# 第一章. Framebuffer应用开发

## 1. Framebuffer编程

### 1.1 LCD Framebuffer操作原理

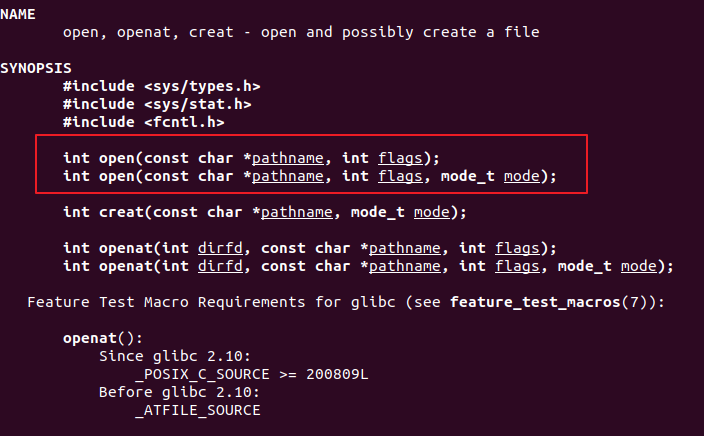
LCD Framebuffer 就是一块显存，在嵌入式系统中，显存是被包含在内存中。LCD Framebuffer里的若干字节（根据驱动程序对LCD控制器的配置而定）表示LCD屏幕中的一个像素点，一一对应整个LCD屏幕。举个例子，LCD屏幕是800\*600的分辨率，即LCD屏幕存在480000个像素点，若每个像素点4个字节表示，那么LCD Framebuffer显存大小为480000\*4=960000字节，即1.92MB。因此我们的内存将会分割至少1.92MB的空间用作显存。具体地址在哪里，这个就是又驱动程序去定，应用程序只需直接使用即可，硬件相关操作已由驱动程序封装好。



如上图，我们只需要往Framebuffer中填入不同的值，驱动程序和硬件控制器就会把这些数据传输到对应LCD屏幕上的像素点，从而显示不同的颜色。由此可知，我们应用程序只需要针对Framebuffer操作即可，其他交给驱动程序和硬件。

### 1.2 Framebuffer API接口

#### 1.2.1 open系统调用



头文件：#include <sys/types.h>，#include <sys/stat.h>，#include <fcntl.h>

函数原型：

int open(const char \*pathname, int flags);

int open(const char \*pathname, int flags, mode\_t mode);

函数说明：

pathname 表示打开文件的路径；

Flags表示打开文件的方式，常用的有以下6种，

①：O\_RDWR表示可读可写方式打开;

②：O\_RDONLY表示只读方式打开;

③：O\_WRONLY表示只写方式打开;

④：O\_APPEND 表示如果这个文件中本来是有内容的，则新写入的内容会接续到原来内容的后面;

⑤：O\_TRUNC表示如果这个文件中本来是有内容的，则原来的内容会被丢弃，截断；

⑥： O\_CREAT表示当前打开文件不存在，我们创建它并打开它，通常与O\_EXCL结合使用，当没有文件时创建文件，有这个文件时会报错提醒我们；

Mode表示创建文件的权限，只有在flags中使用了O\_CREAT时才有效，否则忽略。

返回值：打开成功返回文件描述符，失败将返回-1。

#### 1.2.2 ioctl系统调用



头文件：#include <sys/ioctl.h>

函数原型：

int ioctl(int fd, unsigned long request, ...);

函数说明：

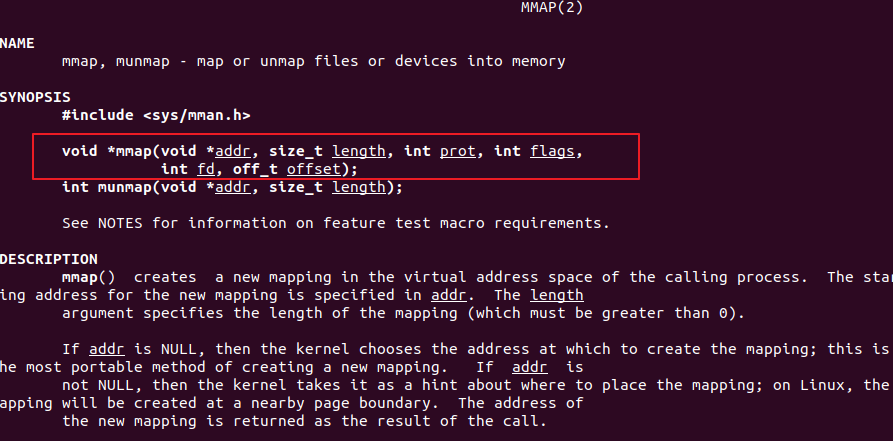
fd 表示文件描述符；

request表示与驱动程序交互的命令，用不同的命令控制驱动程序输出我们需要的数据；

… 表示可变参数arg，根据request命令，设备驱动程序返回输出的数据。

返回值：打开成功返回文件描述符，失败将返回-1。

#### 1.2.3 mmap系统调用



头文件：#include <sys/mman.h>

函数原型：

void \*mmap(void \*addr, size\_t length, int prot, int flags,int fd, off\_t offset);

函数说明：

addr表示指定映射的內存起始地址，通常设为 NULL表示让系统自动选定地址，并在成功映射后返回该地址；

length表示将文件中多大的内容映射到内存中；

prot 表示映射区域的保护方式，可以为以下4种方式的组合

①PROT\_EXEC 映射区域可被执行

②PROT\_READ 映射区域可被读写

③PROT\_WRITE 映射区域可被写入

④PROT\_NONE 映射区域不能存取

Flags 表示影响映射区域的不同特性，常用的有以下两种

①MAP\_SHARED 表示对映射区域写入的数据会复制回文件内，原来的文件会改变。

②MAP\_PRIVATE 表示对映射区域的操作会产生一个映射文件的复制，对此区域的任何修改都不会写回原来的文件内容中。

返回值：若成功映射，将返回指向映射的区域的指针，失败将返回-1。

### 1.3 在LCD上描点操作

#### 1.3.1 在LCD上显示点阵理论基础



如上图，当我们需要显示一个字母‘A’时，是通过判断点阵的每一个位数值状态，来填充颜色，达到显示字符效果。其中‘1’表示一种颜色，‘0’表示填充另一种颜色。上图的是8\*16的点阵，我们也可以用其他不同大小点阵，只要有这个点阵，我们就可以在LCD上面描点，达到显示字符的效果。

#### 1.3.2 获取fb\_var\_screeninfo结构体

在用点阵显示字符之前，我们需要先从设备fb0中获取相关的LCD信息，下图截取我们将用到的fb\_info结构体部分内容。



通过系统调用ioctl，获取xres(x方向总像素点)，yres（y方向总像素点），bits\_per\_pixel（每个像素点占据的位数），根据获取的三个资源，外加点阵，根据这四个资源，我们就可以显示一个字符。

程序文件：show\_ascii.c

4718 fd\_fb = open("/dev/fb0", O\_RDWR);

4719 if (fd\_fb < 0)

4720 {

4721 printf("can't open /dev/fb0\n");

4722 return -1;

4723 }

4724 if (ioctl(fd\_fb, FBIOGET\_VSCREENINFO, &var))

4725 {

4726 printf("can't get var\n");

4727 return -1;

4728 }

先打开LCD设备（fb0），获得文件描述符，再通过ioctl获取fb\_var\_screeninfo信息并保存在var变量，后续只需访问var这个结构体，就可以获得xres(x方向总像素点)，yres（y方向总像素点），bits\_per\_pixel（每个像素点占据的位数）这三个关于fb0的资源。

#### 1.3.3 根据fb\_var\_screeninfo计算变量

fb\_var\_screeninfo已保存在var结构体变量中，接着来访问var结构体变量即可

根据xres与bits\_per\_pixel算出每行像素点所占据的字节数

程序文件：show\_ascii.c

4730 line\_width = var.xres \* var.bits\_per\_pixel / 8;

根据bits\_per\_pixel算出每个像素点所占据的字节数

程序文件：show\_ascii.c

4731 pixel\_width = var.bits\_per\_pixel / 8;

根据xres，yres，bits\_per\_pixel算出全部像素点所占据的字节总和

程序文件：show\_ascii.c

4732 screen\_size = var.xres \* var.yres \* var.bits\_per\_pixel / 8;

#### 1.3.4 使用mmap系统调用，映射内存

程序文件：show\_ascii.c

4733 fbmem = (unsigned char \*)mmap(NULL , screen\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd\_fb, 0);

4734 if (fbmem == (unsigned char \*)-1)

4735 {

4736 printf("can't mmap\n");

4737 return -1;

4738 }

4739

4740 /\* 清屏: 全部设为黑色 \*/

4741 memset(fbmem, 0, screen\_size);

调用mmap将显存映射在内存中，以可读可写（PROT\_READ | PROT\_WRITE）及内存回写（MAP\_SHARED）的方式映射，从而获得一个指向映射在内存空间的首地址fbmem，后续操作就是在这个首地址的基础上计算各种不同的偏移量，填充颜色值。

#### 1.3.5 描点函数编写

程序文件：show\_ascii.c

4641 void lcd\_put\_pixel(int x, int y, unsigned int color)

描点函数有3个参数，x坐标，y坐标，像素点颜色值。

程序文件：show\_ascii.c

4643 unsigned char \*pen\_8 = fbmem+y\*line\_width+x\*pixel\_width;

4644 unsigned short \*pen\_16;

4645 unsigned int \*pen\_32;

4646

4647 unsigned int red, green, blue;

4648

4649 pen\_16 = (unsigned short \*)pen\_8;

4650 pen\_32 = (unsigned int \*)pen\_8;

在此处函数参数x与y表示的是像素点的坐标，而单个像素点所占据的显存大小可能会有不同的情况出现，如1字节表示一个像素点，2字节表示一个像素点，4字节表示一个像素点等，为了更多的兼容不同的情况，因此申请3个指针，pen\_8指向的是占据1个字节的像素点空间, pen\_16指向的是占据2个字节的像素点空间，pen\_32指向的是占据4个字节的像素点空间。

fbmem是系统调用mmap返回的显存首地址，根据fbmem计算填充颜色的内存空间。

当像素点占据1个字节空间时

对应描点地址= fbmem+Y \* 一行所占据的字节数 + x \* 每个像素点所占据的字节数

程序文件：show\_ascii.c

4652 switch (var.bits\_per\_pixel)

4653 {

4654 case 8:

4655 {

4656 \*pen\_8 = color;

4657 break;

4658 }

4659 case 16:

4660 {

4661 /\* 565 \*/

4662 red = (color >> 16) & 0xff;

4663 green = (color >> 8) & 0xff;

4664 blue = (color >> 0) & 0xff;

4665 color = ((red >> 3) << 11) | ((green >> 2) << 5) | (blue >> 3);

4666 \*pen\_16 = color;

4667 break;

4668 }

4669 case 32:

4670 {

4671 \*pen\_32 = color;

4672 break;

4673 }

4674 default:

4675 {

4676 printf("can't surport %dbpp\n", var.bits\_per\_pixel);

4677 break;

4678 }

4679 }

4680 }

根据设备fb0实际的bits\_per\_pixel值，选择对应的pen（pen\_8，pen\_16，pen\_32其中一个），最后把color颜色变量传入选择的pen中。

### 1.4 在LCD上使用点阵写字

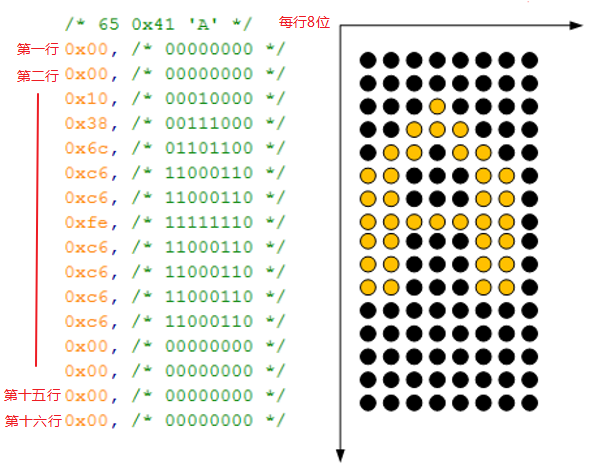
#### 1.4.1 在LCD上显示英文字母

①找出英文字母在点阵数组中的地址，c所代表的是一个英文字母（ASCII值）。

程序文件：show\_ascii.c

4693 unsigned char \*dots = (unsigned char \*)&fontdata\_8x16[c\*16];

②根据获得的英文字母点阵，每一位依次判断，描点，‘1’表示白色，‘0’表示黑色。



根据上图，我们分析下如何利用点阵在LCD上显示一个英文字母，因为有十六行，所以首先要有一个循环16次的大循环，然后每一行里有8位，那么在每一个大循环里也需要一个循环8次的小循环，小循环里的判断单行的描点情况，如果是1，就填充白色，如果是0就填充黑色，如此一来，就可以显示出黑色底，白色轮廓的英文字母。

程序文件：show\_ascii.c

4697 for (i = 0; i < 16; i++)

4698 {

4699 byte = dots[i];

4700 for (b = 7; b >= 0; b--)

4701 {

4702 if (byte & (1<<b))

4703 {

4704 /\* show \*/

4705 lcd\_put\_pixel(x+7-b, y+i, 0xffffff); /\* 白 \*/

4706 }

4707 else

4708 {

4709 /\* hide \*/

4710 lcd\_put\_pixel(x+7-b, y+i, 0); /\* 黑 \*/

4711 }

4712 }

4713 }

③调用我们编写的lcd\_put\_ascii函数

程序文件：show\_ascii.c

4743 lcd\_put\_ascii(var.xres/2, var.yres/2, 'A'); /\*在屏幕中间显示8\*16的字母A\*/

④编译c文件show\_ascii.c

编译命令：arm-linux-gnueabihf-gcc -o show\_ascii show\_ascii.c

⑤将编译出来的show\_ascii传输到开发板，并进入show\_ascii的目录下

执行命令：./show\_ascii

如果实验成功，我们将看到屏幕中间会显示出一个白色的字母‘A’。

#### 1.4.2 在LCD上显示汉字



与显示英文字母有点不同，因为汉字的点阵我们是需要通过汉字库提取出来，并没有直接提供点阵数组，因此我们程序开头需要打开汉字库文件(HZK16)，然后再找到相应的位置，提取出汉字的点阵，最后再按显示英文字母一样显示它，不过这个汉字是16\*16的。

1. 打开汉字库文件

程序文件：show\_font.c

4760 fd\_hzk16 = open("HZK16", O\_RDONLY);

② 获取汉字库文件的属性，存在hzk\_stat结构体变量中

程序文件：show\_font.c

4793 if(fstat(fd\_hzk16, &hzk\_stat))

此处主要是用知道该文件的大小，因为后面mmap时需要知道映射的文件大小。

③使用mmap系统调用

程序文件：show\_font.c

4798 hzkmem = (unsigned char \*)mmap(NULL , hzk\_stat.st\_size, PROT\_READ, MAP\_SHARED, fd\_hzk16, 0);

hzkmem与fbmem类似，也是一个指向映射内存的指针，但是它是指向汉字库，方便

后续计算汉字点阵偏移位置用。

④使用汉字库，调出点阵显示汉字

HZK16 字库是符合GB2312标准的16×16点阵字库HZK16的编码，每个字需要32个字节的点阵来表示，例如我们将要显示的‘中’字，编码是D6D0，难道就是2个字节表示吗？不是说32字节吗？D6D0编码是一个类似于索引码，D6是区码，D0是位码，先要找到D6-A1才是真正区，在D6-A1区里找到D0-A1的真正位置，这才是‘中’字点阵的起始位置（减去A1是为了兼容ascii），每一个区有94个汉字。

程序文件：show\_font.c

4734 unsigned int area = str[0] - 0xA1;

4735 unsigned int where = str[1] - 0xA1;

4736 unsigned char \*dots = hzkmem + (area \* 94 + where)\*32;



上图是汉字点阵排布的示意图，总共有十六行，因此需要一个循环16次的大循环，考虑到一行有两个字节，我们大循环中加入一个循环2次的循环用于区分是哪个字节，最后判断当前字节的每一位，如果为 ‘1’描白色，如果为‘0’描黑色

程序文件：show\_font.c

4740 for (i = 0; i < 16; i++)

4741 for (j = 0; j < 2; j++)

4742 {

4743 byte = dots[i\*2 + j];

4744 for (b = 7; b >=0; b--)

4745 {

4746 if (byte & (1<<b))

4747 {

4748 /\* show \*/

4749 lcd\_put\_pixel(x+j\*8+7-b, y+i, 0xffffff); /\* 白 \*/

4750 }

4751 else

4752 {

4753 /\* hide \*/

4754 lcd\_put\_pixel(x+j\*8+7-b, y+i, 0); /\* 黑 \*/

4755 }

4756 }

4757 }

⑤调用我们编写的lcd\_put\_chinese函数

程序文件：show\_font.c

4810 printf("chinese code: %02x %02x\n", str[0], str[1]);

4811 lcd\_put\_chinese(var.xres/2 + 8, var.yres/2, str);

⑥编译c文件show\_font.c

编译命令：arm-linux-gnueabihf-gcc -o show\_font show\_font.c

注：使用此命令HZK16文件必须与show\_font.C在同一目录下。

⑦将编译出来的show\_font传输到开发板，并进入show\_font的目录下

执行命令：./show\_font

如果实验成功，我们将看到屏幕中间会显示出一个白色的字母‘A’与汉字‘中’，同时在串口打印信息中看到‘中’对应的编码。

chinese code: d6 d0

### 1.5 搭建freetype相关环境

#### 1.5.1 交叉编译freetype，并安装

①解压freetype源文件

tar xjf freetype-2.4.10.tar.bz2

②进入解压后的freetype-2.4.10目录

cd freetype-2.4.10

③配置freetype-2.4.10

./configure --host=arm-linux-gnueabihf --prefix=/home/book/100ask\_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86\_64\_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/usr/

④建个目录，避免后面安装出错提示缺少这个internal目录

mkdir /home/book/100ask\_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86\_64\_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/usr/include/freetype2/freetype/internal -p

④编译

make

⑤安装

make install

⑥移动freetype头文件，避免以后编译总是需要指定头文件路径

mv /home/book/100ask\_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86\_64\_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/usr/include/freetype2/freetype /home/book/100ask\_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86\_64\_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/usr/include/

#### 1.5.2 freetype库，头文件移植至开发板

由于100ask开发板已经有freetype相关的库和头文件，因此不需要移植，如果开发板没有freetype库和头文件就需要按以下方法移植

/home/book/100ask\_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86\_64\_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/usr/include/\* 复制到开发板的头文件目录中

/home/book/100ask\_imx6ull-sdk/ToolChain/gcc-linaro-6.2.1-2016.11-x86\_64\_arm-linux-gnueabihf/arm-linux-gnueabihf/libc/usr/lib/\*so\* 复制到开发板的库文件目录中

注：链接文件需要保持它的链接属性（即加-d选项）。

### 1.6 使用freetype

#### 1.5.1 矢量字体引入

点阵显示英文字母，汉字时，大小固定，如果放大会有锯齿出现，为了解决这个问题，引用矢量字体。

矢量字体形成分三步，若干的关键点，数学曲线（贝塞尔曲线），填充颜色组合而成。

①假设A字母的关键点如图中的黄色圈圈，确定关键点。



②用数学曲线将关键点都连接起来，成为封闭的曲线。



③最后把封闭空间填满颜色，就显示出一个A字母。



如果需要放大或者缩小字体，关键点的相对位置是不变的，跟进放大比例放大或缩小，但是相对位置不变，好像分数中的1/2 和 2/4，比例是不变的，但是值却大了，类似这个味道。

#### 1.5.2 Freetype理论介绍

开源的Freetype字体引擎库它提供统一的接口来访问多种字体格式文件，从而实现矢量字体显示。我们只需要移植这个字体引擎，调用对应的API接口，提供字体关键点，就可以让freetype库帮我们实现闭合曲线，填充颜色，达到显示矢量字体的目的。

关键点（glyph）存在字体文件中，Windows使用的字体文件在FONTS目录下，扩展名为TTF的都是矢量字库，本次使用实验使用的是新宋字体simsun.ttc。



字体文件结构如上图

Charmaps表示字符映射表，字体文件可能支持哪一些编码，GBK，UNICODE，BIG5还是别的编码，如果字体文件支持该编码，跟进编码，通过charmap，找到对应的glyph，一般而言都支持UNICODE码。

有了以上基础，我们想象一个文字的显示过程

①给定一个文字吗‘A’（0x41），‘中’（GBK，UNICODE ,BIG5）可以确定它的编码值；

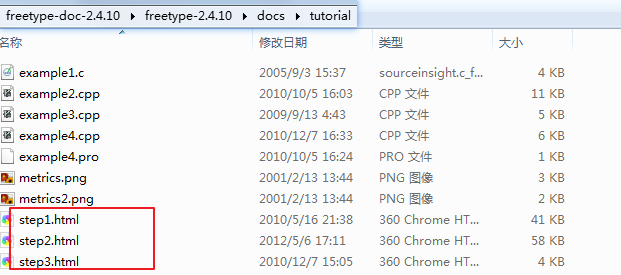
②跟进编码值，从枝头文件中通过charmap找到对应的关键点（glyph）；

③设置字体大；

④用某些函数把关键点（glyph）缩放为我们设置的字体大小；

⑤转换为位图点阵

⑥在LCD上显示出来



如上图，参照step1，step2，step3里的内容，可以学习如何使用freetype库，大致总结下，为如下步骤。

①初始化：FT\_InitFreetype

②加载（打开）字体Face：FT\_New\_Face

③设置字体大小：FT\_Set\_Char\_Sizes 或 FT\_Set\_Pixel\_Sizes

④选择charmap：FT\_Select\_Charmap

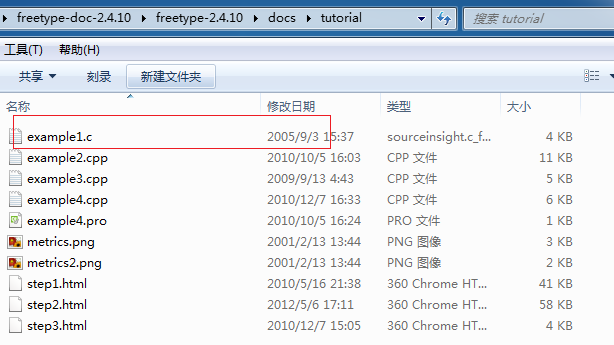
⑤根据编码值charcode找到glyph : glyph\_index = FT\_Get\_Char\_Index（face，charcode）

⑥根据glyph\_index取出glyph：FT\_Load\_Glyph（face，glyph\_index）

⑦转为位图：FT\_Render\_Glyph

⑧移动或旋转:FT\_Set\_Transform

#### 1.5.2 在LCD上显示一个矢量字体



我们可以参考上图位置的c程序，编写程序。

①初始化freetype库

程序文件：freetype\_show\_font.c

4872 error = FT\_Init\_FreeType( &library ); /\* initialize library \*/

②用freetype库中的FT\_New\_Face函数创建一个face字体文件对象，保存在&face中

程序文件：freetype\_show\_font.c

4875 error = FT\_New\_Face( library, argv[1], 0, &face ); /\* create face object \*/

③提取face对象中的glyph，即关键点集

程序文件：freetype\_show\_font.c

4877 slot = face->glyph;

④设置像素点大小，24\*24

程序文件：freetype\_show\_font.c

4879 FT\_Set\_Pixel\_Sizes(face, 24, 0);

⑤确定坐标

目前我们前面所用的都是LCD的坐标系对应的x与y坐标，然后再freetype上却是使用的笛卡尔坐标系，因此我们还需要转换x与y坐标



由上图我们可知，LCD坐标系与笛卡尔坐标系的x坐标是没有变化的，只是在y坐标上，会变成纵向高度减去LCD坐标系中的y，即V-y。



我们将要显示的是‘繁’字，根据上图可知，先计算在lcd坐标系的情况下‘繁’字

的左下角的x坐标与y坐标，因为在笛卡尔坐标中左下角为字符的原点，‘A’是的左上角为整个屏幕的中心点，即（xres/2，yres/2）。

lcd\_x = var.xres/2 + 8 + 16；lcd\_y = var.yres/2 + 16

则笛卡尔座标系:x = lcd\_x = var.xres/2 + 8 + 16 ； y = var.yres - lcd\_y = var.yres/2 – 16

单位是1/64像素，所以需要乘以64

程序文件：freetype\_show\_font.c

4888 pen.x = (var.xres/2 + 8 + 16) \* 64;

4889 pen.y = (var.yres/2 - 16) \* 64;

4890

4891 /\* set transformation \*/

4892 FT\_Set\_Transform( face, 0, &pen);

⑥找到glyph的位置，然后取出，并转换为位图

4895 error = FT\_Load\_Char( face, chinese\_str[0], FT\_LOAD\_RENDER );

4896 if (error)

4897 {

4898 printf("FT\_Load\_Char error\n");

4899 return -1;

4900 }



FT\_Load\_Char函数调用替代了上图这3步。

最后把转换出来的位图打印出来，也是参考example1.c编写

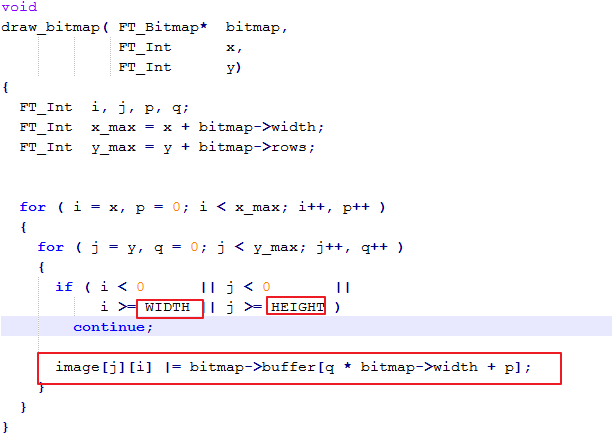
程序文件：freetype\_show\_font.c

4902 draw\_bitmap( &slot->bitmap,

4903 slot->bitmap\_left,

4904 var.yres - slot->bitmap\_top);

程序文件：example1.c



修改上图3处位置

Width宽度：因为在LCD上显示，宽度自然就是x方向的像素点数，var.xres；

Height高度：因为在LCD上显示，高度自然就是y方向的像素点数，var.yres；

用点阵实验中的的描点函数lcd\_put\_pixel替代image数组

lcd\_put\_pixel(i, j, bitmap->buffer[q \* bitmap->width + p]);

⑥编译C程序文件freetype\_show\_font.c

编译命令：arm-linux-gnueabihf-gcc -finput-charset=GBK -fexec-charset=GBK -o freetype\_show\_font freetype\_show\_font.c -lfreetype -lm

⑦将编译好的freetype\_show\_font的文件与simsun.ttc字体文件拷贝至开发板，simsun.ttc字体文件放在freetype\_show\_font执行文件的上一层目录下，执行以下命令。

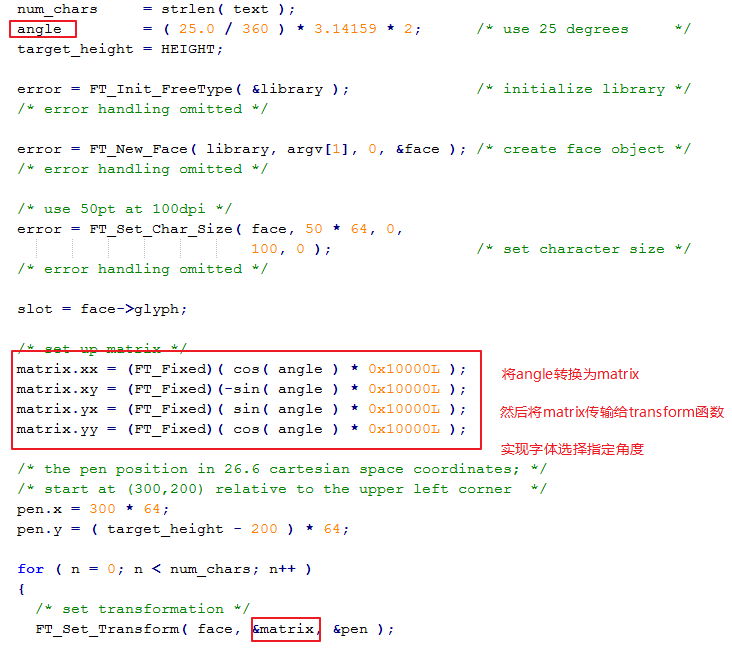
执行命令：./freetype\_show\_font ../simsun.ttc

如果实验成功，我们将看到屏幕中间会比之前实验多出一个蓝色的‘繁’字。

#### 1.5.3 在LCD上令矢量字体旋转某个角度

在实现显示一个矢量字体后，我们可以添加让该字旋转某个角度的功能。

我们根据输入的第二个参数，判断其旋转角度，主要代码还是参照example1.c



根据上图，增加旋转角度功能，旋转的角度由执行命令的第二个参数指定。

程序文件：freetype\_show\_font\_angle.c

4894 angle = ( 1.0 \* strtoul(argv[2], NULL, 0) / 360 ) \* 3.14159 \* 2; /\* use 25 degrees \*/

4895 /\* set up matrix \*/

4896 matrix.xx = (FT\_Fixed)( cos( angle ) \* 0x10000L );

4897 matrix.xy = (FT\_Fixed)(-sin( angle ) \* 0x10000L );

4898 matrix.yx = (FT\_Fixed)( sin( angle ) \* 0x10000L );

4899 matrix.yy = (FT\_Fixed)( cos( angle ) \* 0x10000L );

4900

4901 /\* set transformation \*/

4902 FT\_Set\_Transform( face, &matrix, &pen);

最后编译，在开发板上运行

编译命令如下：

编译命令：arm-linux-gnueabihf-gcc -finput-charset=GBK -fexec-charset=GBK -o freetype\_show\_font\_angle freetype\_show\_font\_angle.c -lfreetype -lm

编译出的文件名为freetype\_show\_font\_angle，将文件拷贝至开发板

在含有该文件的目录下执行以下命令，以下命令正确执行前提是执行文件freetype\_show\_font在此目录，而且字体文件simsun.ttc，在上一级目录：

执行命令：./freetype\_show\_font\_angle ../simsun.ttc 90

如果实验成功，我们将看到屏幕中间的蓝色‘繁’字，旋转了90度。