空间是从 0x6000 0000 到 0x6FFF FFFF， 所以

iHMI43 的 EVTFT 的寄存器对应的物理寻址空间分别是 0x6000 0000、0x6002 0000、0x6004 0000 和 0x6006 0000。iHMI43 总线的地址线对应的物理寻址空间如代码 1-1 所示：

代码 1-1　iHMI43 的寻址空间

```
29 #define LCD_DATA        *((volatile unsigned short int*)(0x60000000))
30 #define LCD_ADDX        *((volatile unsigned short int*)(0x60020000))
31 #define LCD_ADDY        *((volatile unsigned short int*)(0x60040000))
32 #define LCD_CTL         *((volatile unsigned short int*)(0x60060000))
```

2. iBoard 硬件连接

《iBoard 电子学堂》硬件连接电路如图 1-5 所示，与 iHMI 电路不同，其总线地址线为 AB[17：16]，片选 CS 经过 Decoder 译码成 CS0 和 CS1 两个片选空间，CS0 与 EVTFT 相连。

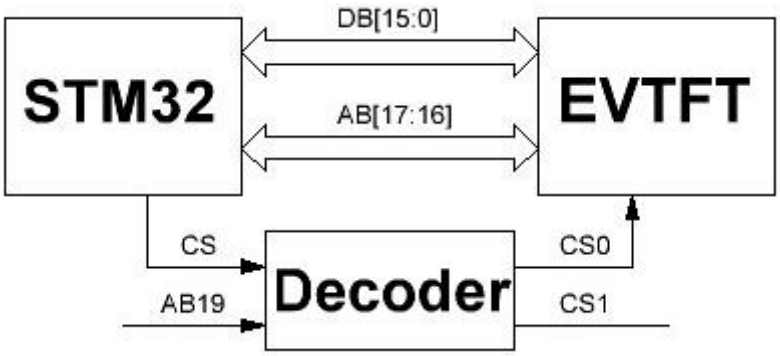

图 1-5 iBoard 硬件连接图

由上小节可知，STM32 的 Bank1 的物理寻址空间是从 0x6000 0000 到 0x6FFF FFFF，但是由于片选 CS 经过译码器，所以它对应的物理寻址空间从 0x6010 0000 开始。所以 iBoard 的 EVTFT 寄存器对应的物理寻址空间分别为 0x6010 0000、0x6012 0000、0x6014 0000 和 0x6016 0000。iBoard 总线的地址线对应的物理寻址空间见如代码 1-2 所示：

代码 1-2　iBoard 液晶寻址空间

```
29 #define LCD_DATA        *((volatile unsigned short int*)(0x60100000))
30 #define LCD_ADDX        *((volatile unsigned short int*)(0x60120000))
31 #define LCD_ADDY        *((volatile unsigned short int*)(0x60140000))
32 #define LCD_CTL         *((volatile unsigned short int*)(0x60160000))
```

四、 背光电路

背光系统包含 8 只串联的高亮白光 LED，由 BOOST 升压电路驱动，驱动模式为横流模式。

BOOST 升压电路由 STM32 输出的 PWM 控制，如图 1-6 所示。PWM 引脚为高电平时，开启背光系统；PWM 引脚为低电平时，关断背光系统。

通过调节不同的 PWM 占空比，来控制 8 只 LED 亮度。

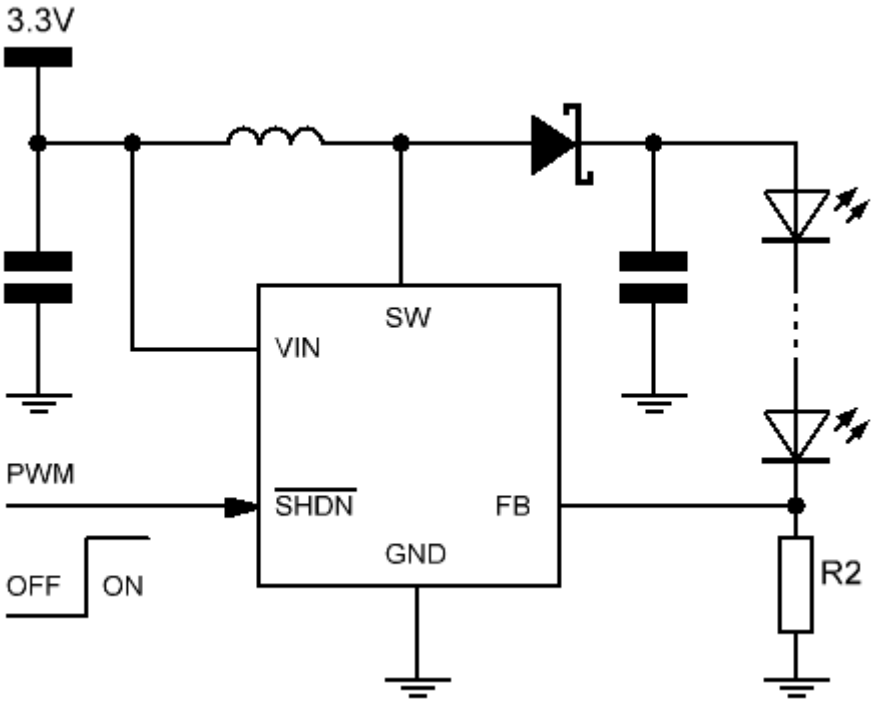

图 1-6 boost 电路

五、 EVTFT 驱动解析

代码 1-3 为 EVTFT 的驱动代码，详细工程见 include\evtft.h。

代码 1-3 EVTFT 头文件定义

```
//---------------- Define ----------------------------//

#define LCD_XSIZE       480         //定义液晶屏的长度
#define LCD_YSIZE       272         //定义液晶屏的宽度

#define LCD_DATA        *((volatile unsigned short int*)(0x60100000))
#define LCD_ADDX        *((volatile unsigned short int*)(0x60120000))
```

```
31 #define LCD_ADDY        *((volatile unsigned short int*)(0x60140000))
32 #define LCD_CTL         *((volatile unsigned short int*)(0x60160000))
33
34 #define DISPLAY_ON      1                   //开始显示
35 #define DISPLAY_OFF     0                   //停止显示
36
37 #define CSRMODE_H       0                   //地址沿水平 x 方向增长
38 #define CSRMODE_V       1                   //地址沿竖直 y 方向增长
39
40 #define lcd_data(x)     LCD_DATA = x        //定义数据寄存器
41 #define lcd_csrx(x)     LCD_ADDX = x        //定义光标行地址
42 #define lcd_csry(x)     LCD_ADDY = x        //定义光标列地址
43 #define lcd_ctl(x)      LCD_CTL = x         //定义控制寄存器
44
45
46 //----------------- Typedef-----------------------------//
47
48 typedef struct {
49         int (* initialize)(void);           //液晶初始化函数
50         int (* set_point)(int /* x */, int /* y */, unsigned short int
/* color */);                                  //设置点
51         unsigned short int (* get_point)(int /* x */, int /* y */);
                                               //获取点
52         int (* cls)(COLOR_T);               //清屏
53         void (* change)(void);              //双缓冲
54         int (* write_a)(void);              //写显存 a
55         int (* write_b)(void);              //写显存 b
56 }LCD_T;
57
58 typedef union {
59         struct {
60                 unsigned short int dispen : 1;  //显示
61                 unsigned short int tmode : 5;   //测试模式
62                 unsigned short int csrmode : 1; //地址沿 y 方向自增模式
63                 unsigned short int write_buf : 1;// 写显存
64                 unsigned short int disp_buf : 1; // 读显存
65                 unsigned short int reserve1 : 6; //保留
66                 unsigned short int LED : 1;      //LED 灯亮
67         }b;
68         unsigned short int w;
69 }LCD_CTL_T;
70
```

第 48~56 句定义了 LCD_T 结构体，第 52 句是清屏，可以将屏清成任一种颜色。比如“lcd.cls(COLOR_WINDOW_BACKGROUND);”这句代码能将屏清成系统背景色。第 53 句是双缓冲。第 54、55 句分别是往逻辑 a、b 里写数据。

双缓冲过程的流程图如图 1-7 所示。需要显示的信息首先被写入显存里，然后显存里的数据读给 EVTFT 显示。其中，显存被分为两部分，即逻辑 a 和逻辑 b。当 write_buf=0 时，disp_buf=0，STM32 的数据写入逻辑 a，逻辑 b 的数据读给 EVTFT；当 write_buf=1 时，disp_buf=1，STM32 的写入数据逻辑 b，逻辑 a 的数据读给 EVTFT。双缓冲可以使得在写入数据时，EVTFT 里的原有信息不被立即擦除，由此不会发生闪屏的情况。

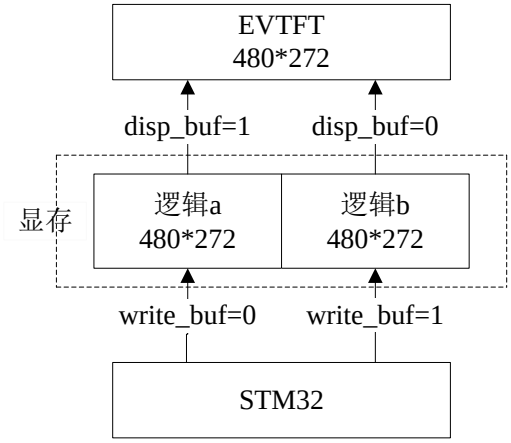

图 1-7　双缓冲

第 58~69 句定义了一个联合体，包括 b 结构体和 w 变量。b 和 w 映射到同一地址空间，但是 b 是按位域进行访问的，w 是按字进行访问的。w 是 unsigned short int 类型，占 16 个位。

第 60 句“unsigned short int dispen :1”中 dispen 后面的 1 表示位域，即 dispen 占 1 位。第 61 句是测试模式，当其值为 0000 时为正常模式，否则显示红色，tmode 占 5 位。第 64 句是显示开关信号，当其为 0 时，显示开关打开，当其为 1 时，显示开关关闭，disp_buf 占 1 位。第 5 句是保留，注意不要向保留寄存器写 1，该寄存器占 1 位。

第二章　颜色模型

颜色模型介绍

本章介绍了 GUI 中各种颜色模型。

本章分为以下几个部分：

一、RGB 简介
二、RGB 颜色模型
三、RGB 程序示例

一、 RGB 简介

颜色模型，又可以叫做颜色空间，是人们解析现实中颜色的方法。常用的颜色模型有 RGB、CMYK、HSB、YUV、Lab 等，我们的 EVTFT 液晶模块就是由密密麻麻的 RGB 色块构成的。

顾名思义，RGB 即 RED、GREEN、BLUE 三种颜色通道的缩写。RGB 颜色模型采用加法混色法，加法混色法的意思就是各种 R、G、B 三种色光，通过不同分量的叠加，来形成五彩缤纷的颜色。如图 2-1 所示。

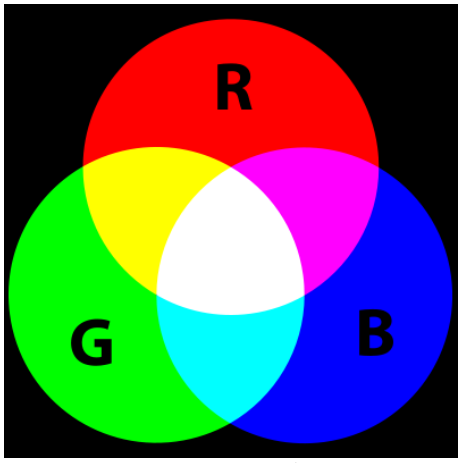

图 2-1 RGB 颜色空间

大家都知道，计算机使用离散的数字信号来描述数据，RGB 模型也不例外。每种颜色通道的离散点越多，对颜色的描述越精细，不过这也会带来庞大的数据量；实际应用过程中，在色彩数量与数据量之间做了个权衡，这样就出现了几种固定的 RGB 颜色模型格式，他们分别为 RGB444、RGB555、RGB565、RGB18、RGB24、RGB32 等。

二、 RGB 颜色模型

1、 RGB444 颜色模型

RGB444 多用于嵌入式系统的显示器上，R、G、B 三种颜色通道分别用四位表示，总共 12 位，可以产生 4096 个颜色。图 2-2 为 RGB444 色阶图，从图中可以看到 16 个色阶，每个色阶中颜色的跳变都是不连续的，这就是数字量化的结果。

图 2-2　RGB444 的色阶

2、 RGB555 颜色模型

RGB555 中 R、G、B 的三个颜色通道分别用五位表示，可以产生 32768 色，即 32k 色。计算机中，使用两个字节表示这种颜色格式，表示方法如图 2-3，最高位 X 表示没使用。

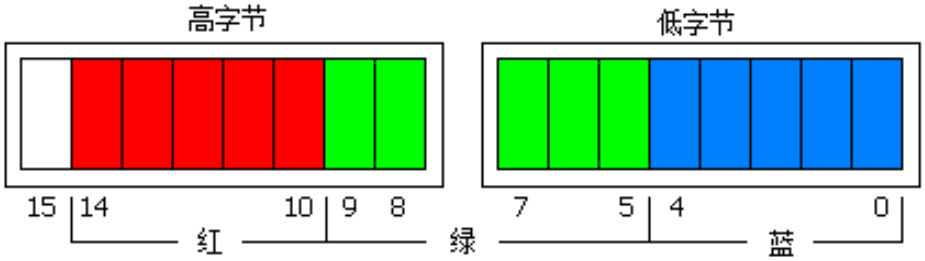

图 2-3　RGB555 的存储格式

图 2-4　RGB555 的色阶

图 2-4 为 RGB555 色阶图，与图 2-1 相比可以看出，RGB555 的细化程度，比 RGB444 好了点，但是人的眼睛还是能分辨出阶梯状。

3、 RGB565 颜色模型

RGB565 与 RGB555 相似，唯一区别的就是它使用六位表示 G 颜色通道，这样可以达到 65536 色，即 64k 色或 16bit 色。其存储格式如图 2-5 所示，RGB565 所占用的两个字节中，R 分量存于高五位，G 分量存于中六位，B 分量存于低五位。如此，相同的字节可以产生更多的颜色。EVTFT 系列液晶模块、VGA 显示驱动器也是采用这种存储格式。

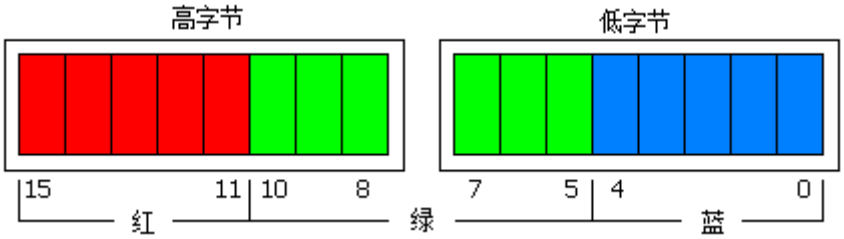

图 2-5　RGB555 的存储格式

图 2-6　RGB565 的色阶

图 2-6 为 RGB565 的色阶，从图中可以看出来，中间的 G 色阶，比两边的 R、B 细腻一点，这就是用六位表示 G 颜色通道而只用 5 为表示其他两位颜色通道的结果。

根据 RGB565 的存储特征，在 C 语言中，我们可以用下面的宏定义，来从 RGB565 数据中，分别获取 R、G、B 的值。

代码 2-1　获取 RGB 分量

```
#define get_rgb565_b_value(rgb) ((rgb & 0x1f) << 3)
#define get_rgb565_g_value(rgb) (((rgb >> 5) & 0x3f) << 2)
#define get_rgb565_r_value(rgb) (((rgb >> 11) & 0x1f) << 3)
```

同样的道理，也可以通过下面的宏定义，由 R、G、B 的值合成 RGB565 数据。

代码 2-2　合成 RGB565 数据

```
#define make_rgb565(r, g, b)   ((r >> 3) << 11 | (g >> 2) << 5 | (b >> 3))
```

常用 RGB565 颜色定义如代码 2-3 所示。

代码 2-3　常用 RGB565 颜色及各自的数值

```
#define COLOR_BLACK          0x0000
#define COLOR_NAVY           0x0010
#define COLOR_GREEN          0x0400
#define COLOR_TEAL           0x0410
#define COLOR_MAROON         0x8000
#define COLOR_PURPLE         0x8010
```

```
#define COLOR_OLIVE                 0x8400
#define COLOR_GRAY                  0x8410
#define COLOR_SILVER                0xC618
#define COLOR_BLUE                  0x001F
#define COLOR_LIME                  0x07E0
#define COLOR_AQUA                  0x07FF
#define COLOR_RED                   0xF800
#define COLOR_FUCHSIA               0xF81F
#define COLOR_YELLOW                0xFFE0
#define COLOR_WHITE                 0xFFFF
#define COLOR_ORANGE                0xfc00
#define COLOR_DARK_GREEN            0x2a40
#define COLOR_DARK_GRAY             0x4208
#define COLOR_DARK_BLUE             0x0018
#define COLOR_WINDOW_BACKGROUND     0xce79
#define COLOR_WINDOW_SHADOW         0x8430
#define COLOR_DARK                  0X39E7
#define COLOR_SYSTEM                0X9680
```

4、 RGB24 和 RGB32 颜色模型

RGB24 使用 24 位即三个字节表示 RGB 颜色通道，每个颜色通道各占一个字节即各有 256 个色阶，计算机上大部分都是按这个来表示的。256 级的变化，人的肉眼已经分辨不出来色阶的跳变了。

RGB32 也是使用 24 位即三个字节表示 RGB 颜色通道，R、G、B 各占用一个字节，只不过它用最后一个字节表示 Alpha 通道，这就是所谓的真彩 32 位色。通俗的讲，Alpha 就是表示像素透明度的，0 表示全透明，255 表示不透明，在此之间为透明度的线性变化。

三、RGB 程序示例

```
int main(void)
{
        int i, j, k;
        STRING_T s;
        RECT_T r;

        const int color_tab[] = {
                COLOR_BLACK,
                COLOR_NAVY,
```

```
106                 COLOR_GREEN,
107                 COLOR_TEAL,
108                 COLOR_MAROON,
109                 COLOR_PURPLE,
110                 COLOR_OLIVE,
111                 COLOR_GRAY,
112                 COLOR_SILVER,
113                 COLOR_BLUE,
114                 COLOR_LIME,
115                 COLOR_AQUA,
116                 COLOR_RED,
117                 COLOR_FUCHSIA,
118                 COLOR_YELLOW,
119                 COLOR_WHITE,
120                 COLOR_ORANGE,
121                 COLOR_DARK_GREEN,
122                 COLOR_DARK_GRAY,
123                 COLOR_DARK_BLUE,
124                 COLOR_DARK,
125                 COLOR_SYSTEM,
126                 COLOR_WINDOW_SHADOW
127         };
128         unsigned char * color_text[] = {
129                 "BLACK", "NAVY", "GREEN", "TEAL", "MAROON",
130                 "PURPLE", "OLIVE", "RAY", "SILVER", "BLUE",
131                 "LIME", "AQUA", "RED", "FUCHSIA", "YELLOW",
132                 "WHITE", "ORANGE", "DARK_GREEN", "DARK_GRAY",
133                 "DARK_BLUE", "DARK", "SYSTEM", "WINDOW_SHADOW"
134         };
135
136         initialize();
137
138         font._default.single_byte = &fixedsys;
139
140         k = 0;
141
142         for (i = 0; i < 5; i++) {
143                 for (j = 0; j < 5; j++) {
144                         r.x = 30 + 90 * j;
145                         r.y = 20 + 50 * i;
146                         r.width = 60;
147                         r.height = 20;
148                         r.fill_flag = 1;
149                         r.color = color_tab[k];
```

```
150                         gui.rect(&r);
151
152                         s.x = 30 + 90 * j;
153                         s.y = 45 + 50 * i;
154                         s.space.line = 0;
155                         s.space.word = 0;
156                         s.color = COLOR_BLACK;
157                         s.background_color = COLOR_WINDOW_BACKGROUND;
158                         s.inverse = NULL;
159                         font.printf(&s, "%s", color_text[k]);
160
161                         k++;
162                         if (k >= 23) break;
163                     }
164             }
165
166             while (1) ;
167 }
168
```

程序中，数组 color_tab 为颜色常量，数组 color_text 存放与 color_tab 相对应的颜色名称，第 136 句是调用初始化函数，第 138 句是将默认显示字体类型设为 fixedsys，第 144~150 句是画 60*20 的矩形，并且分别由数组 color_tab 内的颜色填充，第 152~159 句程序代码的作用是将名称放在相应的色块的下方。

这个程序是 RGB 的示例，它的作用是显示包括背景色在内的 24 种颜色，色块排列共 5 行，前 4 行每行显示 5 种颜色，最后一行显示 3 种颜色。详细工程在 chapter_2\color 中，程序运行结果如图 2-7 所示。

图 2-7　color

第三章　字体驱动

字体驱动

本章介绍了GUI 中的各种字体，以及字体的驱动。

本章分为以下几个部分：

一、字体构架
二、中文字体
三、等宽英文字体
四、比例英文字体

一、 字体构架

1、 驱动架构

X –GUI 字体驱动架构流程图如图 3-1 所示。

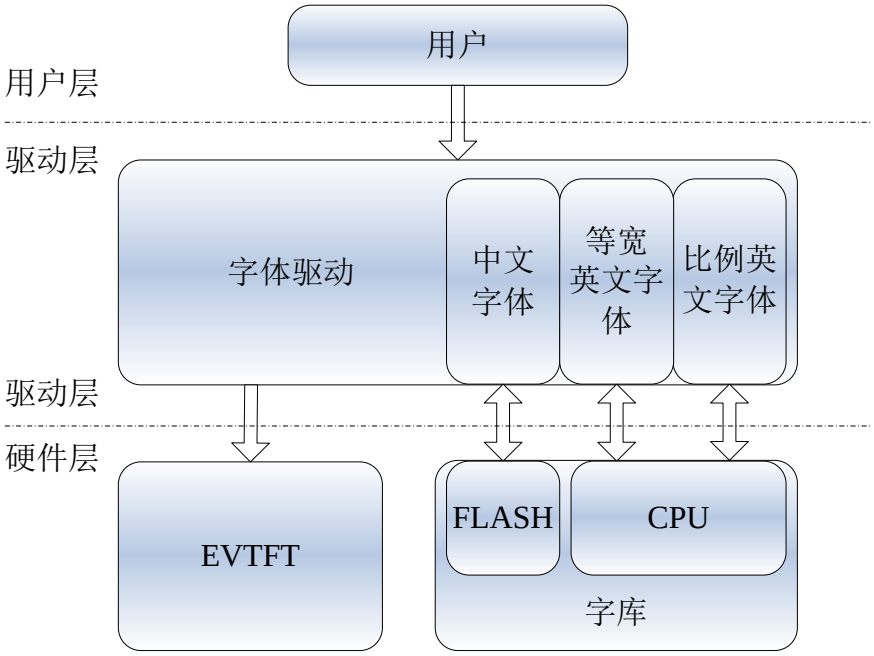

图 3-1 X-GUI 字体驱动架构

用户输入语句后，语句以编码的形式进入驱动层。在驱动层中，编码都被存在字符串内，由逻辑字体对需要显示的字体进行分类，并驱动字体的字库，从硬件层的字库中获取其点阵信息。在硬件层中，点阵信息及像素信息被存入 EVTFT 的寄存器中，之后就可以显示字体了。字库不仅可以存储在 FLASH、CPU 内置存储器中，还可以存储在 SD 卡中。

2、 字体类型

X-GUI 支持的字体都是点阵字体。点阵字体，就是把每一个字符都分成 16×16 或 24×24 个点 ,用点的虚实来表示字符的轮廓。点阵信息也就是一组二维像素信息。该字体可分为中文字体和英文字体，也可以分为比例字体和等宽字体。

等宽字体（monospaced font）是指字元宽度相同的电脑字体，比例字体（proportional font）是字元宽度不尽相同的电脑字体。等宽与比例字体的区别是等宽字体的宽度不为 0，而比例字体的宽度被定为 0，但是事实上是对比例字体进行了比例调整，它的宽度是未知的。

图 3-2　等宽点阵字体与比例点阵字体

图 3-2 中字体均为 8 号，前 26 个英文字母为等宽字体中的 courier 字体，后 26 个为比例字体中的 tahoma 字体，相比之下可以看出在等宽字体中，字母 i、j 显得两侧馀白较多，而字母 w、m 等的笔画显得相当拥挤，而比例字体的字符排列较紧凑但不拥挤。

X-GUI 支持的字体类型的详细信息如表 3-1 所示。

表 3-1　字体详细信息

序号	类型	驱动方向	字高/byte	字宽/byte	字库驱动的位置
1	simhei24	H	3	3	firmware\ font\ simhei24.c
2	simsun12	H	2	1.5	firmware\ font\ simsun12.c
3	simsun16	H	2	2	firmware\ font\ simsun16.c
4	simsun16f	H	2	2	firmware\ font\ simsun16f.c
5	courier10	V	2	1	firmware\ font\ courier10.c
6	fixedsys	V	2	1	firmware\ font\ fixedsys.c
7	monaco	V	2	1	firmware\ font\ monaco.c
8	borlandTE	V	2	1	firmware\ font\ borlandTE.c
9	tahoma8	V	2	0	firmware\ font\ tahoma8.c
10	tahoma9	V	2	0	firmware\ font\ tahoma9.c
11	tahoma10	V	2	0	firmware\font\ tahoma10.c
12	tahoma11	V	2	0	firmware\ font\ tahoma11.c
13	tahoma12	V	2	0	firmware\ font\ tahoma12.c
14	tahoma26	V	5	0	firmware\ font\ tahoma26.c

表 3-1 中，1~4 号字体是中文字体，同时也是等宽字体；5~8 号字体是等宽英文字体；9~14 号字体是比例英文字体。方向是字体的显示方向，它取决于字库的存储方式是横向矩阵还是纵向矩阵。

3、 数据结构

X-GUI 字体驱动所用到的结构体都在文件 firmware\include\font.h 中，它们几乎包含了与字体显示有关的所有信息。具体内容分别如代码 3-1、3-2· · · · · ·3-7 所示。

代码 3-1　字体点阵结构体

```
42 typedef struct _FONT_MATRIX {
43         unsigned char character[2];    //字体编码数组
44         int direction;                 //点阵显示方向
45         int height;                    //点阵高度
46         int width;                     //点阵宽度
47         char const * dat;              //点阵信息在字库中的首地址
48  }FONT_MATRIX_T;                       //定义字体点阵类型变量
49
```

字体点阵结构体包含字体编码和点阵信息。点阵信息由驱动程序内的 get 函数获取。第 43 句中编码数组存放字体的字节编码。英文字体的编码用 ASCII 码表示，仅有一个字节；中文字体的编码又称为内码，有两个字节。利用规定的计算公式，可以根据字体编码获得字体点阵信息在字库中的首地址。

代码 3-2、3-3 是分别关于单字节字体、双字节字体的结构体。单字节字体也就是英文字体，双字节字体也就是中文字体。

代码 3-2　单字节字体结构体

```
50 typedef struct  _SINGLE_BYTE_FONT {
51         char * name;            //单字节字体类型的名称
52         unsigned long int id;   //字体在单字节字体链表内的偏移量
53         int height;
54         int width;              //width!=0 is monospaced font,
                                   width==0 is proportional font
55         char first_char;
56         int (*get_matrix)(FONT_MATRIX_T *); //字体点阵是点阵结构体类型
57         void * dat;                         //点阵信息在字库中的首地址
58         struct _SINGLE_BYTE_FONT * pre; //前一个字体是单字节结构体类型
59         struct _SINGLE_BYTE_FONT* next; //后一个字体是单字节结构体类型
60 }SINGLE_BYTE_FONT_T;                     //定义单字节字体类型变量
61
```

第 52 句中偏移量 id 的作用是定位字体在单字节链表的点阵信息。

代码 3-3　双字节字体结构体

```
64 typedef struct _DOUBLE_BYTE_FONT {
65         char * name;                    //双字节字体类型的名称
66         unsigned long int id;
67         int height;
68         int width;
69         int (*get_matrix)(FONT_MATRIX_T *);
70         void * dat;
71         struct _DOUBLE_BYTE_FONT * pre;//前一个字体是单字节结构体类型
72         struct _DOUBLE_BYTE_FONT * next;
73  }DOUBLE_BYTE_FONT_T;                   //定义双字节字体类型变量
74
```

第 66 句中偏移量 id 的作用是定位字体在双字节链表的点阵信息。

代码 3-4　逻辑字体结构体

```
77 typedef struct _LOGIC_FONT {
78      char * name;
79      int height;                        //逻辑字体高度
80      SINGLE_BYTE_FONT_T * single_byte;//指定* single_byte 是单字节字体
81      DOUBLE_BYTE_FONT_T * double_byte;//指定* double_byte 是双字节字体
82   }LOGIC_FONT_T;
83
```

逻辑字体是字体的描述，它是抽象的物件。逻辑字体的作用是支持某几种字体类型，分析字体的模块和多字节字符串。也就是说，用户输入的多个字体、字符将由逻辑字体进行筛选分类，之后才驱动所需类型的字库。

代码 3-5　反白链表结构体

```
85 typedef struct _INVERSE {
86         int start;                      //反白的起始地址
87         int end;                        //反白的终止地址
88         COLOR_T color;                  //字体颜色
89         COLOR_T background_color;       //字体的背景颜色
90         struct _INVERSE * next;         //后一个字体是反白结构体类型
91   }INVERSE_T;                           //定义反白类型变量
92
```

反白是改变字符像素值，达到突出、强调的目的，它的显示效果与 Microsoft office 中选中某段句子后的现象相似。需要反白的字体会被加入 inverse 链表内。

代码 3-6 字符串联合体

```
 95 typedef struct {
 96         int x;                          //lcd 光标 x 的地址
 97         int y;                          //lcd 光标 y 的地址
 98         int color;
 99         int background_color;
100         int effect;
101         INVERSE_T * inverse;            //* inverse 是反白链表的内容
102         struct {
103                 int word;               //字间距
104                 int line;               //行间距
105         }space;                         //间距类型变量
106 }STRING_T;                              //定义字符串类型变量
107
```

字符串内存放所有需要显示的字体，以及换行符“\n”、空格、空行等。第 96、97 行的光标地址也就是字体在液晶屏上的坐标（x，y）。第 100 行 effect 的值如果是 1，就代表需要反白，否则，就正常显示。

代码 3-7 字体结构体

```
110 typedef struct {
111         LOGIC_FONT_T _default;      //默认字体类型是逻辑字体
112         int (* initialize)(void);
113         int (* register_single_byte_font)(SINGLE_BYTE_FONT_T *);
114                                     //该单字节字体在单字节链表中
115         int (* register_double_byte_font)(DOUBLE_BYTE_FONT_T *);
116         int (* unregister_single_byte_font)(SINGLE_BYTE_FONT_T *);
117                                     //该单字节字体在不在单字节链表中
118         int (* unregister_double_byte_font)(DOUBLE_BYTE_FONT_T *);
119         int (* printf)(STRING_T *, const char *, ...); //输出函数
120 }FONT_T;                            //定义字体类型变量
121
```

register_single_byte_font 函数和 register_double_byte_font 函数在 firmware\include\font.c 文件中，作用是分别建立单字节链表与双字节链表，注册用户输入的单、双字节字体。注册后的每个字体都会给定一个偏移量 id。

二、 中文字体

1、 字库驱动

中文字库占用很大的存储空间，一个中文字库至少占 200kB，所以它被存放在外置的 flash 中，可以通过 fatfs 作为文件系统读取。

字库的驱动代码都在 firmware\font 内。其中，offset 是字体点阵信息首地址相对于字库头文件的偏移量，可以帮助我们定位字体的点阵信息。汉字偏移量的计算公式是汉字编码的标准规定，不同字体类型的偏移量计算公式可能不同。

代码 3-8　字体 simsun12 的 get 函数

```
54 static int
55 get(FONT_MATRIX_T *f)
56 {
57         unsigned long int offset;
58
59         offset = f->character[0];
60         offset -= 0xa1;
61         offset *= 94;
62         offset += f->character[1];
63         offset -= 0xa1;
64         offset *= 32;
65
66         f->direction = DIRECTION_H;
67         f->height = simsun12.height;
68         f->width = simsun12.width;
69
70         flash.read_32(offset + SIMSUN12_ADDRESS_BASE, font_buffer);
71         f->dat = font_buffer;
72
73         return 0;
74 }
75
```

代码 3-8 的第 59~64 句是 sinsum12 的偏移量 offset 的计算公式。

2、 示例

```
 99 int main(void)
100 {
101         int i = 0;
102         STRING_T s;
103
104         const void * font_list[] = {
105                 &simsun12,
106                 &simsun16,
107                 &simsun16f,
```

```
108                 &simhei24
109         };
110
111         const unsigned char * font_text1[] = {
112                 "simsun12  ",
113                 "simsun16  ",
114                 "simsun16f ",
115                 "simhei24  "
116         };
117
118         const unsigned char * font_text2[] = {
119                 "中国美丽的城市洛阳欢迎您",
120                 "中国美丽的城市洛阳欢迎您",
121                 "中国美丽的城市洛阳欢迎您",
122                 "中国美丽的城市洛阳欢迎您"
123         };
124
125         initialize();
126
127         font._default.single_byte = &fixedsys;
128         font._default.double_byte = &simsun16;
129
130         //TITLE
131         s.x = 100;
132         s.y = 35;
133         s.space.line = 0;
134         s.space.word = 0;
135         s.color = COLOR_RED;
136         s.background_color = COLOR_WINDOW_BACKGROUND;
137         s.inverse = NULL;
138         font.printf(&s, "-- X-GUI 汉字 Font sample --");
139
140         for (i = 0; i < 4; i++) {
141                 font._default.double_byte =
(DOUBLE_BYTE_FONT_T*)font_list[i];
142                 s.x = 25;
143                 s.y = 90 + i * 30;
144                 s.space.line = 10;
145                 s.space.word = 0;
146                 s.color = COLOR_BLACK;
147                 s.background_color = COLOR_WINDOW_BACKGROUND;
148                 s.inverse = NULL;
```

```
149                 font.printf(&s, "%s     %s", font_text1[i],
font_text2[i]);
150         }
151
152         while (1) ;
153 }
154
```

以上程序代码将分别用 simsun12、simsun16、simsun16f、simhei24 这四种等中文字体显示同一句话："中国美丽的城市洛阳欢迎您"。详细工程见 chapter_3\hanzi，程序结果如图 3-3 所示。

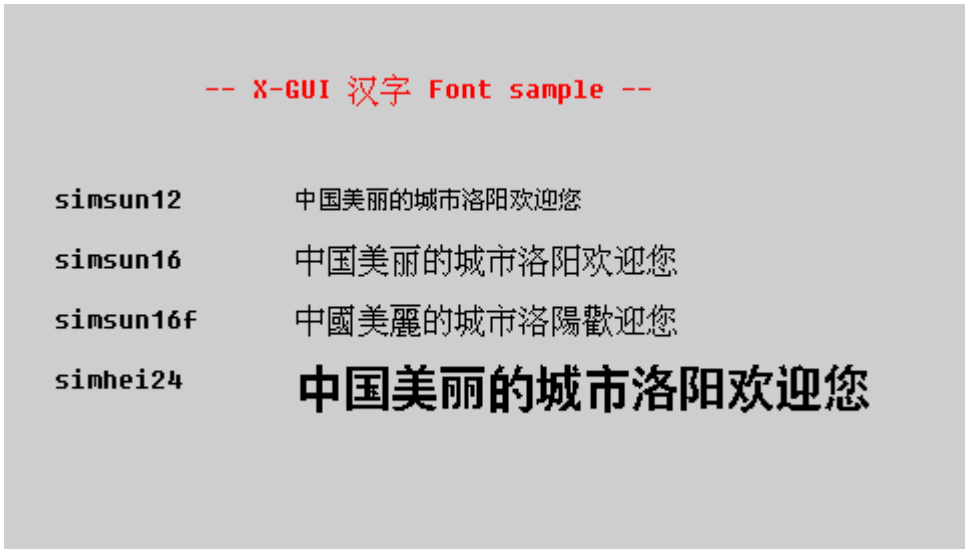

图 3-3　汉字示例

三、　等宽英文字体

1、 字库驱动

由于等宽英文字体其占用空间较小，所以与汉字字库的的存储位置不同，它们存储在 CPU 里。

等宽英文字体点阵信息的偏移量 offset 是线性的。代码 3-9 是 courier 字体的 get 函数，第 56 句是利用字体编码计算点阵首地址的公式，编码是唯一的自变量。

代码 3-9　字体 courier 的 get 函数

```
50 static int
51 get(FONT_MATRIX_T *f)
52 {
53         f->direction = DIRECTION_V;
54         f->height = courier.height;
55         f->width = courier.width;
56         f->dat = (char*)courier.dat + (f->character[0] - ' ') * 16;
57
58         return 0;
59 }
60
```

2、 示例

```
 96 int main(void)
 97 {
 98         int i;
 99         STRING_T s;
100
101         const void * font_list[] = {
102                 &fixedsys,
103                 &courier,
104                 &monaco,
105                 &borlandTE
106         };
107
108         const unsigned char * font_text[] = {
109                 "fixedsys    the quick brown fox jumps over the lazy dog",
110                 "courier     the quick brown fox jumps over the lazy dog",
111                 "monaco      the quick brown fox jumps over the lazy dog",
112                 "borlandTE   the quick brown fox jumps over the lazy dog"
113         };
114
115         initialize();
116         font._default.single_byte = &fixedsys;
117
118         //TITLE
119         s.x = 100;
120         s.y = 35;
121         s.space.line = 0;
122         s.space.word = 0;
```

```
        s.color = COLOR_RED;
        s.background_color = COLOR_WINDOW_BACKGROUND;
        s.inverse = NULL;
        font.printf(&s, "-- X-GUI monospaced Font sample --");

        for (i = 0; i < 4; i++) {
                font._default.single_byte = (SINGLE_BYTE_FONT_T*)font_list[i];
                s.x = 15;
                s.y = 90 + i * 30;
                s.space.line = 0;
                s.space.word = 0;
                s.color = COLOR_BLACK;
                s.background_color = COLOR_WINDOW_BACKGROUND;
                s.inverse = NULL;
                font.printf(&s, "%s", font_text[i]);
        }

        while (1) ;
}

```

以上程序代码将分别用 courier、borlandTE、fixedsys、monaco 这四种等宽英文字体类型显示同一句话："the quick brown fox jumps over the lazy dog"，以此来展示这四种类型显示的 26 个字母。程序运行结果如图 3-4 所示。

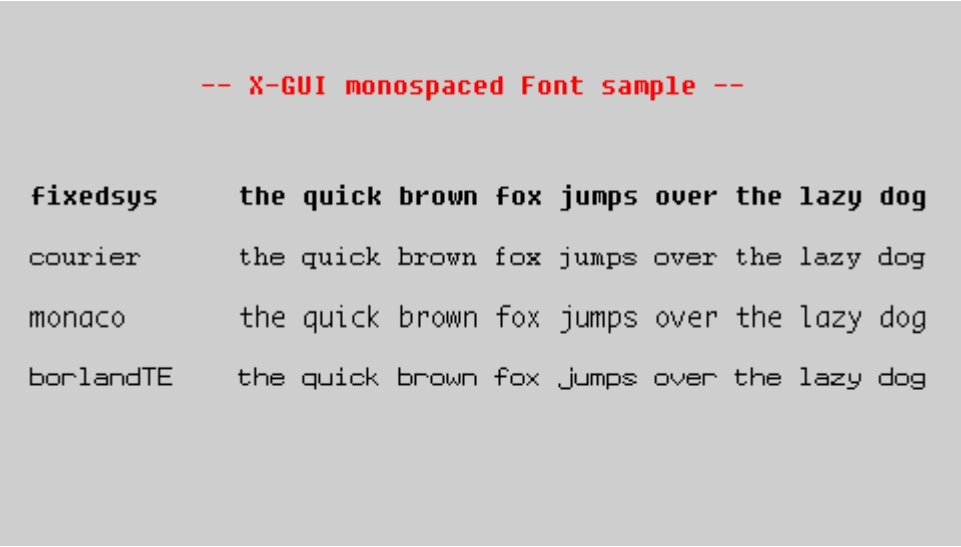

图 3-4　等宽英文字体示例

四、 比例英文字体

1、 字库驱动

比例字体的字库存储在 CPU 中。

因为比例字体的字宽未知，所以需要一个存储偏移量 offset 的数据表，这个数据表由美国人制出，有统一的规定，如代码 3-10 所示。比例字体的偏移量 offset 与等宽字体的不同，它不仅用来定位点阵信息，还可以用来计算字宽，如代码 3-11 所示。

代码 3-10　tahoma8 字体偏移量的数据表

```
36 const short int tahoma8_offset[] = {
37        //     !   "   #   $   %   &   '   (
38         0, 3, 7, 11, 19, 25, 36, 43, 45, 49,
39        // )  *   +   ,   -   .   /   0   1   2
40         53, 59, 67, 71, 75, 79, 83, 89, 95, 101,
41        // 3   4   5   6   7   8   9   :   ;   <
42         107, 113, 119, 125, 131, 137, 143, 147, 152, 159,
43        // =   >   ?   @   A   B   C   D   E   F
44         168, 175, 180, 190, 197, 203, 210, 217, 223, 229,
45        // G   H   I   J   K   L   M   N   O   P
46         236, 243, 247, 252, 258, 263, 271, 278, 286, 292,
47        // Q   R   S   T   U   V   W   X   Y   Z
48         300, 307, 313, 319, 326, 332, 342, 348, 354, 360,
49        // [   \   ]   ^   _   `   a   b   c   d
50         364, 368, 372, 380, 387, 392, 398, 404, 409, 415,
51        // e   f   g   h   i   j   k   l   m   n
52         421, 425, 431, 437, 439, 442, 447, 449, 457, 463,
53        // o   p   q   r   s   t   u   v   w   x
54         469, 475, 481, 485, 490, 494, 500, 506, 514, 520,
55        // y   z   {   |   }   ~
56         526, 531, 536, 540, 545, 553
57 };
58
```

代码 3-11　tahoma8 的 get 函数

```
169 static int
170 get(FONT_MATRIX_T *f)
171 {
172         f->direction = DIRECTION_V;
173         f->height = 16;
174         f->width = tahoma8_rc.offset[f->character[0] - ' ' + 1]
                        - tahoma8_rc.offset[f->character[0] - ' '];
175         f->dat = tahoma8_rc.dat
                      + tahoma8_rc.offset[f->character[0] - ' '] * 2;
176
177         return 0;
178 }
179
```

代码 3-11 中第 174 句是利用偏移量计算比例字体的字宽，第 175 句是利用偏移量计算在字库中字的点阵首地址。

2、 示例

```
 96 int main(void)
 97 {
 98         int i;
 99         STRING_T s;
100
101         const void * font_list[] = {
102                 &tahoma8,
103                 &tahoma9,
104                 &tahoma10,
105                 &tahoma11,
106                 &tahoma12,
107                 &tahoma26
108         };
109
110         const unsigned char * font_text[] = {
111                 "tahoma8    the quick brown fox jumps over the lazy dog",
112                 "tahoma9    the quick brown fox jumps over the lazy dog",
113                 "tahoma10   the quick brown fox jumps over the lazy dog",
114                 "tahoma11   the quick brown fox jumps over the lazy dog",
115                 "tahoma12   the quick brown fox jumps over the lazy dog",
116                 "tahoma26   the quick brown fox jumps over the lazy dog"
117         };
118
```

```
119         initialize();
120
121         font._default.single_byte = &fixedsys;
122
123         //TITLE
124         s.x = 100;
125         s.y = 20;
126         s.space.line = 0;
127         s.space.word = 0;
128         s.color = COLOR_RED;
129         s.background_color = COLOR_WINDOW_BACKGROUND;
130         s.inverse = NULL;
131         font.printf(&s, "-- X-GUI proportional Font sample --");
132
133         for (i = 0; i < 6; i++) {
134                 font._default.single_byte =
(SINGLE_BYTE_FONT_T*)font_list[i];
135                 s.x = 15;
136                 s.y = 60 + i * 30;
137                 s.space.line = 0;
138                 s.space.word = 0;
139                 s.color = COLOR_BLACK;
140                 s.background_color = COLOR_WINDOW_BACKGROUND;
141                 s.inverse = NULL;
142                 font.printf(&s, "%s", font_text[i]);
143         }
144
145         while (1) ;
146 }
147
```

以上程序代码将分别用 X-GUI 支持的六种 tahoma 比例英文字体类型显示同一句话："the quick brown fox jumps over the lazy dog "，以此来展示这六种类型显示的 26 个字母。程序运行结果如图 3-5 所示。

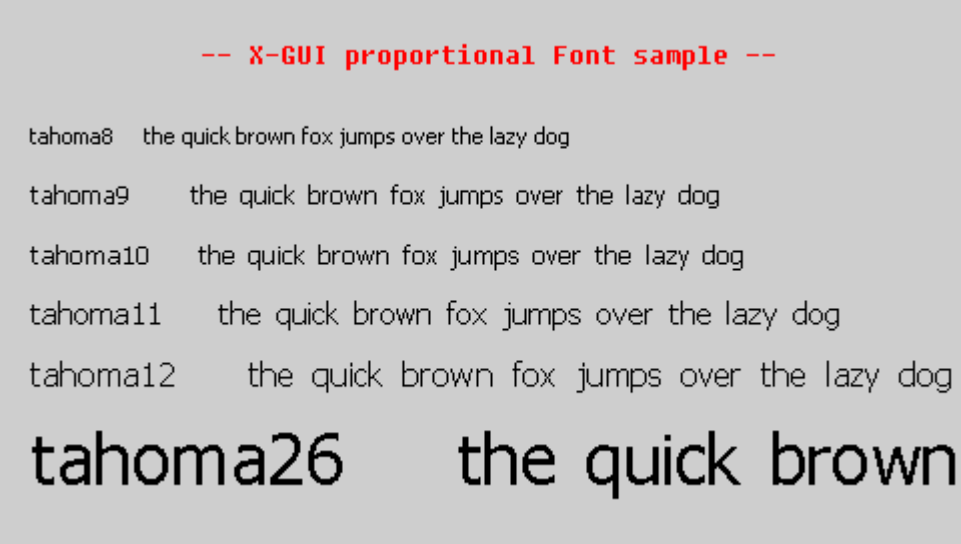

图 3-5　比例英文字体示例

第四章 GUI 驱动

GUI 驱动

本章介绍了 GUI 中定义的各个函数，如描点的，画线的，画矩形的等等。

本章分为以下几个部分：

一、initialize 函数
二、dot 函数
三、line 函数
四、line_to 函数和 set_cursor 函数
五、rect 函数
六、box 函数
七、check_box 函数
八、draw_arrow 函数
九、circle 函数
十、processbar 函数
十一、 splitter 函数
十二、 scrollbar 函数

一、 initialize 函数

本章介绍 GUI 驱动，每一部分都需要用到初始化函数，所以我们把众多的初始化函数放在一起组成一个初始化函数，在这里统一介绍，在之后的小节中将不再介绍。

代码 4-1　初始化函数

```
66  static int initialize(void)
67  {
68          int i;
69          rcc.initialize();
70          nvic.initialize();
71          spi1.initialize();
72          fsmc.initialize();
73
74          for(i = 0;i<2000000;i++);
75          font.initialize();
76          lcd.initialize();
77          lcd.cls(COLOR_WINDOW_BACKGROUND);
78          pwm.initialize(100);
79          pwm.open(100);
80
81  return 0;
82  }
83
```

代码 4-1 是初始化函数，第 68 句定义了一个整型变量 i，第 69~72 句依次是初始化时钟函数 rcc、初始化中断函数 nvic、初始化 spi1、初始化总线函数 fsmc；然后 74 句是 for 循环的延时，是为了等待 FPGA 配置成功以便输出 TFT 所用时钟；第 75 句是初始化字体；第 76 句是初始化液晶屏；第 77 句是清屏，该句可以将屏清成任意定义的颜色；第 78 句是 PWM 背光初始化；第 79 句是打开背光，最后返回一个 return，至此，初始化函数就介绍完了。

二、 dot 函数

1、 介绍

dot 函数是 GUI 图形界面里最基础的，几乎所有的图形函数都是基于 dot 函数来完成的。

2、 函数原型

```
static int dot(int x, int y, COLOR_T c);
```

3、 参数

dot 函数的参数有点的坐标（x，y）以及点的颜色。

4、 示例

```
int main(void)
{
        int i,j,k,n;
        STRING_T s;

        const unsigned char * text[]={
                "RED",
                "GREEN",
                "BLUE",
                "YELLOW",
                "PURPLE",
                "AQUA",
                "GRAY"
                };

        initialize();

        lcd.cls(COLOR_BLACK);

  //TITLE
        font._default.single_byte = &fixedsys;
        s.x = 100;
        s.y = 10;
        s.space.line=0;
        s.space.word=0;
        s.color=COLOR_RED;

```

```
192         s.inverse=NULL;
193         font.printf(&s,"-- X-GUI dot Function sample --");
194
195         //描点
196         for(n = 0; n< 7; n++){
197                 for(k = 0; k < 32; k ++){           //颜色由黑开始渐变
198                         for(i = 0; i<15; i ++){
199                                 for(j = 0; j<15;j++)gui.dot(i + k*12 +
85,j + n*30 + 55,  get_color(n,k));
200                         }
201                 }
202         for(k = 0; k < 32; k ++){                   //颜色由白开始渐变
203                 for(i = 0; i < 15; i ++){
204                         for(j = 0; j < 15;j ++)gui.dot(i + k*12 + 85,j
+ n*30 + 15 + 55,  get_colorb(n,k));
205                         }
206                 }
207         }
208
209         for(i = 0; i<7; i ++){
210             s.x = 10;
211             s.y = 64+i*30;
212             s.space.line = 0;
213             s.space.word = 0;
214             s.color = COLOR_RED;
215             s.background_color = COLOR_BLACK;
216             s.inverse = NULL;
217             font.printf(&s,"%s",text[i]);
218         }
219
220          while(1);
221 }
222
```

这个程序的作用是用描点的方式画出颜色渐变的彩条，代码中的 get_color 和 get_colorb 是自定义的两个颜色渐变的函数。详细工程见 chapter_4\dot，程序结果如图 4-1 所示。

图 4-1　dot 例程屏幕截图

三、 line 函数

1、 介绍

line 函数是 GUI 图形界面中不可缺少的函数，图形界面的很多图形都需要由画线函数来实现。

2、 函数原型

```
static int line(int /* x0 */, int /* y0 */, int /* x1 */, int /* y1 */, COLOR_T);
```

3、 参数

线的起点坐标（x0，y0）和终点坐标（x1，y1），以及线的颜色 COLOR_T。

4、 示例

```
int main(void)
{
        STRING_T s;                             //初始化字符串

        initialize();
        font._default.single_byte = &fixedsys; //字体设置成 fixedsys 类型
        //TITLE
        s.x = 100;
        s.y = 10;
        s.space.line = 0;
        s.space.word = 0;
        s.color = COLOR_RED;
        s.background_color = COLOR_WINDOW_BACKGROUND;
        s.inverse = NULL;
        font.printf(&s, "-- X-GUI line Function sample --");

        gui.line(240, 30, 180, 75, COLOR_GREEN);
        gui.line(240, 30, 300, 75, COLOR_GREEN);
        gui.line(180, 75, 300, 75, COLOR_GREEN);

        gui.line(240, 75, 150, 135, COLOR_GREEN);
        gui.line(240, 75, 330, 135, COLOR_GREEN);
        gui.line(150, 135, 330, 135, COLOR_GREEN);

```

```
121         gui.line(240, 135, 120, 195, COLOR_GREEN);
122         gui.line(240, 135, 360, 195, COLOR_GREEN);
123         gui.line(120, 195, 360, 195, COLOR_GREEN);
124
125         gui.line(200, 195, 200, 260, COLOR_BLACK);
126         gui.line(280, 195, 280, 260, COLOR_BLACK);
127
128         while (1) ;
129 }
130
```

本程序的作用是是用 line 语句画了一幅大树的简笔画，程序的详细工程见 chapter_4\line。,程序结果见图 4-2。

其中，第 96~103 行的作用是为图形加标题，内容包括设置起始坐标、字间距、行间距、字体及其背景颜色以及标题内容。第 113、117、121 句是画倾斜角为锐角的绿色线，第 114、118、122 句是画倾斜角为钝角的绿色线，第 115、119、123 句是画水平的绿色线，第 125、126 句是画竖直的黑色线。

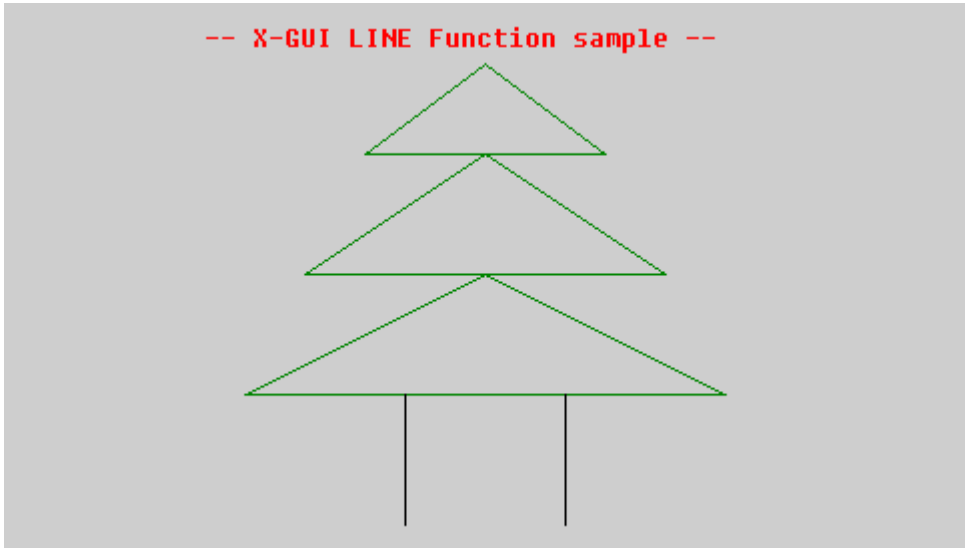

图 4-2　line 例程屏幕截图

四、 line_to 函数和 set_cursor 函数

1、介绍

line_to 和 line 一样，也是用来画线的函数，但是与 line 不同的是，line_to 函数的起点坐标是当前光标所在位置，所以只需要确定终点的坐标即可，在这里我们把 line_to 函数和 set_cursor 函数放在一起介绍。

2、函数原型

line_to 函数：

```
static int line_to(int /* x */, int /* y */, COLOR_T /* c */);
```

set_cursor 函数：

```
static int set_cursor(int /* x */, int /* y */);
```

3、参数

line_to 函数的相关参数：line 终点坐标（x，y）和线的颜色。

set_cursor 函数的相关参数：设置光标的位置（x，y）。

4、示例

```
int main(void)
{
        int i, j;
        STRING_T s;
        BOX_T b;

        initialize();
        font._default.single_byte = &fixedsys;
        //TITLE
        s.x = 50;
        s.y = 20;
        s.space.line = 0;
        s.space.word = 0;
        s.color = COLOR_RED;
        s.background_color = COLOR_WINDOW_BACKGROUND;
        s.inverse = NULL;
        font.printf(&s, "-- X-GUI line_to & set_cursor Function sample --");
```

```
112
113         //画凹陷的 box
114         b.startx = 10;
115         b.starty = 60;
116         b.endx = 470;
117         b.endy = 240;
118         b.attrib = BOX_RECESSED;
119         b.color = COLOR_BLACK;
120         gui.box(&b);
121
122         gui.set_cursor(15, 150);        //设置光标位置
123
124         for (i = 0; i < 450; i++) {     //用 line_to 画亮绿色的正弦函数
125                 j = 80 * sin(0.08 * i);
126                 gui.line_to(i + 15, j + 150, COLOR_LIME);
127         }
128
129         gui.set_cursor(15, 230);        //设置光标位置
130
131         for (i = 0; i < 450; i++) {     //用 line_to 画红色的正弦函数
132                 j = 80 * sin(0.08 * i + 1.57);
133                 gui.line_to(i + 15, j + 150, COLOR_RED);
134         }
135
136         while (1) ;
137 }
138
```

以上程序的作用是画两条正弦波，程序结果见图 4-3。具体的工程见 chapter_4 \line_to。

其中，第 122、129 句中设置的光标位置都是正弦波的起点，同时是 line_to 函数的起点，第 122~127 句是画绿色正弦波，第 129~134 句是画红色正弦波。

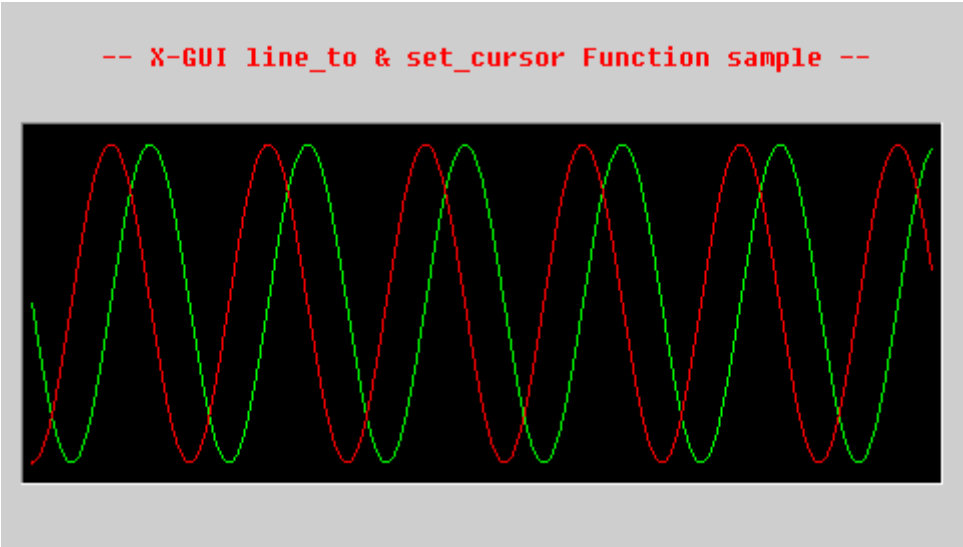

图 4-3　line_to&set_cursor 例程屏幕截图

五、 rect 函数

1、 介绍

rect 函数用来画矩形。

2、 函数原型

```
static int rect(RECT_T *);
```

3、 参数

```
typedef struct {
        int x;                      //矩形起点 x 坐标
        int y;                      //矩形起点 y 坐标
        int height;                 //矩形高度
        int width;                  //矩形高度
        int fill_flag;              //是否填充,0:不填充,1:填充,2:填充图片
        COLOR_T color;              //矩形的颜色
        const unsigned short int *p;//指针
}RECT_T;
```

4、 示例

```
int main(void)
{
        int i, j;
        STRING_T s;
        RECT_T r;

        for (i = 0; i < 1000000; i++) ;

        const int height1[] = { //2011 年降雨量高度
                40,
                150,
                170,
                70
        };

        const int height2[] = { //2011 年降雨量高度
                35,
                180,
                160,
```

```
117                 80
118         };
119
120         static char * text[] = {
121                 " 第一季度", " 第二季度", "第三季度", "第四季度"
122         };
123
124         initialize();
125
126         font._default.single_byte = &fixedsys;
127         font._default.double_byte = &simsun16;
128
129         //TITLE
130         s.x = 100;
131         s.y = 20;
132         s.space.line = 0;
133         s.space.word = 0;
134         s.color = COLOR_RED;
135         s.background_color = COLOR_WINDOW_BACKGROUND;
136         s.inverse = NULL;
137         font.printf(&s, "-- X-GUI rect Function Sample --");
138
139         //画坐标轴
140         gui.line(30, 225, 430, 225, COLOR_BLACK);
141         gui.line(30, 50, 30, 225, COLOR_BLACK);
142
143         gui.arrow(30, 50, 0, COLOR_BLACK);
144         gui.arrow(430, 225, 1, COLOR_BLACK);
145
146         //画 2011 年降雨量的柱形图
147         for (i = 0; i < 4; i++) {
148                 r.x = 50 + i * 90;
149                 r.y = 225 - height1[i];
150                 r.width = 25;
151                 r.height = height1[i];
152                 r.fill_flag = 0;
153                 r.color = COLOR_BLACK;
154                 gui.rect(&r);
155         }
156
157         //画 2012 年降雨量的柱形图
158         for (i = 0; i < 4; i++) {
159                 r.x = 80 + i * 90;
160                 r.y = 225 - height2[i];
```

```
161                 r.width = 25;
162                 r.height = height2[i];
163                 r.fill_flag = 1;
164                 r.color = COLOR_BLUE;
165                 gui.rect(&r);
166         }
167
168         for (j = 0; j < 4; j++) {
169                 s.x = 40 + j * 90;
170                 s.y = 235;
171                 s.space.line = 2;
172                 s.space.word = 0;
173                 s.color = COLOR_BLACK;
174                 s.background_color = COLOR_WINDOW_BACKGROUND;
175                 s.inverse = NULL;
176                 font.printf(&s, "%s", text[j]);
177         }
178
179         r.x = 320;
180         r.y = 60;
181         r.width = 140;
182         r.height = 50;
183         r.fill_flag = 2;
184         r.p = (unsigned short int*)rc_rect1;
185         gui.rect(&r);
186
187         while (1) ;
188 }
189
```

以上程序代码的作用是画一个关于降雨量的柱状图，详细工程见 chapter_4\rect。其中，rc_rect1 在工程的资源中，它存储的图片如图 4-4 所示。该工程运行的结果如图 4-5 所示。

图 4-4　rc_rect1 图片

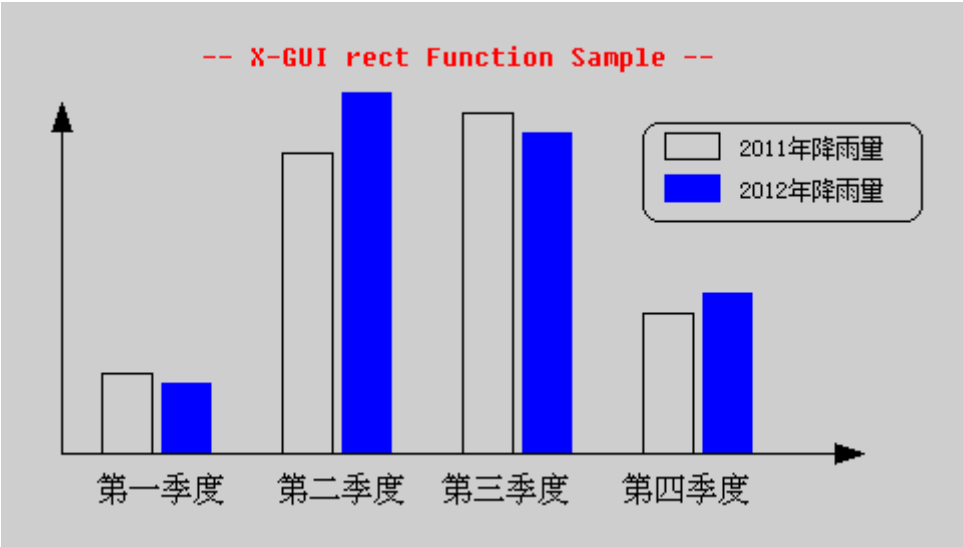

图 4-5　rect 例程屏幕截图

六、 box 函数

1、 介绍

box 函数用来画具有立体效果的 box。

2、 函数原型

```
static int box(BOX_T *);
```

3、 参数

```
typedef struct {
        int startx;                     //起点 x 坐标
        int starty;                     //起点 y 坐标
        int endx;                       //终点 x 坐标
        int endy;                       //终点 y 坐标
        COLOR_T color;                  //box 的颜色
        unsigned short attrib;          //属性
}BOX_T;
```

box 的属性：当 b.attrib=BOX_RAISED 时，box 是凸起的；当 b.attrib=BOX_RECESSED 时，box 是凹陷的；当 b.attrib= BOX_FRAME 时，box 的四周是明亮的线。

4、 示例

```
int main(void)
{
        BOX_T b;
        STRING_T s;
        initialize();

        font._default.single_byte = &fixedsys;

        //TITLE
        s.x = 100;
        s.y = 10;
        s.space.line = 0;
        s.space.word = 0;
        s.color = COLOR_RED;
        s.background_color = COLOR_WINDOW_BACKGROUND;
        s.inverse = NULL;
```

```
113         font.printf(&s, "-- X-GUI box Function sample --");
114
115         //画凸起的 box
116         b.startx = 80;
117         b.starty = 40;
118         b.endx = 180;
119         b.endy = 100;
120         b.attrib = BOX_RAISED;
121         b.color = COLOR_WINDOW_BACKGROUND;
122         gui.box(&b);
123
124         //box 的属性注释
125         s.x = 200;
126         s.y = 60;
127         s.space.line = 0;
128         s.space.word = 0;
129         s.color = COLOR_BLACK;
130         s.background_color = COLOR_WINDOW_BACKGROUND;
131         s.inverse = NULL;
132         font.printf(&s, "box attirb = BOX_RAISED");
133
134         //画凹陷的 box
135         b.startx = 80;
136         b.starty = 120;
137         b.endx = 180;
138         b.endy = 180;
139         b.attrib = BOX_RECESSED;
140         b.color = COLOR_WINDOW_BACKGROUND;
141         gui.box(&b);
142
143         //box 的属性注释
144         s.x = 200;
145         s.y = 140;
146         s.space.line = 0;
147         s.space.word = 0;
148         s.color = COLOR_BLACK;
149         s.background_color = COLOR_WINDOW_BACKGROUND;
150         s.inverse = NULL;
151         font.printf(&s, "box attirb = BOX_RECESSED");
152
153         //画四周有明亮的线的 box
154         b.startx = 80;
155         b.starty = 200;
156         b.endx = 180;
```

```
157         b.endy = 260;
158         b.attrib = BOX_FRAME;
159         b.color = COLOR_WINDOW_BACKGROUND;
160         gui.box(&b);
161
162         //box 的属性注释
163         s.x = 200;
164         s.y = 220;
165         s.space.line = 0;
166         s.space.word = 0;
167         s.color = COLOR_BLACK;
168         s.background_color = COLOR_WINDOW_BACKGROUND;
169         s.inverse = NULL;
170         font.printf(&s, "box attirb = BOX_FRAME");
171
172         while (1) ;
173 }
174
```

以上程序的作用是画三个具有不同属性的 box，具体工程见 chapter_4\box，该程序的结果如图 4-6 所示。

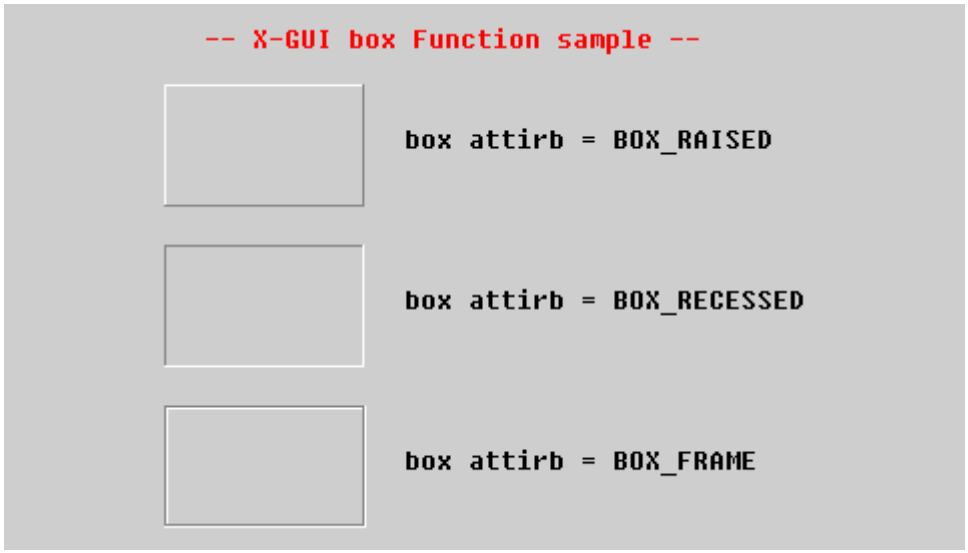

图 4-6　box 例程屏幕截图

七、 check_box 函数

1、 介绍

check_box 是查看某选项的当前状态是否选中的一种图形。

2、 函数原型

```
static int check_box(CHECK_BOX_T *);
```

3、 参数

```
typedef struct {
        int attrib;      //属性
        int x;           //x 坐标
        int y;           //y 坐标
}CHECK_BOX_T;
```

check_box 的属性：当 c.attrib = 0 时，check_box 为空心的；当 c.attrib = 1 时，check_box 是填充的。

4、 示例

```
int main(void)
{
        CHECK_BOX_T c;
        STRING_T s;
        initialize();

        font._default.single_byte = &fixedsys;

        //TITLE
        s.x = 80;
        s.y = 30;
        s.space.line = 0;
        s.space.word = 0;
        s.color = COLOR_RED;
        s.background_color = COLOR_WINDOW_BACKGROUND;
        s.inverse = NULL;
        font.printf(&s, "-- X-GUI check_box Function sample --");

        //画空心的 check_box
        c.attrib = 0;
        c.x = 80;
```

```
118         c.y = 100;
119         gui.check_box(&c);
120
121         //check_box 属性注释
122         s.x = 95;
123         s.y = 97;
124         s.space.line = 0;
125         s.space.word = 0;
126         s.color = COLOR_BLACK;
127         s.background_color = COLOR_WINDOW_BACKGROUND;
128         s.inverse = NULL;
129         font.printf(&s, "check_box attirb = 0");
130
131         //画实心的 check_box
132         c.attrib = 1;
133         c.x = 80;
134         c.y = 180;
135         gui.check_box(&c);
136
137         //check_box 属性注释
138         s.x = 95;
139         s.y = 177;
140         s.space.line = 0;
141         s.space.word = 0;
142         s.color = COLOR_BLACK;
143         s.background_color = COLOR_WINDOW_BACKGROUND;
144         s.inverse = NULL;
145         font.printf(&s, "check_box attirb = 1");
146
147         while (1) ;
148 }
149
```

以上程序的具体工程见 chapter_4\ check_box，程序结果见图 4-7。

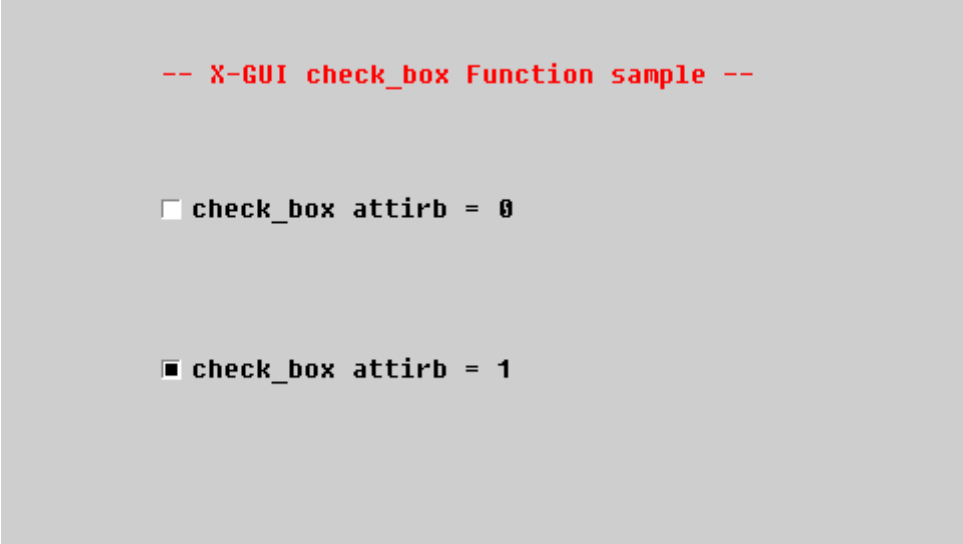

图 4-7　check_box 例程屏幕截图

八、 draw_arrow 函数

1、 介绍

图形界面里，通常需要用箭头起指示作用，draw_arrow 函数就是画箭头的函数。

2、 函数原型

```
static int draw_arrow(int x, int y, int dir, COLOR_T c);
```

3、 参数

箭头坐标（x，y），箭头的方向 dir 和箭头的颜色。

箭头的属性：当 dir = 0 时，箭头朝上；当 dir = 1 时，箭头向右；当 dir = 2 时，箭头朝下；当 dir = 3 时，箭头向左。

4、 示例

```
97  int main(void)
98  {
99          STRING_T s;
100         initialize();
101
102         font._default.single_byte = &fixedsys;
103
104         //TITLE
105         s.x = 100;
106         s.y = 20;
107         s.space.line = 0;
108         s.space.word = 0;
109         s.color = COLOR_RED;
110         s.background_color = COLOR_WINDOW_BACKGROUND;
111         s.inverse = NULL;
112         font.printf(&s, "-- X-GUI draw_arrow Function sample --");
113
114         gui.line(160, 155, 310, 155, COLOR_BLACK);
115         gui.line(235, 80, 235, 230, COLOR_BLACK);
116
117         gui.arrow(235, 76, 0, COLOR_BLACK);        //画向上的箭头
118
119         s.x = 175;
120         s.y = 55;
121         s.space.line = 0;
```

```
122         s.space.word = 0;
123         s.color = COLOR_BLACK;
124         s.background_color = COLOR_WINDOW_BACKGROUND;
125         s.inverse = NULL;
126         font.printf(&s, "draw_arrow DIR = 0");
127
128         gui.arrow(325, 155, 1, COLOR_BLACK);       //画向右的箭头
129
130         s.x = 330;
131         s.y = 147;
132         s.space.line = 0;
133         s.space.word = 0;
134         s.color = COLOR_BLACK;
135         s.background_color = COLOR_WINDOW_BACKGROUND;
136         s.inverse = NULL;
137         font.printf(&s, "draw_arrow DIR = 1");
138
139         gui.arrow(235, 230, 2, COLOR_BLACK);       //画向下的箭头
140
141         s.x = 175;
142         s.y = 240;
143         s.space.line = 0;
144         s.space.word = 0;
145         s.color = COLOR_BLACK;
146         s.background_color = COLOR_WINDOW_BACKGROUND;
147         s.inverse = NULL;
148         font.printf(&s, "draw_arrow DIR = 2");
149
150         gui.arrow(150, 155, 3, COLOR_BLACK);       //画向左的箭头
151
152         s.x = 5;
153         s.y = 147;
154         s.space.line = 0;
155         s.space.word = 0;
156         s.color = COLOR_BLACK;
157         s.background_color = COLOR_WINDOW_BACKGROUND;
158         s.inverse = NULL;
159         font.printf(&s, "draw_arrow DIR = 3");
160
161         while (1) ;
162 }
163
```

上述程序的作用是画四个具有不同正方向的箭头，详细工程见 chapter_4\draw_arrow，

程序结果见图 4-8。

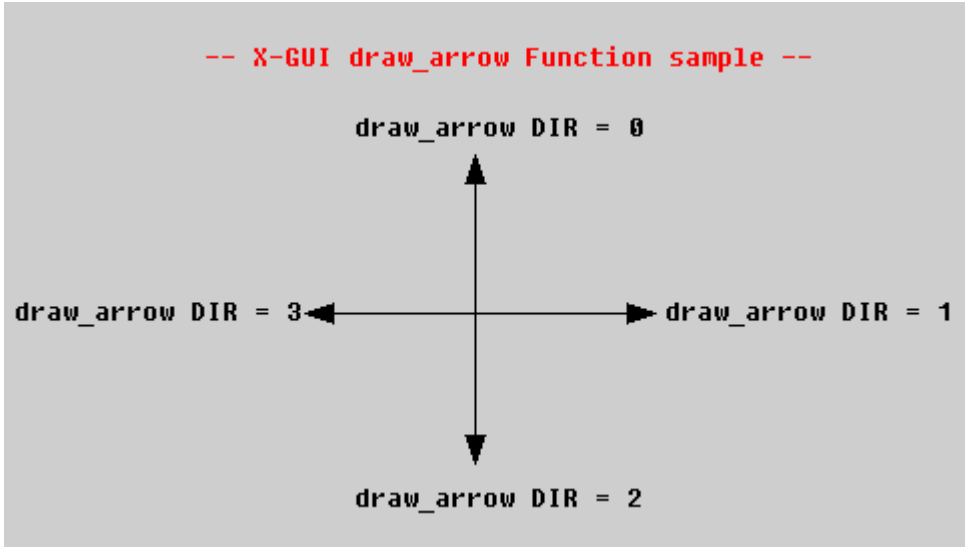

图 4-8　draw_arrow 例程屏幕截图

九、 circle 函数

1、 介绍

circle 函数是用来画圆的。圆是图形界面不可缺少的一种图形，可以分为填充和不填充两种类型。

2、 函数原型

```
static int circle(int x, int y, int r, int fill, COLOR_T c);
```

3、 参数

圆心坐标（x，y），圆半径长 r，是否填充 fill，圆的颜色。

圆的属性：当 circle_fill = 0 时，为空心圆；当 circle_fill = 1 时，为实心圆。

4、 示例

```
int main(void)
{
        STRING_T s;
        initialize();

        font._default.single_byte = &fixedsys;

        //TITLE
        s.x = 100;
        s.y = 25;
        s.space.line = 0;
        s.space.word = 0;
        s.color = COLOR_RED;
        s.background_color = COLOR_WINDOW_BACKGROUND;
        s.inverse = NULL;
        font.printf(&s, "-- X-GUI circle Function Sample --");

        //画三个实心圆
        gui.circle(100, 140, 50, 1, COLOR_RED);
        gui.circle(200, 140, 50, 1, COLOR_GREEN);
        gui.circle(150, 170, 50, 1, COLOR_BLUE);

        //实心圆属性注释
        s.x = 85;
        s.y = 230;
```

```
122         s.space.line = 0;
123         s.space.word = 0;
124         s.color = COLOR_BLACK;
125         s.background_color = COLOR_WINDOW_BACKGROUND;
126         s.inverse = NULL;
127         font.printf(&s, "circle fill = 1");
128
129         //画三个空心圆
130         gui.circle(330, 140, 50, 0, COLOR_RED);
131         gui.circle(390, 140, 50, 0, COLOR_GREEN);
132         gui.circle(360, 170, 50, 0, COLOR_BLUE);
133
134         //空心圆属性注释
135         s.x = 300;
136         s.y = 230;
137         s.space.line = 0;
138         s.space.word = 0;
139         s.color = COLOR_BLACK;
140         s.background_color = COLOR_WINDOW_BACKGROUND;
141         s.inverse = NULL;
142         font.printf(&s, "circle fill = 0");
143
144         while (1) ;
145 }
146
```

上述程序的作用是画两组具有不同属性的圆，详细工程见 chapter_4\ circle，运行结果见图 4-9。

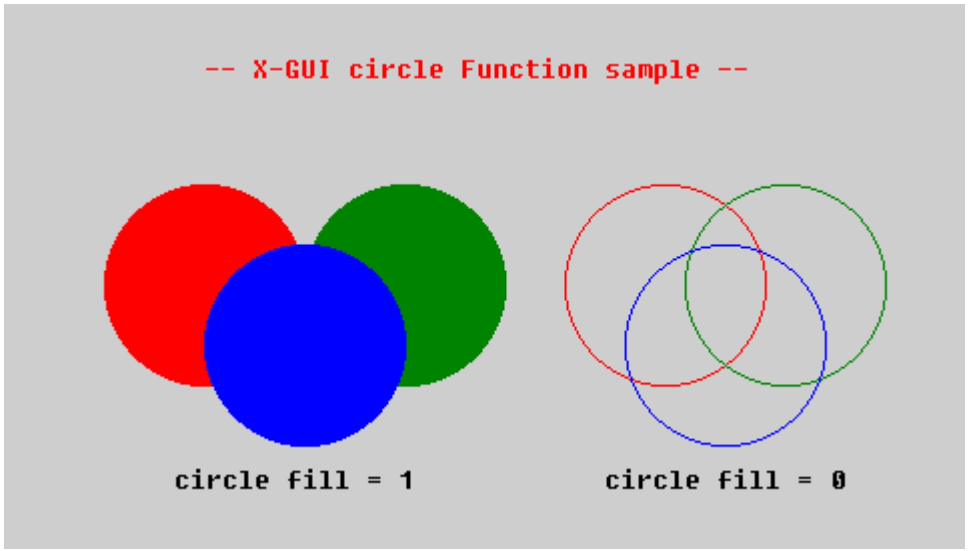

图 4-9　circle 例程屏幕截图

十、 processbar 函数

1、 介绍

在 GUI 图形界面中，有时需要查看某一状态来确定它的进度，这时可以选用进度条来显示其当前的状态。进度条可以使界面显得非常直观、清晰。

2、 函数原型

```
static int processbar(PROCESSBAR_T *);
```

3、 参数

```
typedef struct {
        int x;                          //x 坐标
        int y;                          //y 坐标
        int width;                      //进度条宽度
        int percentage;                 //所占的百分比
        COLOR_T color;                  //进度条的颜色
}PROCESSBAR_T;
```

4、 示例

```
 97 int main(void)
 98 {
 99         STRING_T s;
100         PROCESSBAR_T pr;
101         initialize();
102
103         font._default.single_byte = &fixedsys;
104
105         //TITLE
106         s.x = 100;
107         s.y = 30;
108         s.space.line = 0;
109         s.space.word = 0;
110         s.color = COLOR_RED;
111         s.background_color = COLOR_WINDOW_BACKGROUND;
112         s.inverse = NULL;
113         font.printf(&s, "-- X-GUI processbar Function sample --");
114
115         //画绿色进度条
116         pr.x = 20;
117         pr.y = 80;
```

```
118         pr.width = 240;
119         pr.color = COLOR_GREEN;
120         pr.percentage = 90;
121         gui.processbar(&pr);
122
123         //绿色进度条注释
124         s.x = 280;
125         s.y = • ;
126         s.space.line = 0;
127         s.space.word = 0;
128         s.color = COLOR_BLACK;
129         s.background_color = COLOR_WINDOW_BACKGROUND;
130         s.inverse = NULL;
131         font.printf(&s, "pr.color = COLOR_GREEN");
132
133         //画蓝色进度条
134         pr.x = 20;
135         pr.y = 140;
136         pr.width = 240;
137         pr.color = COLOR_BLUE;
138         pr.percentage = 55;
139         gui.processbar(&pr);
140
141         //蓝色进度条注释
142         s.x = 280;
143         s.y = 142;
144         s.space.line = 0;
145         s.space.word = 0;
146         s.color = COLOR_BLACK;
147         s.background_color = COLOR_WINDOW_BACKGROUND;
148         s.inverse = NULL;
149         font.printf(&s, "pr.color = COLOR_BLUE");
150
151         //画红色进度条
152         pr.x = 20;
153         pr.y = 200;
154         pr.width = 240;
155         pr.color = COLOR_RED;
156         pr.percentage = 15;
157         gui.processbar(&pr);
158
159         //红色进度条注释
160         s.x = 280;
```

```
161         s.y = 202;
162         s.space.line = 0;
163         s.space.word = 0;
164         s.color = COLOR_BLACK;
165         s.background_color = COLOR_WINDOW_BACKGROUND;
166         s.inverse = NULL;
167         font.printf(&s, "pr.color = COLOR_RED");
168
169         while (1) ;
170 }
171
```

以上程序代码的详细具体的工程见 chapter_4\processbar，程序的运行结果见图 4-10。

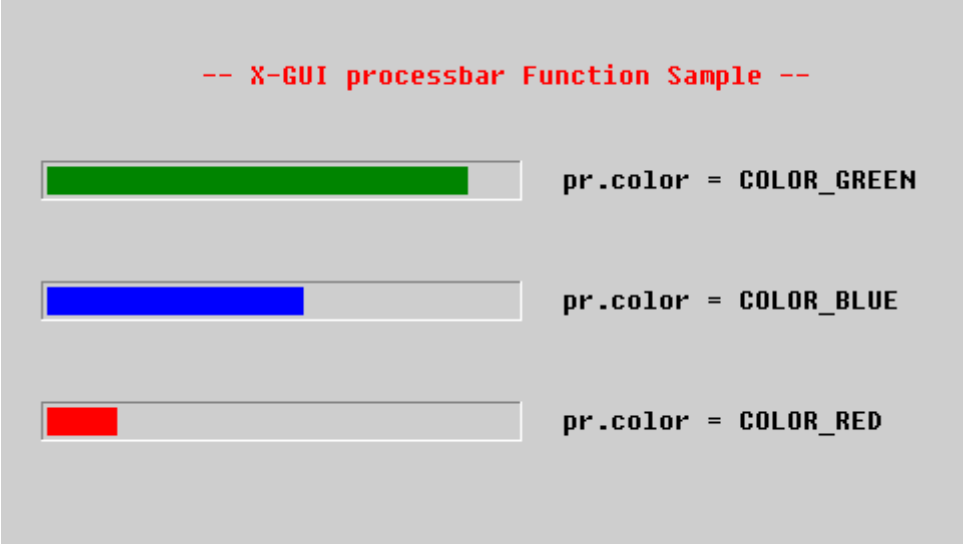

图 4-10　processbar 例程屏幕截图

十一、 splitter 函数

1、 介绍

分隔符的用途比较广泛，Window 下也比较常见，如图 4-11 所示。

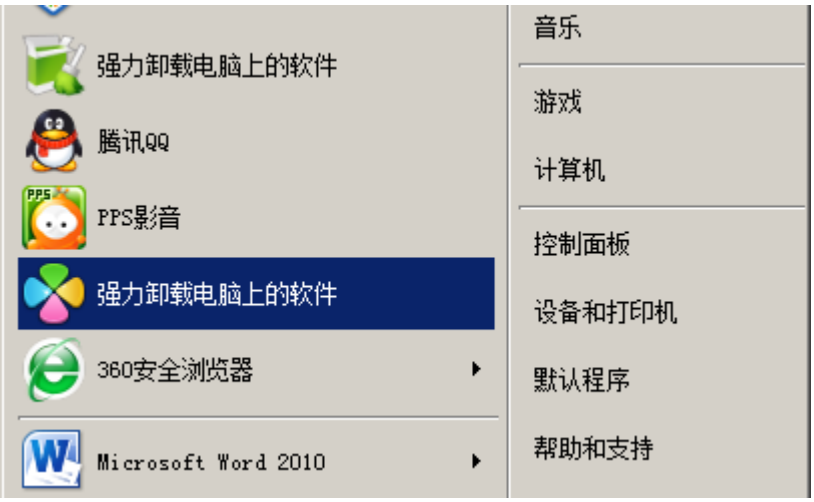

图 4-11 Window 开始界面截图

分隔符有水平、竖直这两种类型。

2、 函数原型

```
static int splitter(SPLITTER_T *);
```

3、 参数

```
typedef struct {
        int attrib;             //属性
        int x;                  //x 坐标
        int y;                  //y 坐标
        int extend;             //长度
}SPLITTER_T;
```

分隔符的属性：当 sp.attrib = SPLITTER_H | SPLITTER_RECESSED 时，为水平分隔符，且其特性是凹陷的；当 sp.attrib = SPLITTER_V | SPLITTER_RECESSED 时，为竖直分隔符。

4、 示例

```
int main(void)
{
```

```
99          int i;
100         BOX_T b;
101         STRING_T s;
102         SPLITTER_T sp;
103
104         unsigned char * text[7] = {
105                 "So far so good",
106                 "No pains, no gains",
107                 "Never say goodbye",
108                 "It is up to you",
109                 "The sooner the better",
110                 "It is up in the air ",
111                 "Think nothing of it"
112         };
113
114         initialize();
115         font._default.single_byte = &fixedsys;
116
117         //TITLE
118         s.x = 100;
119         s.y = 25;
120         s.space.line = 0;
121         s.space.word = 0;
122         s.color = COLOR_RED;
123         s.background_color = COLOR_WINDOW_BACKGROUND;
124         s.inverse = NULL;
125         font.printf(&s, "-- X-GUI splitter Function sample --");
126
127         //画 box
128         b.startx = 30;
129         b.starty = 50;
130         b.endx = 450;
131         b.endy = 250;
132         b.attrib = BOX_RAISED;
133         b.color = COLOR_WINDOW_BACKGROUND;
134         gui.box(&b);
135
136         for (i = 0; i < 3; i++) {
137                 s.x = 50;
138                 s.y = 80 + i * 60;
139                 s.space.line = 0;
140                 s.space.word = 0;
141                 s.color = COLOR_BLACK;
142                 s.background_color = COLOR_WINDOW_BACKGROUND;
```

```
143                 s.inverse = NULL;
144                 font.printf(&s, "%s", text[i]);
145         }
146
147         for (i = 3; i < 7; i++) {
148                 s.x = 260;
149                 s.y = 80 + (i - 3) * 40;
150                 s.space.line = 0;
151                 s.space.word = 0;
152                 s.color = COLOR_BLACK;
153                 s.background_color = COLOR_WINDOW_BACKGROUND;
154                 s.inverse = NULL;
155                 font.printf(&s, "%s", text[i]);
156         }
157
158         //画水平凹陷的分隔符
159         sp.x = 50;
160         sp.y = 180;
161         sp.extend = 150;
162         sp.attrib = SPLITTER_H | SPLITTER_RECESSED;
163         gui.splitter(&sp);
164
165         //画竖直凹陷的分隔符
166         sp.x = 230;
167         sp.y = 70;
168         sp.extend = 160;
169         sp.attrib = SPLITTER_V | SPLITTER_RECESSED;
170         gui.splitter(&sp);
171
172         //画水平凹陷的分隔符
173         sp.x = 260;
174         sp.y = 110;
175         sp.extend = 160;
176         sp.attrib = SPLITTER_H | SPLITTER_RECESSED;
177         gui.splitter(&sp);
178
179         //画水平凹陷的分隔符
180         sp.x = 260;
181         sp.y = 150;
182         sp.extend = 160;
183         sp.attrib = SPLITTER_H | SPLITTER_RECESSED;
184         gui.splitter(&sp);
185
186         while (1) ;
```

```
187 }
188
```

以上程序的作用是用分隔符画一个与图 4-11 相似的界面。代码的详细工程见 chapter_4\splitier，程序运行的结果见图 4-12。

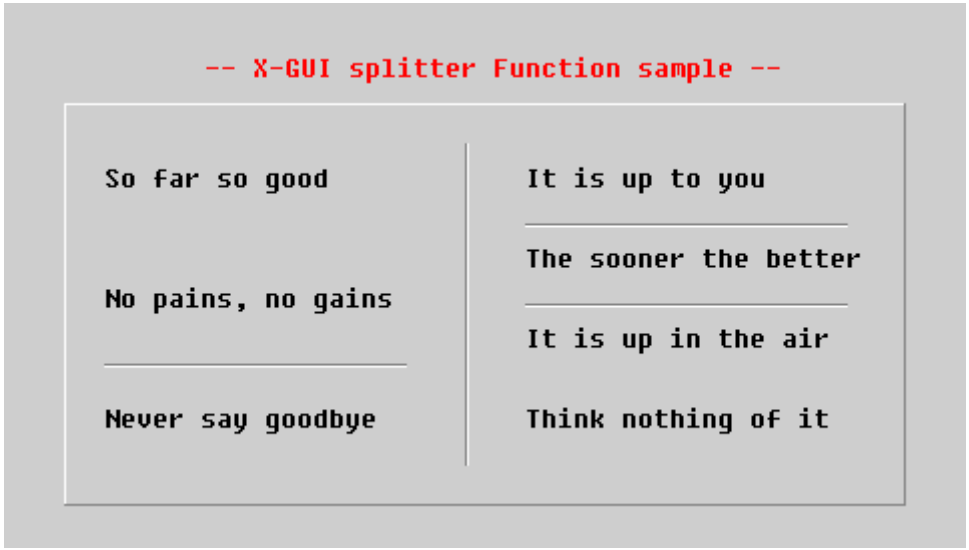

图 4-12　splitter 例程屏幕截图

十二、 scrollbar 函数

1、 介绍

在 GUI 图形界面中，滚动条用于辅助翻页，以便查看当前窗口以外的文件内容。

2、 函数原型

```
static int scrollbar(SCROLLBAR_T *);
```

3、 参数

我们给 scrollbar 函数定义了一个结构体(SCROLLBAR_T *);

```
typedef struct {
        short int x;            //起点 x 坐标
        short int y;            //起点 y 坐标
        short int height;       //滚动条高度
        short int sum;          //滚动条总长
        short int sum_per_page; //当前页所占的比例
        short int position;     //当前位置
        short int attrib;       //属性
}SCROLLBAR_T;
```

4、 示例

```
int main(void)
{
        int i;
        STRING_T s;
        BOX_T b;
        SCROLLBAR_T sc;
        initialize();

        font._default.single_byte = &fixedsys;

        //TITLE
        s.x = 100;
        s.y = 10;
        s.space.line = 0;
        s.space.word = 0;
        s.color = COLOR_RED;
        s.background_color = COLOR_WINDOW_BACKGROUND;
        s.inverse = NULL;
```

```
115         font.printf(&s, "-- X-GUI scrollbar Function sample --");
116
117         //画 box
118         for (i = 0; i < 3; i++) {
119                 b.startx = 20 + i * 150;
120                 b.starty = 40;
121                 b.endx = 150 + i * 150;
122                 b.endy = 220;
123                 b.attrib = BOX_RECESSED;
124                 b.color = COLOR_WINDOW_BACKGROUND;
125                 gui.box(&b);
126         }
127
128         //不出现滚动体滚动条
129         sc.x = 133;
130         sc.y = 40;
131         sc.height = 180;
132         sc.sum_per_page = 180;
133         sc.sum = 180;
134         sc.position = 0;
135         gui.scrollbar(&sc);
136
137         //滚动体占总长的比例较小
138         sc.x = 283;
139         sc.y = 40;
140         sc.height = 180;
141         sc.sum_per_page = 30;
142         sc.sum = 300;
143         sc.position = 170;
144         gui.scrollbar(&sc);
145
146         //滚动体占总长的比例较大
147         sc.x = 433;
148         sc.y = 40;
149         sc.height = 180;
150         sc.sum_per_page = 170;
151         sc.sum = 180;
152         sc.position = 160;
153         gui.scrollbar(&sc);
154
155         while (1) ;
156 }
157
```

以上程序代码的作用是画具有三种典型状态的滚动条，具体的工程见 chapter_4\scrollbar\，程序运行结果见图 4-13。

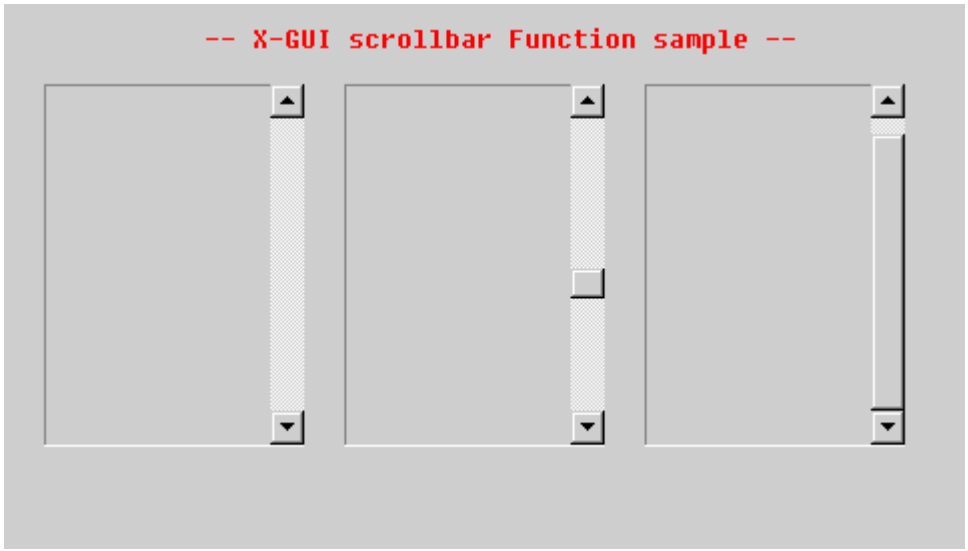

图 4-13　scrollbar 例程屏幕截图