r

基于汉字字模的 *Stellaris GUI* 控件驱动设计

摘要

TI Stellaris GUI 在 TI Stellaris 系列控制器上为用户提供了一系列免费的原始绘图和控件的显示方法，能够方便绘制用户界面，但由于 Stellaris GUI 不支持中文汉字，对中国用户的实际需求难以满足，在实际应用中经常存在用户对该方面进行咨询，因此开发支持汉字显示的 Stellaris GUI 控件驱动是非常有必要的。

内容

1. 基于汉字字模的 Stellaris GUI 控件驱动设计简介

StellarisWare 软件是 TI 提供的一套扩展软件，旨在简化和加速基于 Stellaris（Cortex-M3）的微处理器的应用开发。Stellaris GUI 是一组免版税的图形基元和小工具集，主要用于创建图形用户界面。但由于 Stellaris GUI 控件本身并不支持汉字显示，在中国市场无论是对客户需求的满足还是对 StellarisWare 本身的推广都造成了一定的困难；另外，由于片上 Flash 资源有限，不能够支持汉字库的放置，因此需要开发基于汉字字模的 Stellaris GUI 的控件驱动。

2．系统结构和总体设计方案

TI Stellaris GUI 在 TI Stellaris 系列控制器上为用户提供了一系列免费的原始绘图和控件的显示方法，整个图形库采用的是分层设计方法，如图 1 所示，分别为：显示驱动层，基本图元层，控件层。

（1） 显示驱动层：提供与显示屏相关的具体底层连接实现方式；

（2） 基本图元层：基于显示驱动层，为用户提供多种方便的原始图形绘制方法，包括图片、圆、文本等；

（3） 控件层：基于原始会图层，为用户提供多种便捷实用的控件绘制方法，包括按键、复选框、单选框等控件功能

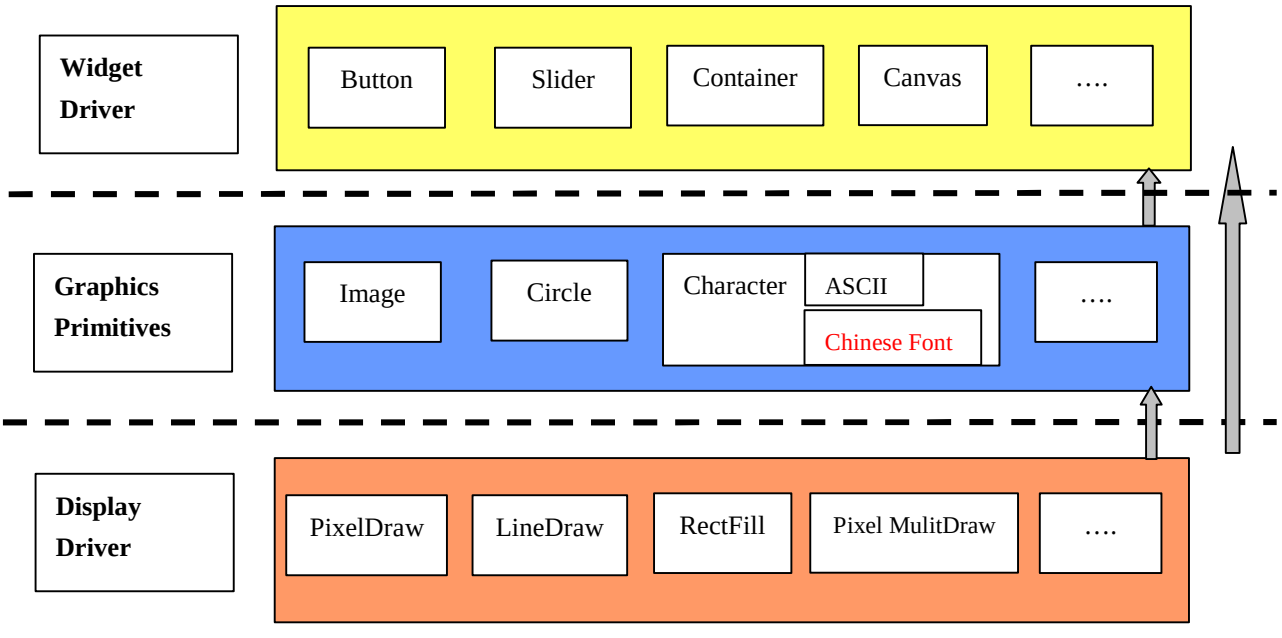

图 1 Stellaris GUI 层次图

以绘制控件 Pushbutton 为例，其函数流程如图 2 所示。其中 WidgetAdd 为添加 RectangularButton 控件，WidgetPaint(WIDGET_ROOT)用于生成控件树的根，而 WidgetMessageQueueProcess 用来处理控件消息队。

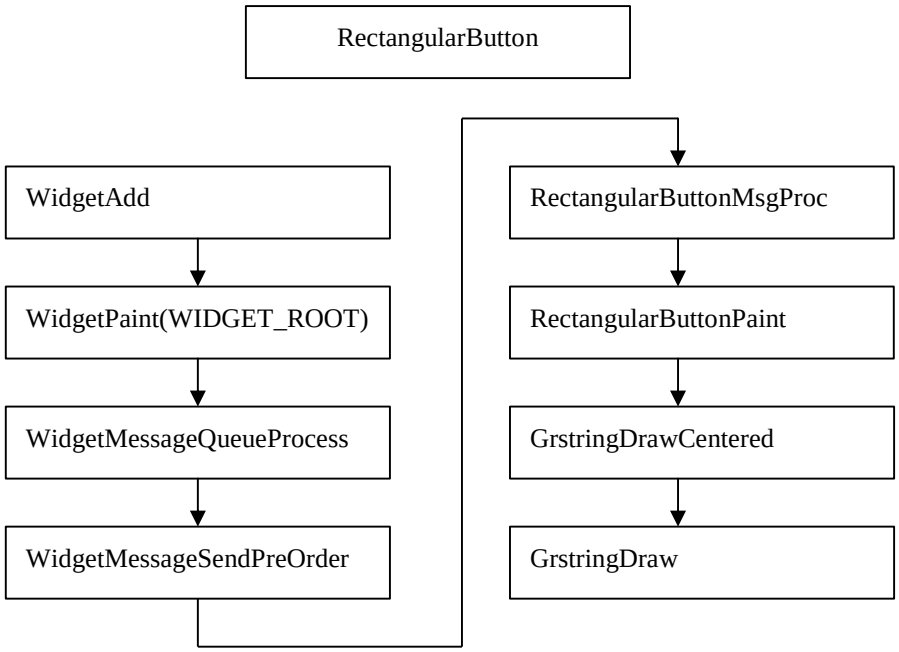

图 2 控件 PushButton 绘制函数流程图

3．汉字字模提取及驱动设计

3.1 汉字字模提取及索引

汉字的字模提取是从第一列开始向下取 8 个点作为一个字节，然后从第二列开始向下取 8 个点作为第二个字节...依此类推。如果最后不足 8 个点就补满 8 位。取模顺序是从低到高，即第一个点作为最低位。如图 3 所示。

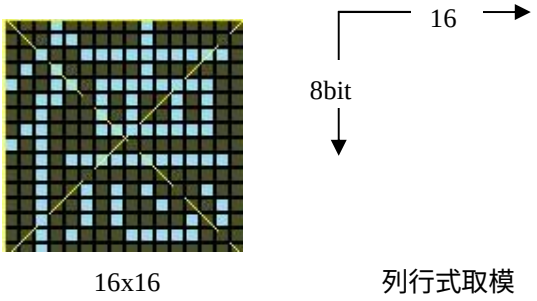

图 3 汉字字模的提取方式

在一般工程项目中，使用的汉字字数有限（几十个汉字），因此，只要提取所需使用的汉字字模即可。以生成 16x16 的点阵汉字为例，每个汉字点阵数据占用 32 个字节，索引码占 3 个字节，一个汉字是由区码和位码两个字节构成，另外按照字符串格式存储，最后有一个\0。在 grlib.h 中定义的汉字字模数据结构为：

```
//***********汉字字模的数据结构定义***********//
typedef struct                                  //汉字字模数据结构
{
const unsigned char *Msk;                 //点阵数据指针
```

```
unsigned char Index[3];                    //汉字内码索引
}
typ_CHFont;
```

汉字的索引是通过比较汉字的区码与位码是否相同，来确定其在结构体数组中的位置，然后由指针 Msk 指向其点阵数据数组。

3.2 控件接口设计

为保证 Stellaris GUI 中字母与汉字所使用接口函数的一致性，在原有的字母字体结构中加入汉字结构：

```
//**********字体的数据结构定义**********//
typedef struct
{
  //The format of the font.  Can be one of FONT_FMT_UNCOMPRESSED or
  //FONT_FMT_PIXEL_RLE.

  unsigned char ucFormat;

  //The maximum width of a character; this is the width of the widest
  //character in the font, though any individual character may be narrower
  //than this width.

  unsigned char ucMaxWidth;

  // The height of the character cell; this may be taller than the font data
  //for the characters (to provide inter-line spacing).

  unsigned char ucHeight;

  //The offset between the top of the character cell and the baseline of
  //the glyph.  The baseline is the bottom row of a capital letter, below
  //which only the descenders of the lower case letters occur.

  unsigned char ucBaseline;

  //The offset within pucData to the data for each ASCII character in the font. Or
  //pusOffset[0] is the number of Chinese Font that stored in CHFontdata when the
  //Chinese Font is selected.

  unsigned short pusOffset[96];

  //A pointer to the data for ASCII font.

  const unsigned char *pucData;
```

```
  //A pointer to the data for the Chinese  font.

    const typ_CHFont *CHFontdata;
}
tFont;
```

设置一个判断标识符来区分当前字符是字母还是汉字，因此在 grlib.h 中添加汉字标识符：

```
// Indicates that the font data is stored in an uncompressed format.
#define FONT_FMT_UNCOMPRESSED   0x00     //

//!Indicates that the font data is stored using a pixel-based RLE format.
#define FONT_FMT_PIXEL_RLE      0x01

//Indicates that the font data is stored in Chinese format.
#define FONT_CH_STYLE           0x02
```

在 grlib.h 中完成字体结构体的重新定义后，在 StellarisWare\grlib\fonts 中构建汉字库文件。

Example：16x16 的汉字字体 fontCH16.c

```
#include "grlib/grlib.h"

//***********汉字字模数据数组***********//
const unsigned char CHFont16_De[]=
{0x10,0x88,0xE4,0x3B,0x12,0x04,0xF4,0x94,0xF4,0x9F,0x94,0xF4,0x94,0xF4,0x04
,0x00,0x01,0x00,0xFF,0x00,0x42,0x3A,0x02,0x3A,0x42,0x4A,0x52,0x42,0x62,0x0
A,0x32,0x00};
const unsigned char CHFont16_Zhou[]=
{0x00,0xE0,0x00,0x00,0xFF,0x20,0xC0,0x00,0xFE,0x10,0x60,0x80,0x00,0xFF,0x0
0,0x00,0x01,0x80,0x60,0x18,0x07,0x00,0x00,0x00,0x7F,0x00,0x00,0x01,0x00,0xFF
,0x00,0x00};
const unsigned char CHFont16_Yi[]=
{0x40,0x20,0xF0,0x0C,0x03,0x00,0x38,0xC0,0x01,0x0E,0x04,0xE0,0x1C,0x00,0x0
0,0x00,0x00,0x00,0x00,0xFF,0x00,0x40,0x40,0x20,0x10,0x0B,0x04,0x0B,0x10,0x20,0x6
0,0x20,0x00};
const unsigned char CHFont16_Qi[]=
{0x40,0x40,0x4F,0x49,0x49,0xC9,0xCF,0x70,0xC0,0xCF,0x49,0x59,0x69,0x4F,0x0
0,0x00,0x02,0x02,0x7E,0x45,0x45,0x44,0x7C,0x00,0x7C,0x44,0x45,0x45,0x7E,0x0
6,0x02,0x00};

//***********汉字字模结构体数组***********//
typ_CHFont g_CHData16[]=
```

```
{
        {CHFont16_De,"德"},
        {CHFont16_Zhou,"州"},
        {CHFont16_Yi,"仪"},
        {CHFont16_Qi,"器"},
};

const static unsigned char pudata[];

//***********16x16 的汉字字体结构体***********//
const tFont g_sFontCH16=
{
        FONT_CH_STYLE,
        16,
        16,
        16,
        {4},
        pudata,
        g_CHData16
};
```

最后在 grlib.h 中将 g_sFontCH16 声明为外部变量：

```
extern const tFont g_sFontCH16;
```

在 Stellaris GUI 中控件的字符显示最终是调用函数 GrStringDraw 完成的，其在 StellarisWare\grlib\String.c 中进行了具体的实现。因此，在 GrStringDraw 中添加实现汉字显示的具体代码：

```
if(sCon.pFont->ucFormat == FONT_CH_STYLE )
{
    while(*pcString)
    {
       unsigned char CHFontNum,width,i,m,Colum,EndNum;
       width=sCon.pFont->ucMaxWidth;
       if(sCon.pFont->ucHeight%8 == 0)
       {
          Colum = sCon.pFont->ucHeight/8;
          EndNum = 0;
        }
        else
        {
          Colum = (sCon.pFont->ucHeight/8)+1;
          EndNum = sCon.pFont->ucHeight%8;
        }
        for(CHFontNum=0;CHFontNum<sCon.pFont->pusOffset[0];CHFontNum++)
        {
```

```
if((sCon.pFont->CHFontdata[CHFontNum].Index[0]==pcString[0])&&
        (sCon.pFont->CHFontdata[CHFontNum].Index[1]==pcString[1]))
{
  for(m=0; m<Colum; m++)
  {
        for(i=0; i<width; i++)
        {
          if(sCon.pFont->CHFontdata[CHFontNum].Msk[i+m*width]>0)
         {
                lIdx=1;
                lBit=0;
                if(m!=Colum-1 || EndNum==0)
                {
                  while(lBit<9)
                {
                if(sCon.pFont->CHFontdata[CHFontNum].Msk
                   [i+m*width]&lIdx)
                {
                  GrPixelDraw(&sCon,lX+i,lY+lBit+8*m);
                }
                        lIdx=lIdx<<1;
                        lBit++;
                        }
                  }
             }
        else
        {
          if(EndNum!=0)
        {
          while(lBit!=EndNum+1)
         {
            if(sCon.pFont->CHFontdata[CHFontNum].Msk[i+m*width]&lIdx)
             {
                GrPixelDraw(&sCon,lX+i,lY+lBit+8*m);
              }
           lIdx=lIdx<<1;
           lBit++;
           }
          }
         }
        }
     }
  }
}
```

```
        lX=lX+width;
        pcString+=2;
         }
    }
```

在 StellarisWare\grlib 中完成的汉字控件驱动的代码实现如上所示，并根据字体的结构体添加所需的汉字字模文件，将修改后的 grlib 重新进行编译生成新的 grlib.lib。

4.应用实例　　PushButton

为了验证新生成的 grlib.lib 是否能够对 Stellaris GUI 控件汉字显示的支持，对\StellarisWare\boards\dk-lm3s9b96\hello_widget 中的 Pushbutton 进行修改。当点击　显示　控件时（如图 4 所示），会在下方出现　德州仪器　字样，控件字体变为　隐藏　（如图 5 所示）；当点击　隐藏　控件时，图 5 变为图 4。因此验证了 Stellaris GUI 能够支持控件的汉字显示。

图 4　显示　控件

图 5　隐藏　控件

5.总结

目前已实现的 Stellaris GUI 控件汉字显示能够很好地满足工程要求，但是由于汉字库本身容量较大，而 cortex-m3 片上资源有限，并不能实现一次对所有汉字的支持，在不增加片外 Flash 和不使用带字库的显示屏等增加成本的设计方案中，根据客户的实际需求，在 grlib 中添加相应的汉字应该是一种较好的实现方式。

6.参考文档

1. Stellaris Graphics Library User Guide.
2. Stellaris LM3S9B96 Microcontroller Datasheet