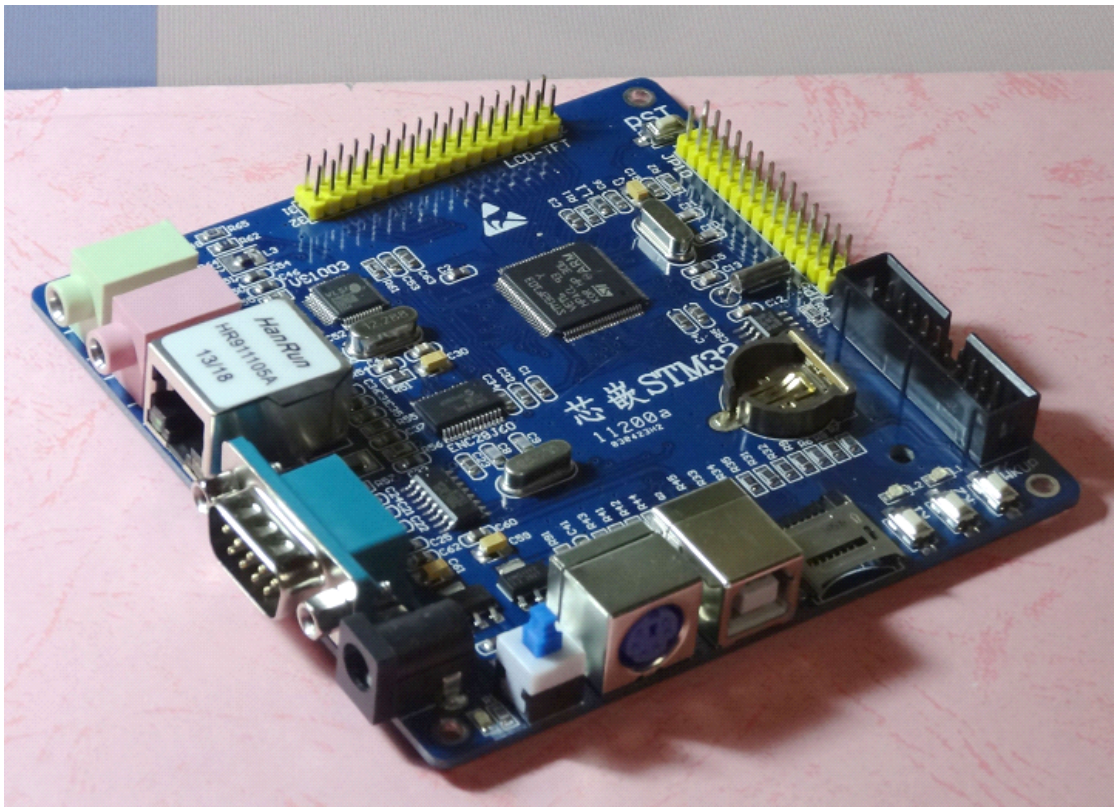
有梦想，
哪里都是你的舞台。
芯嵌stm32开发板

STM32 提高系列教程

基于 STM32 的 uC/GUI 移植手册（无 OS）

Revision V1.0

(2013-08-05)



版权声明

本手册版权归属福州芯嵌电子工作室(以下简称“芯嵌”)所有，并保留一切权力。非经芯嵌同意（书面形式），任何单位及个人不得擅自摘录本手册部分或全部内容，违者（我们）公司将追究其法律责任。

内容提要

本手册较为详细地介绍如何基于STM32系列处理器进行uC/GUI界面的设计与编程（无操作系统os移植），使得STM32初学者对液晶屏的编程不再停留在简单的字符显示，而是更进一步考虑如何DIY自己的图形界面。

本手册从uC/GUI的来源简介开始，step by step，循序渐进，讲解从零开始如何进行uC/GUI移植。读者可以自己配置、移植、裁剪相关内容，编写简单控制程序，并最终实现自己DIY的界面。

本手册并非高手所写，相反，它非常适合刚刚接触STM32的初学者学习 —— 只要读者具备STM32最基本的编码知识（比如IO口点灯控制和LCD初始化等），我们就有足够的信心来协助您一起完成uC/GUI移植。同时，它也可作为嵌入式培训教材参考。

前 言

从芯达 STM32 初级版到芯嵌 STM32 升级版，笔者很早以前就想写一份提高型的文档，一方面可以为芯嵌 STM32 开发板提供配套的教材，同时也为广大嵌入式爱好者提供更为方便的 STM32 学习参考资料。《uC/GUI 中文手册》有 400 多页，加之 STM32 的理解需要时间，初学者真正阅读并完成移植得猴年马月？这也是笔者撰写本文档的原因之一。

本手册硬件平台是芯嵌 STM32 开发板，软件平台为 MDK3.8 版本，使用 ST 官方固件库 V3.5 版以及 uC/GUI 源码 3.90 版。详情参考芯嵌 stm32 开发板光盘，也可在芯嵌 stm32 官网下载获取。后续会继续推出一系列提高型的教程，欢迎大家在我们论坛以及 QQ 群讨论交流，一起进步。这里建议，大家提出的问题，越详细具体越好，这样越有利于其他网友帮助回答问题。

文档撰写时间仓促，加之水平有限，难免会出现一些不足之处，恳请广大网友提出宝贵的意见。

最后，感谢芯嵌客户的大力支持，没有你们，芯嵌 stm32 不可能一帆风顺！我们也将尽最大努力，做好技术支持服务！祝大家学有所成！

芯嵌（福州）培训中心

2013-08-05

手册的约定与更新

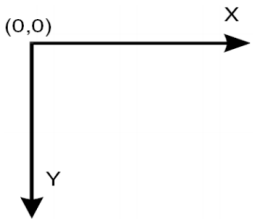
约定 1

本手册首发网站为芯嵌 stm32 官网 www.51stm32.com，建议读者在官网论坛学习讨论。其所有源代码均可在光盘或者官网下载、更新，不再另行通知。

约定 2

本文档使用芯嵌 stm32 开发板，及配套 2.8 寸液晶屏。

由于 uC/GUI 的 API 中向用户程序提供的文本和绘图函数能够在任何指定像素上写或绘制。因此需要对 LCD 坐标做一个约定。本约定后面不再强调。



芯嵌 stm32 配套屏，朝上（面朝读者），上面是字符“2.8 TFT（320*240）”，下面是插针(座)。此时，左上角是坐标原点（0, 0）。X 坐标，表示从左上角开始，从左到右的列（240 列），Y 坐标，表示从左上角开始，从上到下的行（320 行）。

约定 3

虽然“文件夹”与“目录”是两个概念，但本文档中所提到的“文件夹”与“目录”视为同一个意思。

芯嵌 STM32 用户手册修订记录

日期	版本	修改章节	修改描述	作者
2013-08-05	1.00	全部	创建	51smt32

目 录

第一章 准备工作

为了更方便进行 uC/GUI 图形界面移植与编码，请务必认真对待如下工作。

1.1 uC/GUI 源码获取

本文档使用 uC/GUI 3.90 版本源码，请读者自行获取。也可以通过芯嵌 stm32 官网获取，链接:

http://www.51stm32.com/forum.php?mod=viewthread&tid=194

请务必使用 3.90 版本的源码，不同版本的源码，可能会导致您移植失败。

STM32 固件库 3.5 版本源码，在这里下载:

http://www.51stm32.com/forum.php?mod=viewthread&tid=57&extra=page%3D1

1.2 集成开发环境 MDK

uC/GUI 开发可以在较多开发环境下进行。本处基于 STM32 程序设计与开发，因此选择 MDK 软件，版本 3.8。该软件源于 Keil 公司，目前与 ARM 公司合并，取名 RealView MDK。实际上与 Keil 界面几乎一样。对 51 转型 STM32 的初学者非常适用。

MDK 软件可以在芯嵌 stm32 光盘中获取，也可通过 keil 官网下载，或在芯嵌官网中下载，地址:

http://www.51stm32.com/forum.php?mod=viewthread&tid=192

下载后如何安装、序列号授权等方法，请参考芯嵌 stm32 入门系列教材之五、之六，关于 MDK 的安装和使用。

1.3 硬件平台

硬件平台: 一块 STM32 开发板 + 任意尺寸液晶屏。本文档基于芯嵌 STM32 开发板 V1.x 版本进行移植，读者可以自己 DIY 一个 STM32 最小系统板，或者购买一块带有液晶触摸屏的 STM32 开发板即可。其他开发板修改代码后，也同样可以适用。注意，由于 uC/GUI 涉及到 LCD 驱动，请务必对自己手里开发板的 TFT LCD 接口硬件电路有一定了解。芯嵌 stm32 开发板原理图，详见光盘，或者这里下载:

http://www.51stm32.com/forum.php?mod=viewthread&tid=73&extra=page%3D1

1.4 J-Link 仿真器

程序仿真所需：一台 jlink 仿真器，或者串口 ISP 下载模块也可。建议使用 jlink 仿真器调试程序较为方便。目前一般采用 JLINK V8 版本，可仿真 ARM 内核的几乎所有的 CPU。如需 JLINK 仿真器驱动程序，请点击下面的链接：

http://www.51stm32.com/forum.php?mod=viewthread&tid=15&extra=page%3D1

第二章 uC/GUI 介绍

大部分初学 STM32 的读者，大都听说过 uC/GUI。可是很多读者却认为这就是操作系统。实际上，真正的 uCOS II 是没有带图形界面的。图形界面的设计，需要我们自己做。大家可能不会想到，手机液晶屏显示，或者其他数码产品上的图形显示，虽然好看，但这里面注入了巨大的代码量。如果每一个产品，都重新写这么庞大的代码来做图形界面的话，那岂不是很不方便？因此，Micrium 公司研发了一款通用的嵌入式用户图像界面系统，这就是 uC/GUI。它的通用性在于，能给任何使用图像 LCD 的应用程序提供独立于（不依赖）处理器和 LCD 控制器之外的有效的图形用户接口，能应用于任何 LCD 控制器和 CPU 的任何尺寸的物理显示或模拟显示中。它能够应用于单一任务环境，也能够应用于多任务环境中。下面，我们就从零开始进入 uC/GUI 界面。

2.1 uC/GUI 特点

《uC/GUI 中文手册》写了很多特性，不是移植的重点，这里就做如下总结。如需下载该 uC/GUI 中文手册，链接在这里：

http://www.51stm32.com/forum.php?mod=viewthread&tid=193

适用于任何 8 位/16 位/32 位 CPU，支持 ANSI C 的任何编译器，适用于任何控制器驱动任何 LCD（单色，灰度，或彩色）。通过配置宏，可支持任何接口任意显示尺寸的液晶屏。可在 LCD 的任何一点上显示字符和画位图，对于显示尺寸和速度提供优化进程，编译时间依赖于采用的优化进程支持虚拟显示。

对字体的说明：uC/GUI 提供了不同种类的字体，使用它们很简单，只要先选择一种字体，然后显示即可，使用方法如下所示：

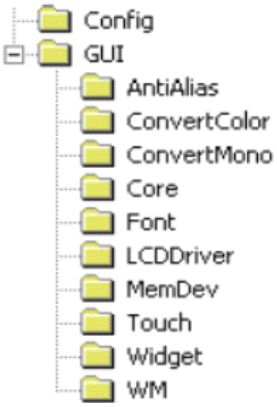
```
GUI_SetFont(&GUI_Font24_1);                    //先设定好字符大小
GUI_DispStringAt("Author: XQ_STM32",60,290); //显示字符
```

该例子，就是在 LCD 坐标（60,290）位置开始，显示字符串“Author: XQ_STM32”。有了 uC/GUI 帮我们写的源码库，我们只要调用这两个函数，就可

以显示了，极为方便。遗憾的是，uC/GUI 目前仍然不支持中文汉字显示。不过，网络上已经出现了网友自己制作的汉字库代码及对应的函数，大家可以参考相关内容。

2.2 uC/GUI 源码结构

在移植 uC/GUI 时，建议将 uC/GUI 源码与读者的应用程序目录分开放置，《uC/GUI 中文手册》中推荐使用两个目录: Config 与 GUI。前者放置配置文件，后者放置源码，具体结构如下:



下面分别对源码中每个目录进行介绍。

目　录	内　　容
Config	配置文件
GUI/AntiAlias	抗锯齿支持 *
GUI/ConvertMono	用于 B/W（黑白两色）及灰度显示的色彩转换程序
GUI/ConvertColor	用于彩色显示的色彩转换的程序
GUI/Core	μC/GUI 内核文件
GUI/Font	字体文件
GUI/LCDDriver	LCD 驱动
GUI/Mendev	存储器件支持 *
GUI/Touch	触摸屏支持 *
GUI/Widget	视窗控件库 *
GUI/WM	视窗管理器 *

注意，该表格中带红色“*”的，是 uC/GUI 的可选项，比如对触摸屏支持的源码就是可选项，对于 uC/GUI 移植入门，可以暂且不管。但是为了保持 uC/GUI 文档的完整性，我们在这里对每个目录都做一个介绍，另外，由于版本不同，可能对应的目录也略有不同，但大同小异。

Config 目录有三个 h 文件，如下:

GUIConf.h: 配置 GUI 移植到不同操作系统的选项。同时液晶屏是否支持触摸的选项也在这里。本文档主要讨论无操作系统的移植。

GUITouchConf.h：配置触摸屏的选项，比如触摸坐标 X 与 Y 的镜像，切换等在这里控制。本文移植 uC/GUI 暂不考虑支持触摸屏。

LCDConf.h：LCD 显示屏的选项文档，包括液晶屏像素（本文档使用芯嵌配套 2.8 屏为 320*240），液晶初始化序列代码等（由于芯嵌 stm32 配套 LCD 例程已经写有对应的 LCD 初始化代码，因此移植时，不用此处代码，详见后续讲解）。

GUI/AntiAlias 目录：包含 9 个 C 文件。该目录源码的功能，在于处理显示的边缘模糊效果，也即抗锯齿和优化 LCD 锯齿。由于液晶是很多个像素点构成，因此，液晶屏上画斜线往往都有锯齿，可以通过优化算法进行美化。

GUI/ConvertMono 目录：描述单色显示的不同模式。

GUI/ConvertColor 目录：包含 14 个.C 文件，涉及调色板模式。uC/GUI 的调色板模式支持 111 模式，222 模式，223，323，332，444，555，565，8666 等相关模式。

GUI/Core 目录：该目录最为关键。它是 uC/GUI 的核心部分。包括：

1) GUI 头文档，GUI 显示各种文本，二进制，十进制，十六进制，字符型文本，字符串，在不同的位置显示二进制，十进制，十六进制，字符型文本，字符串等；GUI 配置各种字体；

2）UI 的 2-D 图像库，GUI 绘图函数，在各种位置绘各种点，线，位图，多边形，长方形，圆等等；

3) GUI 获取函数，获取当前点，线，位图，多边形，长方形，圆，当前字体，当前二进制，十进制，十六进制，字符型文本，字符串等函数；配置 GUI 画笔函数；

4) GUI 支持的鼠标函数；GUI 支持的键盘函数，GUI 支持的触摸屏函数；GUI 配置 LCD 函数等等。

这些函数，在 uC/GUI 系统中都是必须的函数。正是这些函数的组合，使得 uC/GUI 具备复杂而且完备的图像用户接口。而且，这些函数的组合，使得 uC/GUI 能够单独的使用，也能够通过配置文档，移植到各种操作系统中使用。

GUI/Font 目录：uC/GUI 支持的所有字体形式，都在此目录下。

GUI/LCDDriver 目录：该目录下包含很多已完备的 LCD 控制器的驱动程式连同 API 函数。我们可以直接使用或修改这些 API 函数，作为我们自己的底层代码。或者重新新建 C 文件，专门针对自己的 LCD 做驱动文件。本文档采取后者。

GUI/MemDev 目录：有网友指出，该目录用于防止在画交迭图时产生的抖动。另有网友指出，是对存储器件的支持。由于本文档移植的 uC/GUI 3.90 版本，没有该目录，因此无从考证。

GUI/Touch 目录：触摸屏驱动函数所在目录。uC/GUI 3.90 版本，把触摸有关的 C 文件，放入 Core 目录中。

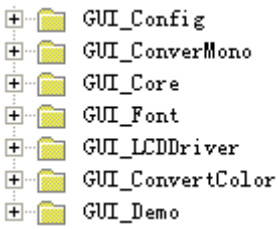
GUI/Widget 目录：包含窗口控件函数。UC/GUI 中窗口控件机制是 uC/GUI 的实现难点，也是应用难点。运用窗口管理和回调机制，运用窗口控件函数，能够任意在 LCD 屏幕上实现类似于 windows 的界面，这样完备的功能在工业自动化控制和触摸屏应用上有着深远而积极的意义。

此目录下的函数主要包括 uC/GUI 的窗口控件，如按钮 BUTTON，校验窗 CHECKBOX，编辑区 EDIT，窗口框 FRAMEWIN，列表 LISTBOX，进度条 PROGBAR，音频按钮 RADIOBUTTON,滚动条 SCROLLERBAR，改变值的灰度条 SLIDER，连同文本框 TEXT 相关的各种函数。

GUI/WM 目录：包括窗口管理函数。uC/GUI 中，窗口管理中的消息传递机制和回调机制，也是 uC/GUI 的实现难点和应用难点。

此目录下的函数主要包括配置、返回、建立背景窗口、父窗口、各种子窗口连同相应的尺寸、窗口句柄，起点 x，y 坐标，窗口宽度，高度，位置等等，还包括改变窗口的大小，连同最关键的窗口的回调函数，窗口重绘函数等等。

以上是对 uC/GUI 源码组织结构做简要介绍，对我们移植来说，本文档移植的思路是，先把 uC/GUI 基本的功能移植成功，感兴趣的读者，可自行钻研扩展功能部分。因此，我们推荐的目录如下：



为了让初学者更好地理解，我们保留了 uC/GUI 最基本的源码，诸如触摸、窗口等扩展源码暂且不谈，不影响移植和显示。但增加了演示部分的 Demo 源码。从目录结构上看，实际上是一样的布局。

第三章 uC/GUI 移植

3.1 uC/GUI 移植思路

前面简单介绍了 uC/GUI。为了更好地理解 uC/GUI 在移植中的位置，笔者特地画了一个层次结构图，如下:

图 3-1 uC/GUI 层次结构图

从该图可以清晰地看出 uC/GUI 移植在整个系统中所处的位置。为了移植这样的图形界面，读者也很容易知道我们应该做哪些工作。下面按从下至上的思路来讲解:

（1）准备好芯嵌 STM32 开发板和液晶屏。

（2）让 STM32 的 CPU 自己工作起来。这个过程，幸好我们有 ST 官方提供的 3.5 版本固件库模板，大家可以下载固件库，并修改编译无错误后，在 main.c 文件中加入 SystemInit();函数，来让 STM32 自己先初始化，进入工作状态。关于 ST 官方固件库模板 V3.5 版，以及修改编译好的模板源码，可以在这里下载:

http://www.51stm32.com/forum.php?mod=viewthread&tid=57&extra=page%3D1

（3）让 LCD 屏也初始化 OK。这就是底层驱动程序要做的。该步骤很关键!请务必把所有关于 LCD 初始化序列的程序，写成一个函数。即 LCD 初始化函数，一般该函数包括对 LCD ID 号读取操作（可省）、LCD 复位操作、寄存器初始化序列操作等。当然，之前需要先写好对 LCD 数据地址的读写操作函数。

对 LCD 初始化函数的命名，可以取名 LCD_Setup ();或 LCD_Config();注意，不能取名 LCD_Init();，因为 uC/GUI 源码内，也有一个函数叫 LCD_Init();，重名会引发错误。写完该函数后，请测试一下，确保能正常驱动 LCD。

（4）uC/GUI 的定制，让它适应自己的 STM32 板。uC/GUI 源码库很强大，可是，大多数情况下，我们不会用到它里面的全部代码。因此，我们要有选择地进行。这就是之前提到的，通过配置文件的宏开关，来打开和关闭 uC/GUI 的某个功能。比如，我们是无操作系统的 uC/GUI，那就关闭下面的选项:

```
#define GUI_OS                    (0)
```

在 uC/GUI 的配置文件中，“0” 表示关闭某项功能，“1” 表示打开某项功能。

（5）阅读并使用 uC/GUI 的常见函数，编写出用户界面。实际上，就是调用 uC/GUI 已经写好的函数，使用这些函数来显示，仅此而已。这个步骤，我们后续有专门的章节讲述。

3.2 uC/GUI 移植全跟踪

本节从零开始，一步一步讲述移植详细过程。先来看看 uC/GUI 移植的工程中，整个源码的组织架构怎么来的。

步骤一：uC/GUI 源码组织

大家下载的 ST 固件库 3.5 版本的源码如下截图：

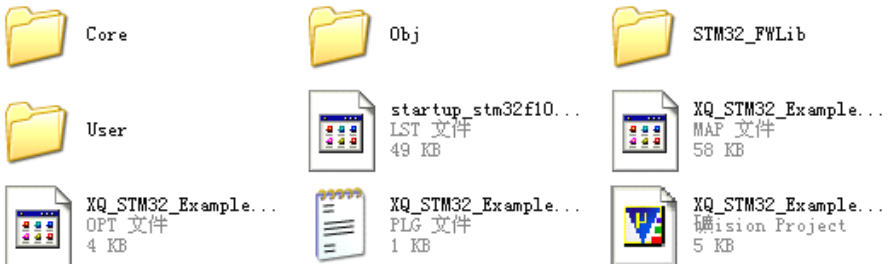


图 3-2　ST 固件库 3.5 版本模板

而 uC/GUI 3.90 版本的源码目录如下：

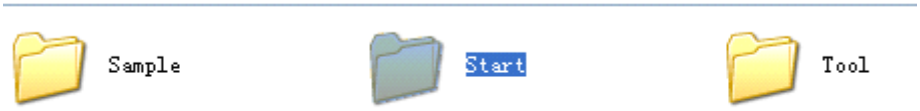


图 3-3　uC/GUI 3.90 版本源码总目录

上面有三个文件夹，Sample 目录中存放了已经写好的用户 DIY 界面的例子，但仅仅是 Demo 而已。Start 目录中才是真正的源码。Tool 目录是在使用 uC/GUI 过程中，可能会用到的工具。如此，可以双加打开 Start 目录，如下所示：

图 3-4　uC/GUI 3.90 版本源码目录

该 Start 目录下，我们能用上的是源码，就是之前提到的 Config 目录和 GUI 目录。请把 Config 目录 copy 到 GUI 目录下（根目录），然后把 GUI 目录 copy

到 STM32 固件库源码根目录下，如下所示：

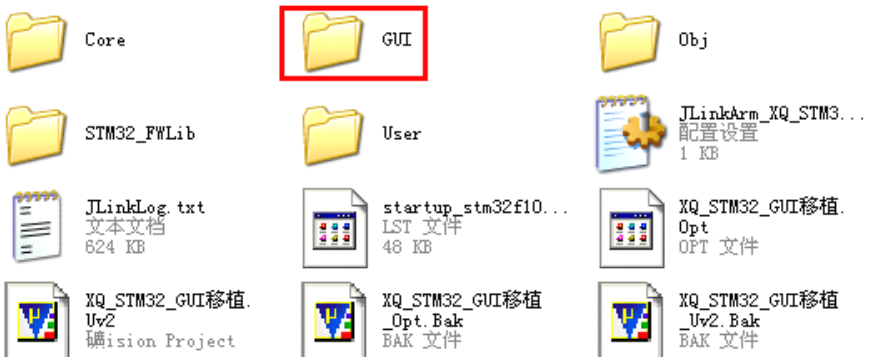


图 3-5　添加完 uC/GUI 源码的目录

对比下，和刚才的 STM32 固件库源码目录相比，是不是只多了一个 GUI？因为我们把所有的关于 uC/GUI 的源码都放在一个目录下了。再次强调，这里的 GUI 目录，包含 Config 文件夹，因为刚才 copy 过来了。

注意！有一个文件需要读者专门添加： GUI_X.c。请在刚才的 uC/GUI 的源码目录下，\Sample\GUI_X 中，复制 GUI_X.c 文件到现在的 GUI\Config 目录中。如果不添加，后续编译会出错。

考虑到最终需要演示，请把 Sample 目录下的 GUIDemo 文件夹也复制过来，同样放在 GUI 目录下。

至此，整个 uC/GUI 移植的源码准备完毕。

步骤二：将源码添加到工程中。

大家将下载的 STM32 固件库模板 3.5 版的工程模板，打开，如下图所示，先编译一下，确认是否编译通过！如果未编译通过，请先查错！

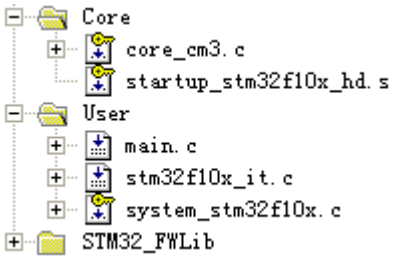


图 3-6 固件库模板工程目录

现在在此基础上，添加 GUI 源码和 Config 配置文件到工程里。

关于如何在 MDK 开发环境中添加 C 文件到工程目录里，请参考芯嵌 stm32 入门教程之七《创建 MDK 工程模板》。该文档图文并茂，详细讲述了从零开始创建一个空的、编译通过的 MDK 模板，也包括添加 C 文件到工程里这部分内容。

这里就简单地文字描述一下：在打开的工程模板界面中，左侧，Project Workspace 这边，有树形的目录结构。对准里面任意一个目录，右键。点击“Manage Components”。在打开的窗口中，有三个栏。在第二个栏添加目录，第三个栏添加 C 文件。OK，详情大家参考芯嵌 stm32 教程之七。

这里强调几点：

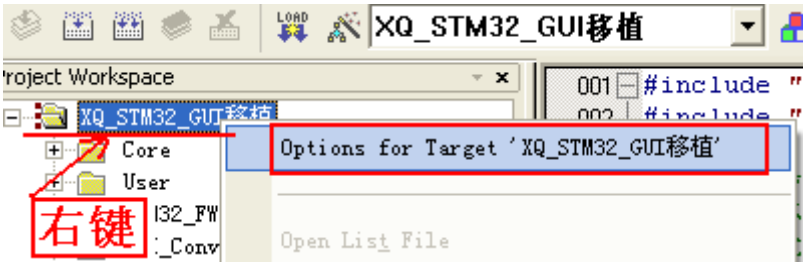
1）添加目录时，只要是 GUI 的源码目录，请务必将目录名前面加上 GUI_，使之与原先的源码目录相区别。后续只要看到“GUI_”前缀，即是 GUI 源码。

2）源码都添加完后，编译肯定不会通过。请不要烦躁。几乎没有人一次性全部通过的。

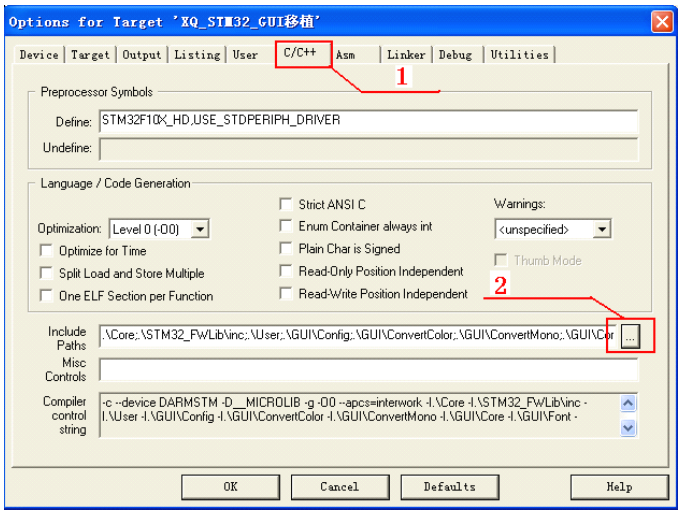
步骤三：添加对应的 h 文件

步骤二中，添加源码默认仅仅是添加 C 文件，并没有添加对应的 h 文件。如何添加 h 文件呢？我们继续图文并茂，分步进行。

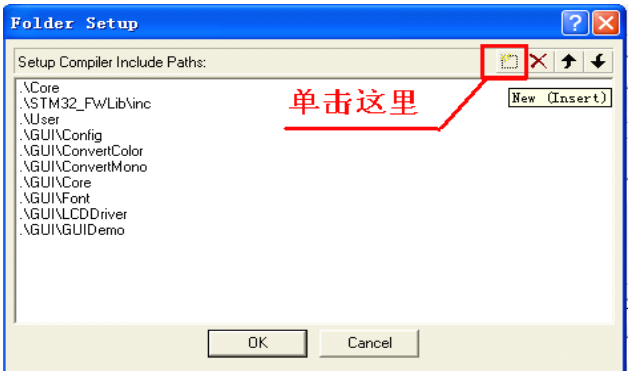
1)在 MDK 界面的左侧，Project Workspace 工作区，对准“XQ_STM32_GUI 移植”右键，选择 Options for Target ‘XQ_STM32_GUI 移植’。



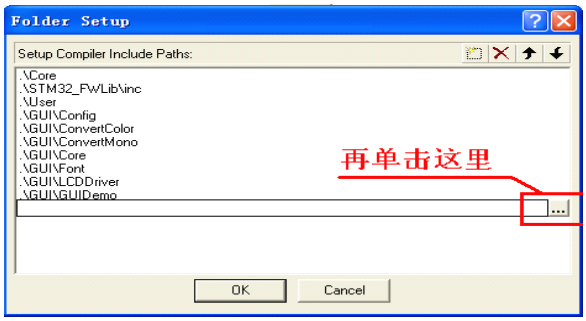
此时会打开如下窗口，请先单击 C/C++选项卡，然后单击下方的红色方框处。



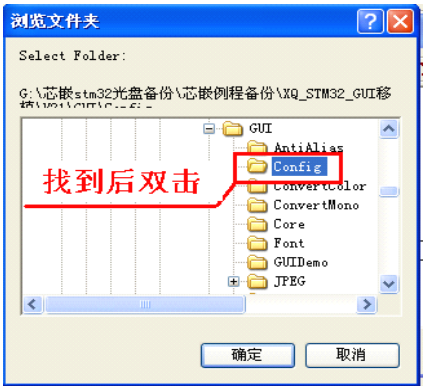
2）此时会打开类似下面的一个窗口。单击红色方框处。此操作是新建一个 h 文件目录的意思。



点击后，就会打开如下窗口：



我们再单击图中右边红色方框按钮。就会打开一个 windows 浏览文件夹的窗口，如下图所示。



在这个窗口中，我们找到 GUI 目录。假设我们现在要添加 GUI\Config 下的 h 文件（实际上，每个文件夹下都有 h 文件，所以要一个文件夹一个文件夹地添加过去）。那么我们就找到 GUI 下的 Config 目录，双击它。就添加完成 Config 目录下的所有 h 文件的路径了。然后点击确定退出。

以上步骤，需要重复多次。因为我们要把用到的所有文件夹目录下的 h 文件路径都添加都工程里。记住，要一个一个添加的，这里不允许批量添加。

这里的“用到的所有文件夹目录”是指哪些呢？我们列出来了：

之前欧版就有的路径：

.\Core;.\STM32_FWLib\inc;.\User;

现在需要你添加的路径：

.\GUI\Config;.\GUI\ConvertColor;.\GUI\ConvertMono;.\GUI\Core;.\GU

I\Font;.\GUI\LCDDriver;.\GUI\GUIDemo

其他就不罗嗦了。现在把创建好的、添加完 GUI 源码的工程截图如下：

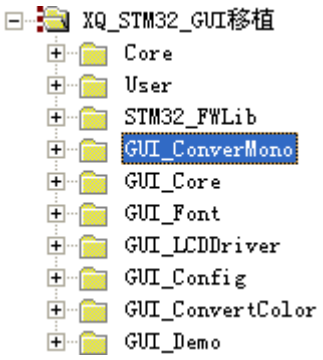


图 3-7 添加 GUI 后的工程目录

```
#include "stm32f10x.h"

int main(void)
{
    while(1)
    {

    }
}
```

图 3-8 空模板的 main 函数截图

左边就是添加好 GUI 源码的工程目录，右边的截图表明，这是一个空的模板。很明显，我们略去了 GUI 的扩展部分。比如触摸 touch 部分，比如窗口部分等等。

步骤四：LCD 驱动编写

我们很清楚，uC/GUI 所有的图形设计操作，最终是需要靠调用底层，控制 LCD 内部寄存器和 GRAM 显示来实现的。因此，我们在修改 uC/GUI 源码之前，要先做好底层 LCD 的初始化和驱动程序，以便后续 uC/GUI 想调用时，能顺利完成操作。

请大家看目录中的 GUI_LCDDriver。这个目录是用来放置 LCD 驱动程序用的，里面有 3 个驱动 C 文件，实际上是都是用来驱动 LCD，并给 GUI 提供接口函数的。很多网友直接把官方的一些 LCD 驱动程序文件 copy 过来，替换掉他们，也是可以的。笔者建议，自己写 LCD 驱动文件。方法是，新建一个 c 文件，写好 LCD 驱动，添加到 GUI_LCDDriver 目录中。然后修改 LCDDummy.c 即可

自己写 LCD 驱动，不难。只要完成以下几个函数即可。

1. LCD 初始化

该步骤主要完成 LCD 寄存器序列的配置初始化。当然，前提是要写好对 LCD 读写操作的函数。芯嵌 stm32 开发板配套的 LCD，使用的是 ILI9341 控制 IC，其初始化过程如下：

由于用到了 FSMC 控制 LCD，因此，我们需要先初始化 FSMC。我们把 FSMC 初始化函数写在了 main 文件中：void FSMC_LCD_Init(void)。如何编写该函数，请参考芯嵌 stm32 入门教程《点亮 LCD》部分。

接下来编写 LCD 读写操作的函数。这里就举一个例子说明：写寄存器。

```
void LCD_Write_Reg(unsigned int LCD_Reg)
{
    *(volatile uint16_t *) (LCD_Reg_Addr) = LCD_Reg;
```

```
}
```

读函数也是类似的写法，不再重复。有了读写操作函数，就可以访问寄存器了。附录中给出 ILI9341 的 LCD 初始化寄存器序列的代码，需要可以参考，我们把 LCD 初始化函数命名为 **LCD_Setup();。此函数名请牢记。**因为 uC/GUI 初始化时，会调用到这个函数，后续会讲解如何把该段初始化代码让 uC/GUI 识别到。

以上函数，随便抓一个 LCD 的驱动，都有。但是**下面这两个函数一定要写好调试通过:**

函数（1）：用某种颜色填充一个像素点;

函数（2）：读取某一个像素点的颜色值;

芯嵌 stm32 开发板移植的时候，把上述 2 个函数写如下:

```
/*************************************************************
 * 填充一个像素，带颜色
 * Xpos 和 Ypos 是坐标点
 * e 是颜色值
 *************************************************************
*/
void lcd_wr_point(unsigned int Xpos, unsigned int Ypos, unsigned int e)
{
      LCD_SetCursor(Xpos, Ypos);
     LCD_WriteRAM_Prepare();          /* Prepare to write GRAM */
      LCD_Write_RAM(e);
      LCD_WriteRAM_Ready();
}
/*************************************************************
 * 读出一个点的颜色
 * a 和 b 是坐标点
 *************************************************************
*/
int lcd_get_point(unsigned int Xpos, unsigned int Ypos)
{
      int temp = 0,temp_R = 0,temp_G = 0,temp_B = 0;

      LCD_SetCursor(Xpos, Ypos);
      LCD_Write_Reg(0X2E);
      LCD_Read_Data();  //读取的第一个数，无效
      temp_R = LCD_Read_Data();
      temp_B = LCD_Read_Data();
      temp_G = temp_R & 0xFF;
      temp_G <<= 8;
      return (((temp_R>>11)<<11)|((temp_G>>10)<<5)|(temp_B>>11));
}
```

以上两个函数都使用了坐标定位函数 LCD_SetCursor(); 请自行编写，并确保以上函数都调试 OK! 可以自行测试一下，比如用函数（1）对 LCD 上某个点设置红色，然后用函数（2）把这个点颜色读出来，在其它点上试下，是不是那个颜色。

再次强调，刚才写的这些函数，请单独新建一个 c 文件，比如取名：LCD_Driver.c（可能会有对应的 h 文件）。添加到工程目录 GUI_LCDDriver 下。注意，不能命名为 Lcd.c 或 lcd.h 文件。因为 uC/GUI 中已经有该文件。

2. 让 uC/GUI 识别到我们写的 LCD 驱动

移植工作到这里，可以说成功了一半。因为整个 uC/GUI 移植，需要写 C 函数的地方，都写 OK 了！现在就让 uC/GUI 识别你刚才写的 C 函数吧！

（1）打开 GUI_LCDDrive\LCDDummy.c，先#include LCD_Driver.h。

（2）找到: #if (LCD_CONTROLLER == -1) && (!defined(WIN32) |
defined(LCD_SIMCONTROLLER))

留下“#if (LCD_CONTROLLER == -1)”后面都删掉。这样，就可以让 uC/GUI 编译执行后续的语句。

（3）找到“void LCD_Lo_SetPixelIndex(int x, int y, int PixelIndex)”，把该函数里所有语句删掉，换成我们自己的函数 lcd_wr_point(); 换完后如下:

```
void LCD_Lo_SetPixelIndex(int x, int y, int PixelIndex) {
    lcd_wr_point(x,y,PixelIndex);  //对应一个坐标上画一个点
}
```

（4）找到“unsigned int LCD_Lo_GetPixelIndex(int x, int y)”，把该函数里所有语句删掉，换成我们自己的函数 lcd_get_point (); 换完后如下:

```
unsigned int LCD_Lo_GetPixelIndex(int x, int y) {
     return lcd_get_point(x, y);
}
```

（5）打开 LCDConf.h 文件，我们发现，里面内容不少！除了配置 LCD 选项之外，还有很多 LCD 寄存器初始化序列也放这里了！因为我们之前已经新建 LCD_Driver.c 文件，把 LCD 所有的初始化都放那边去了，因此，这里都删掉！留下的部分如下:

```
#ifndef LCDCONF_H
#define LCDCONF_H

#define LCD_XSIZE        (240)    /* 根据约定，X 表示 240 列 */
#define LCD_YSIZE        (320)    /* 根据预定，Y 表示 320 行 */

#define LCD_BITSPERPIXEL (16)      /* lcd 颜色深度(灰度)，我们采用 16 级灰度*/
#define LCD_CONTROLLER   (-1)   /*官方对一些具体的 LCD 型号做了注释，而*/
```

```
                                        /*我们并没有使用这些型号的 LCD，因此写-1*/
#define LCD_INIT_CONTROLLER()    LCD_Setup();

#endif /* LCDCONF_H */
```

我们对上面几个语句都做了注释。最后一句：LCD_INIT_CONTROLLER()。这是我们的关键。我们发现，原来在 uC/GUI 中，LCD 的初始化放在这里引用了！请注意，我们之前写的 LCD 初始化的函数名，务必与这里的相同！

步骤五：修改 uC/GUI 的配置文件

修改 uC/GUI 的配置文件，实际上就是要根据自己的移植情况，来裁剪整个 uC/GUI。我们打开 GUIConf.h，修改成如下：

```
#ifndef GUICONF_H
#define GUICONF_H

#define GUI_OS                          (0)   /* 是否支持操作系统，0 表示不支持 */
#define GUI_SUPPORT_TOUCH               (0)   /* 是否支持触摸，0 表示不支持 */
#define GUI_SUPPORT_UNICODE             (1)   /* 是否支持混合的 ASCII/UNICODE 编码，1 表示支持*/

#define GUI_DEFAULT_FONT                &GUI_Font6x8      /* 定义默认的字体大小 */
#define GUI_ALLOC_SIZE                  5000              /* 分配的动态内存空间大小 */

/*********************************************************************
*
*          Configuration of available packages
*/

#define GUI_WINSUPPORT                  0    /* 是否支持窗口功能 */
#define GUI_SUPPORT_MEMDEV              0    /* 是否支持存储器设备*/
#define GUI_SUPPORT_AA                  0    /* 是否支持抗锯齿功能*/

#endif    /* Avoid multiple inclusion */
```

至此，移植完成。请打开 main 文件，找到 main()函数，添加入最基本的 STM32 自身的初始化函数 SystemInit();，然后编译。如果之前的步骤都完成，那么编译可以通过，若未编译通过，请检查是否出错。

第四章 uC/GUI 的使用

移植完成，并不代表就肯定可以显示图形界面。我们需要了解 uC/GUI 系统给我们提供的接口 API，使用它们，我们可以 DIY 出自己想要的图形界面效果!因此，我们还要了解 uC/GUI 中常见的 API。

4.1 uC/GUI 常见 API 介绍

uC/GUI 中常见的 API 很多，我们这里主要介绍关于画图形的接口函数（主要介绍画圆形（包括椭圆），以及画弧形的函数。），以及写字符的函数。

在介绍画图、字符的函数之前，先介绍 GUI_Init();函数。该函数是每个用户界面设计之前，都要先写的。它可以帮初始化整个 GUI，包括 LCD 初始化。

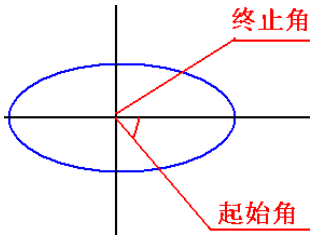
函数（1）：void GUI_DrawCircle(int x0, int y0, int r)

该函数表示在 LCD 上任意坐标位置画一个圆。其使用很简单，(x0,y0)表示坐标（XY 坐标请参考本文档约定 2），r 表示画出来的圆的半径。比如，要在坐标（100,100）处画一个半径为 50 的圆（这里的单位都是“像素”），那么只要调用函数：GUI_DrawCircle(100, 100, 50);即可。

更多的圆的画法（包括实心圆、椭圆），请参考 GUICirc.c 文件。

函数（2）：void GUI_DrawArc(int x0, int y0, int rx, int ry, int a0, int a1)

该函数表示在 LCD 上画出一个圆弧。如果不仔细看函数的实现，很难知道如何使用该函数。实际上，该函数可以理解为，画出来的圆弧，是某个空心椭圆的一部分。(x0, y0)表示这个椭圆的中心，rx 和 ry 表示这个椭圆的 X 坐标和 Y 坐标的半径。难理解的是 a0 和 a1，它们表示的是角度，如下示意图：



其中，a0 表示起始角度，a1 表示终止角度。而画出来的这个圆弧，就是这个椭圆的，从起始角到终止角的这部分圆弧。

函数（3）写字符函数。

为了写一个字符，需考虑：该字符的颜色，字符背景颜色，字符的大小，以及要写什么字符。请看如下：

```
GUI_SetBkColor(Blue);                    //设置背景颜色
```

```
GUI_SetColor(White);                    //设置前景颜色，及字体和绘图的颜色
GUI_SetFont(&GUI_Font24_1);             //先设定好字符大小
GUI_DispStringAt("----uC/GUI Manual----",35,7);
```

以上都很好理解，这里强调的是，后两个函数。

GUI_SetFont(&GUI_Font24_1);，表示设置的字符大小，这里的“24”改成32，就表明字符变大了！也可以改成其他数字，具体允许改成什么数字，可以参考 GUI_Font 下的每个 c 文件。只要看这些 c 文件的名称就知道含义！

GUI_DispStringAt("Author: XQ_STM32",60,290);，该函数表示在坐标(60,290)上，写一个字符串"Author: XQ_STM32"。至于该字符串的字符大小，之前已经通过 GUI_SetFont(&GUI_Font24_1);设置了！

这里就简单地讲解几个函数 API，读者感兴趣，可以查阅《uC/GUI 中文手册》，以获取更详细的信息。

4.2 制作自己的图形界面

这里，我们列出一个芯嵌 stm32 的 uC/GUI 的移植例程，其他的，不解释。送大家四个字：创意无限！

```
int main(void)
{
    SystemInit();
    GPIO_Config();
    FSMC_LCD_Init();
    LCD_Setup();

    GUI_Init();     //GUI 初始化
    GUI_Clear();
    GUI_SetBkColor(Blue); //设置背景颜色
    GUI_SetColor(White); //设置前景颜色，及字体和绘图的颜色
    GUI_SetFont(&GUI_Font24_1);     //先设定好字符大小
    GUI_DispStringAt("----uC/GUI Manual----",35,7);
    GUI_SetFont(&GUI_Font24_1);     //先设定好字符大小
   GUI_DispStringAt("Author: XQ_STM32",60,290); //显示字符

    GUI_SetBkColor(Blue); //设置背景颜色
    GUI_SetColor(Red); //设置前景颜色，及字体和绘图的颜色
    GUI_DrawCircle(120,110,50);     //画圆，圆心坐标是 100,100.。半径 50
    GUI_DrawCircle(95,100,5);
    GUI_DrawCircle(145,100,5);
    GUI_DrawArc(115, 128, 20, 6, -120, -10);      //最后两个是角度
    GUI_SetFont(&GUI_Font32_1);     //先设定好字符大小
    GUI_DispStringAt("So, bad boy!",50,190);
```

```
    GUI_DELAY();
    GUI_DELAY();
    GUI_DELAY();
    GUI_DELAY();
    GUIDEMO_main();
    while(1)
    {
    }
}
```

5 参考文献

由于参考文献全部来自网络资料，内容非常多，无法一一在本文档具体位置标明。在此，对引用过的原作者表示感谢，感谢您能提供如此之好的文章。

[1]. CSDN 博客 http://blog.csdn.net/zhulizhen/article/details/4673526
[2]. 作者:wzt,《STM32 之 UCGUI 移植》, 网络下载 PDF 文档, 未知首发出处。

6 附 录

6.1 ILI9341 初始化代码

```
void LCD_Setup(void)
{
    LCD_CtrlLinesConfig();
    GPIO_ResetBits(GPIOA, GPIO_Pin_4);
    Delay(0x5FFFF);
    GPIO_SetBits(GPIOA, GPIO_Pin_4 );
    Delay(0x5FFFF);
//************** Start Initial Sequence **********//
        LCD_Write_Reg(0xCF);            //Power control B
        LCD_Write_RAM(0x00);            //采用默认参数
        LCD_Write_RAM(0x81);
        LCD_Write_RAM(0X30);
```

```
LCD_Write_Reg(0xED);            //Power on sequence control
LCD_Write_RAM(0x64);
LCD_Write_RAM(0x03);
LCD_Write_RAM(0X12);
LCD_Write_RAM(0X81);

LCD_Write_Reg(0xE8);            //Driver timing control A
LCD_Write_RAM(0x85);
LCD_Write_RAM(0x10);
LCD_Write_RAM(0x7A);

LCD_Write_Reg(0xCB);            //Power control A
LCD_Write_RAM(0x39);
LCD_Write_RAM(0x2C);
LCD_Write_RAM(0x00);
LCD_Write_RAM(0x34);
LCD_Write_RAM(0x02);

LCD_Write_Reg(0xF7);            //Pump ratio control
LCD_Write_RAM(0x20);

LCD_Write_Reg(0xEA);            //Driver timing control B
LCD_Write_RAM(0x00);
LCD_Write_RAM(0x00);

LCD_Write_Reg(0xC0);      //Power control
LCD_Write_RAM(0x1B);      //VRH[5:0]

LCD_Write_Reg(0xC1);      //Power control
LCD_Write_RAM(0x01);      //BT[2:0]

LCD_Write_Reg(0xC5);      //VCOM control
LCD_Write_RAM(0x30);
LCD_Write_RAM(0x30);

LCD_Write_Reg(0xC7);      //VCOM control2
LCD_Write_RAM(0XB7);

LCD_Write_Reg(0x36);      // Memory Access Control      0x48
LCD_Write_RAM(0x08);      //MY = 0:  Row Address Order
                          //MX = 0:  Column Address Order,MX=0 即让屏从左到右扫描
                          //MV = 0:  Row / Column Exchange        ===行列不交换
                          //ML = 0:  Vertical Refresh Order     ===    0 -> 319
```

```
                                    //BGR = 1:  使用 BGR color filter panel
                                    //MH = 0:  Horizontal Refresh Order       ===    0 -> 239

LCD_Write_Reg(0x3A);
LCD_Write_RAM(0x55);

LCD_Write_Reg(0xB1);
LCD_Write_RAM(0x00);
LCD_Write_RAM(0x1A);

LCD_Write_Reg(0xB6);       // Display Function Control
LCD_Write_RAM(0x0A);
LCD_Write_RAM(0xA2);

LCD_Write_Reg(0xF2);       // 3Gamma Function Disable
LCD_Write_RAM(0x00);
LCD_Write_Reg(0x26);       //Gamma curve selected
LCD_Write_RAM(0x01);
LCD_Write_Reg(0xE0);       //Set Gamma
LCD_Write_RAM(0x0F);
LCD_Write_RAM(0x2A);
LCD_Write_RAM(0x28);
LCD_Write_RAM(0x08);
LCD_Write_RAM(0x0E);
LCD_Write_RAM(0x08);
LCD_Write_RAM(0x54);
LCD_Write_RAM(0XA9);
LCD_Write_RAM(0x43);
LCD_Write_RAM(0x0A);
LCD_Write_RAM(0x0F);
LCD_Write_RAM(0x00);
LCD_Write_RAM(0x00);
LCD_Write_RAM(0x00);
LCD_Write_RAM(0x00);
LCD_Write_Reg(0XE1);       //Set Gamma
LCD_Write_RAM(0x00);
LCD_Write_RAM(0x15);
LCD_Write_RAM(0x17);
LCD_Write_RAM(0x07);
LCD_Write_RAM(0x11);
LCD_Write_RAM(0x06);
LCD_Write_RAM(0x2B);
LCD_Write_RAM(0x56);
LCD_Write_RAM(0x3C);
```

```
        LCD_Write_RAM(0x05);
        LCD_Write_RAM(0x10);
        LCD_Write_RAM(0x0F);
        LCD_Write_RAM(0x3F);
        LCD_Write_RAM(0x3F);
        LCD_Write_RAM(0x0F);

        LCD_Write_Reg(0x2B);          //注意事项参见 0x2A 命令
        LCD_Write_RAM(0x00);
        LCD_Write_RAM(0x00);
        LCD_Write_RAM(0x01);
        LCD_Write_RAM(0x3f);

        LCD_Write_Reg(0x2A);          //列地址设置，起始地址为 0，末地址为 EFH，即 239
        LCD_Write_RAM(0x00);           //注意，起始地址必须 比 末地址 小！
        LCD_Write_RAM(0x00);           //并且，地址要小于 239 或 319，否则大出的部分将被
忽略
        LCD_Write_RAM(0x00);
        LCD_Write_RAM(0xef);

        LCD_Write_Reg(0x11); //Exit Sleep
        Delay(0XFFFFF);
        LCD_Write_Reg(0x29); //display on

         LCD_Clear(White);

    }
```