

基于 ucOSII ucGUI 的 LED 闪烁控制例程手册

作者 Sun68

嵌入式实时操作系统 uCosII 是由美国工程师 Jean J.Labrosse 所创，它在中国的流行源于那本被邵贝贝引进翻译的著名书籍《嵌入式实时操作系统 uCos-II》，这本书是学习 uCosII 的宝典，虽然很厚，但理解了关键概念，再结合实际应用例程，还是很容易看懂的。uCosII 通过了美国航天管理局（FAA）的安全认证，可以用于飞机、航天器与人生命攸关的控制系统中。也就是说，用户可以放心将 uCosII 用到自己的产品中，ucGUI 也是 Micrium 公司的产品，在本例程里使用了 ucGUI3.90 版本，它为嵌入式应用提供了功能强大的图形用户接口，使得用户在开发具有人机界面的应用时，可以很方便做出复杂精致的用户显示界面。并提供了交互的接口。

uCOSII 特点：

可移植性：uCOSII 源码绝大部分是用移植性很强的 ANSI C 写的。与微处理硬件相关的部分是用汇编语言写的。uCOS 可以在绝大多数 8 位、16 位、32 位以及 64 位处理器、微控制器及数字信号处理器（DSP）上运行。

可裁剪性：可以通过开关条件编译选项，来定义哪些 uCOSII 的功能模块用于用户程序，方便控制代码运行所占用的空间及内存。

可剥夺性：uCOSII 是完全可剥夺型的实时内核，它总是运行处于就绪状态下的优先级最高的任务。

多任务：uCOSII 可以管理 64 个任务，每个任务对应一个优先级，并且是各不相同。其中 8 个任务保留给 uCOSII。用户的应用程序可以实际使用 56 个任务。

可确定性：绝大多数 uCosII 的函数调用和服务的执行时间具有可确定性，也就是说用户总是能知道函数调用与服务执行了多长时间。

任务栈：每个任务都有自己单独的栈，uCOSII 规定每个任务有不同的栈空间。

系统服务：uCOSII 提供很多系统服务，例如信号量、互斥信号量、事件标志、消息邮箱、消息队列、内存的申请与释放及时间管理函数等。

中断管理：中断可以使正在执行的任务暂时挂起，中断嵌套层数可达 255 层。

ucGUI 特点：

UCGUI 的设计目标是为使用 LCD 作为图形显示装置的应用提供高效的与 LCD 控制器独立及处理器独立的图形用户接口。它适合于单任务环境及多任务环境，如私用的操作系统或是商业的 RTOS(实时操作系统)。UCGUI 以 C 源码形式提供，并适用于任意 LCD 控制器和 CPU 下任何尺寸的真实显示或虚拟显示。它包含以下特性：

一般特性

- [1] 适用任何 8/16/32 位 CPU，只要有相对应的标准 C 编译器。
- [2] 任何的控制器的 LCD 显示器(单色, 灰度, 颜色)，只要有适合的 LCD 驱动可用。
- [3] 在小模式显示时无须 LCD 控制器。
- [4] 所有接口支持使用宏进行配制。
- [5] 显示尺寸可定制。
- [6] 字符和位图可在 LCD 显示器上的任意起点显示, 并不仅局限于偶数对齐的地址起点。
- [7] 程序在大小和速度上都进行了优化。

[8] 编译时允许进行不同的优化。

[9] 对于缓慢一些的 LCD 控制器，LCD 显存可以映射到内存当中，从而减少访问次数到最小并达到更高的显示速度。

[10] 清晰的设计架构。

[11] 支持虚拟显示，虚拟显示可以比实际尺寸大(即放大)。

1. GPIO功能描述

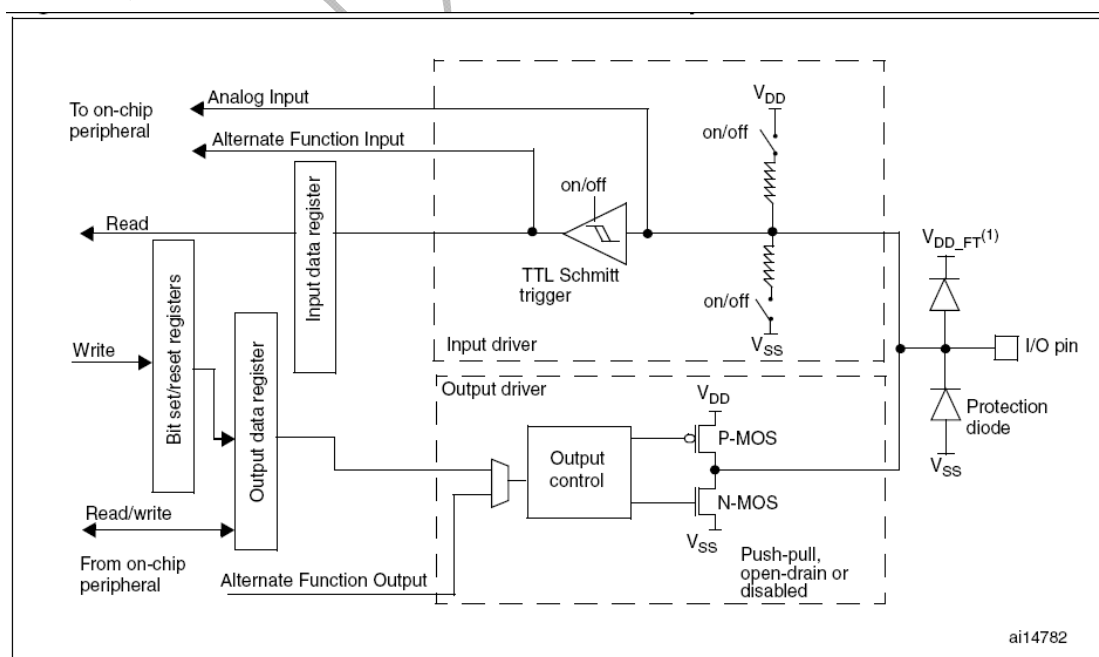
每个GPIO/0 端口有两个32 位配置寄存器(GPIOx_CRL, GPIOx_CRH)，两个32 位数据寄存器(GPIOx_IDR, GPIOx_ODR)，一个32 位置位/复位寄存器(GPIOx_BSRR)，一个16 位复位寄存器(GPIOx_BRR)和一个32 位锁定寄存器(GPIOx_LCKR)。

根据数据手册中列出的每个I/O 端口的特定硬件特征， GPIO 端口的每个位可以由软件分别配置成多种模式。

- 输入浮空
- 输入上拉
- 输入下拉
- 模拟输入
- 开漏输出
- 推挽式输出
- 推挽式复用功能
- 开漏复用功能

每个I/O 端口位可以自由编程，然而I/O 端口寄存器必须按32 位字被访问(不允许半字或字节访问)。GPIOx_BSRR 和GPIOx_BRR 寄存器允许对任何GPIO 寄存器的读/更改的独立访问；这样，在读和更改访问之间产生IRQ 时不会发生危险。

下图给出了一个5V兼容I/O端口位的基本结构。



(1) VDD_FT 对5 伏兼容I/O 脚是特殊的，它与VDD 不同

端口位配置表

配置模式		CNF1	CNF0	MODE1	MODE0	PxODR寄存器
通用输出	推挽式(Push-Pull)	0	0	01 10 11 见表12		0 或 1
	开漏(Open-Drain)		1			0 或 1
复用功能输出	推挽式(Push-Pull)	1	0			不使用
	开漏(Open-Drain)		1			不使用
输入	模拟输入	0	0	00		不使用
	浮空输入		1			不使用
	下拉输入	1	0			0
	上拉输入					1

输出模式位

MODE[1:0]	意义
00	保留
01	最大输出速度为10MHz
10	最大输出速度为2MHz
11	最大输出速度为50MHz

2 通用I/O (GPIO)

复位期间和刚复位后，复用功能未开启，I/O 端口被配置成浮空输入模式(CNFX[1:0]=01b，MODEx[1:0]=00b)。复位后，JTAG 引脚被置于输入上拉或下拉模式：

- PA15：JTDI 置于上拉模式
- PA14：JTCK 置于下拉模式
- PA13：JTMS 置于上拉模式
- PB4：JNTRST 置于上拉模式

当作为输出配置时，写到输出数据寄存器上的值(GPIOx_ODR)输出到相应的I/O引脚。可以以推挽模式或开漏模式(当输出0 时，只有N-MOS 被打开)使用输出驱动器。输入数据寄存器(GPIOx_IDR)在每个APB2 时钟周期捕捉I/O 引脚上的数据。所有GPIO 引脚有一个内部弱上拉和弱下拉，当配置为输入时，它们可以被激活也可以不被激活。

2.1 输入配置

当I/O 端口配置为输入时：

- 输出缓冲器被禁止
- 施密特触发输入被激活
- 根据输入配置(上拉，下拉或浮动)的不同，弱上拉和下拉电阻被连接
- 出现在I/O 脚上的数据在每个APB2 时钟被采样到输入数据寄存器
- 对输入数据寄存器的读访问可得到I/O 状态

2.2 输出配置

当I/O 端口被配置为输出时：

- 输出缓冲器被激活
- 开漏模式：输出寄存器上的0 激活N-MOS，而输出寄存器上的1 将端口置于高阻状态（P-MOS 从不被激活）。
- 推挽模式：输出寄存器上的0 激活N-MOS，而输出寄存器上的1 将激活P-MOS。
- 施密特触发输入被激活
- 弱上拉和下拉电阻被禁止
- 出现在I/O 脚上的数据在每个APB2 时钟被采样到输入数据寄存器
- 在开漏模式时，对输入数据寄存器的读访问可得到I/O 状态
- 在推挽式模式时，对输出数据寄存器的读访问得到最后一次写的值。

2.3 GPIO寄存器描述

2.3.1 端口配置低寄存器(GPIOx_CRL) (x=A..E)

偏移地址：0x00

复位值：0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW
位	CNFy[1:0]：端口x配置位(y = 0...7)														
31 : 30	软件通过这些位配置相应的I/O端口，请参考表11端口位配置表。														
27 : 26	在输入模式(MODE[1:0]=00)：														
23 : 22	00：模拟输入模式														
19 : 18	01：浮空输入模式(复位后的状态)														
15 : 14	10：上拉/下拉输入模式														
11 : 10	11：保留														
7 : 6	在输出模式(MODE[1:0]>00)：														
3 : 2	00：通用推挽输出模式														
	01：通用开漏输出模式														
	10：复用功能推挽输出模式														
	11：复用功能开漏输出模式														
位	MODEy[1:0]：端口x的模式位(y = 0...7)														
9 : 28	软件通过这些位配置相应的I/O端口，请参考表11端口位配置表。														
25 : 24	00：输入模式(复位后的状态)														
21 : 20	01：输出模式，最大速度10MHz														
17 : 16	10：输出模式，最大速度2MHz														
13 : 12	11：输出模式，最大速度50MHz														
9 : 8															
5 : 4															
1 : 0															

2.3.2 端口配置高寄存器(GPIOx_CRH) (x=A..E)

偏移地址：0x04

复位值：0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]	MODE15[1:0]	CNF14[1:0]	MODE14[1:0]	CNF13[1:0]	MODE13[1:0]	CNF12[1:0]	MODE12[1:0]								
rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]	MODE11[1:0]	CNF10[1:0]	MODE10[1:0]	CNF9[1:0]	MODE9[1:0]	CNF8[1:0]	MODE8[1:0]								
rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW

位	CNFy[1:0]：端口x配置位(y = 8...15)
31:30	软件通过这些位配置相应的I/O端口，请参考表11端口位配置表。
27:26	在输入模式(MODE[1:0]=00)：
23:22	00：模拟输入模式
19:18	01：浮空输入模式(复位后的状态)
15:14	10：上拉/下拉输入模式
11:10	11：保留
7:6	在输出模式(MODE[1:0]>00)：
3:2	00：通用推挽输出模式
	01：通用开漏输出模式
	10：复用功能推挽输出模式
	11：复用功能开漏输出模式
位	MODEy[1:0]：端口x的模式位(y = 8...15)
9:28	软件通过这些位配置相应的I/O端口，请参考表11端口位配置表。
25:24	00：输入模式(复位后的状态)
21:20	01：输出模式，最大速度10MHz
17:16	10：输出模式，最大速度2MHz
13:12	11：输出模式，最大速度50MHz
9:8	
5:4	
1:0	

2.3.3 端口输入数据寄存器(GPI0x_IDR) (x=A..E)

地址偏移：0x08

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

位31:16	保留，始终读为0。
位15:0	IDRy[15:0]：端口输入数据(y = 0...15) 这些位为只读并只能以字(16位)的形式读出。读出的值为对应I/O口的状态。

2.3.4 端口输出数据寄存器(GPI0x_ODR) (x=A..E)

地址偏移：0Ch

复位值：00000000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

位31:16	保留，始终读为0。
位15:0	<p>ODRy[15:0]：端口输出数据(y = 0...15)</p> <p>这些位可读可写并只能以字(16位)的形式操作。</p> <p>注：对GPIOx_BSRR(x = A...E)，可以分别地对各个ODR位进行独立的设置/清除。</p>

2.3.5 端口位设置/复位寄存器(GPIOx_BSRR) (x=A..E)

地址偏移：0x10

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

位31:16	<p>BRy: 清除端口x的位y (y = 0...15)</p> <p>这些位只能写入并只能以字(16位)的形式操作。</p> <p>0：对对应的ODRy位不产生影响</p> <p>1：清除对应的ODRy位为0</p> <p>注：如果同时设置了BSy和BRy的对应位，BSy位起作用。</p>
位15:0	<p>BSy: 设置端口x的位y (y = 0...15)</p> <p>这些位只能写入并只能以字(16位)的形式操作。</p> <p>0：对对应的ODRy位不产生影响</p> <p>1：设置对应的ODRy位为1</p>

2.3.6 端口位复位寄存器(GPIOx_BRR) (x=A..E)

地址偏移：0x14

复位值：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

位31:16	保留。
位15:0	BRy: 清除端口x的位y (y = 0...15) 这些位只能写入并只能以字(16位)的形式操作。 0: 对对应的ODRy位不产生影响 1: 清除对应的ODRy位为0

2.3.7 端口配置锁定寄存器(GPIOx_LCKR) (x=A..E)

当执行正确的写序列设置了位16(LCKK)时，该寄存器用来锁定端口位的配置。位[15:0]用于锁定GPIO 端口的配置。在规定的写入操作期间，不能改变LCKP[15:0]。当对相应的端口位执行了LOCK 序列后，在下次系统复位之前将不能再更改端口位的配置。每个锁定位锁定控制寄存器(CRL, CRH)中相应的4 个位。

地址偏移: 0x18
复位值: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															LCKK
															rW
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0
rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW

位31:17	保留。
位16	LCKK: 锁键 该位可随时读出，它只可通过锁键写入序列修改。 0: 端口配置锁键位激活 1: 端口配置锁键位被激活，下次系统复位前GPIOx_LCKR寄存器被锁住。 锁键的写入序列： 写1 -> 写0 -> 写1 -> 读0 -> 读1 最后一个读可省略，但可以用来确认锁键已被激活。 注：在操作锁键的写入序列时，不能改变LCK[15:0]的值。 操作锁键写入序列中的任何错误将不能激活锁键。
位15:0	LCKy: 端口x的锁位y (y = 0...15) 这些位可读可写但只能在LCKK位为0时写入。 0: 不锁定端口的配置 1: 锁定端口的配置

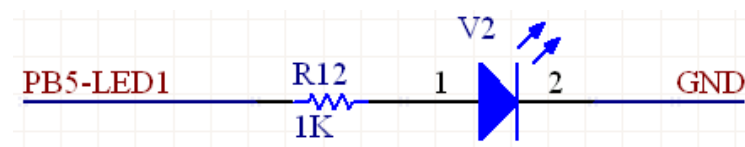
3. 应用实例

3.1. 设计要求

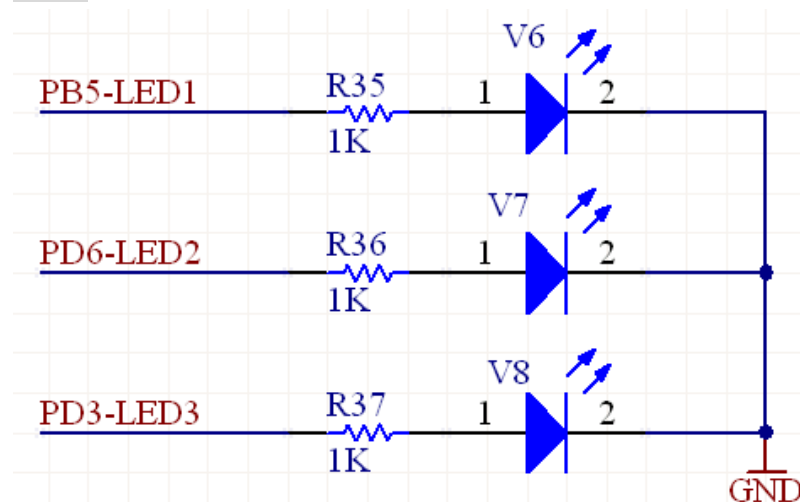
基于ucos2.86和ucgui3.90平台编程，实现滑动液晶屏上滑动条可以控制led的闪烁间隔。间隔范围是50毫秒—20秒。也可以通过串口指令控制led的闪烁间隔。

硬件电路设计

MINI板



V5板



应用

奋斗 STM32 开发板 MINI 及 V5 采用了 STM32F103VET6 作为板上的 MCU，内置 512K FLASH 64K SRAM。非常适合短小精悍的 uCosII 作为操作系统。而且 ucosII 是实时操作系统，也极适合 STM32 所面对的嵌入式微控领域。奋斗板选用了已经被移植到 STM32 平台上的 ucosII2.86 源码。经过广泛测试，这个移植好的源码在 STM32 上是运行可靠的，我们可以更加专心关注应用软件的开发。我们选用了 ucGUI 来作为 STM32 开发的图形用户接口。

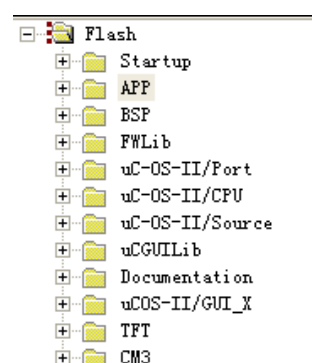
根据设计要求，对这个例程进行了工程策划，选用MDK3.80a作为工程编译环境。JLINK V8作为下载仿真器，采用一个任务作为初始化时的主任务，用于建立6个用户任务。6个用户任务分别是，触摸控制任务（用于实时获得触摸的坐标）、界面任务（用于显示状态，人机交互）、串口1通信任务（用于接收控制指令）、LED1闪烁任务、LED2闪烁任务、LED3闪烁任务，根据实时响应的重要程度，将各个任务的优先级进行了设置。

任务名	优先级
APP_TASK_START_PRIO	2
APP_TASK_USER_IF_PRIO	13
APP_TASK_KBD_PRIO	12
Task_Com1_PRIO	4
Task_Led1_PRIO	7
Task_Led2_PRIO	8
Task_Led3_PRIO	9

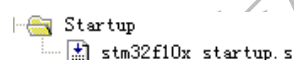
为了兼顾实时效率及 CPU 的负荷。将 ucOSII 的时钟节拍设置为 10ms，ucOSII 需要提供周期性信号源，用于实现时间延时和确认超时，时钟节拍的含意就是任务和任务之间最短切换时间。这个节拍也不能设置的非常短，会造成 CPU 负荷过大，会造成任务执行兼顾不周。某些高优先级任务总是在执行，有些低优先级任务得不到执行。但节拍也不能设置的非常长，这会造成任务执行的实时性变差。一般 10-100ms 就可以了。

下面分析一下这个程序的结构。

打开工程，可以在工程结构栏看到这个例程的工程结构（如下图）



Startup 组项:



包含了适用于 STM32F103 高容量系列的启动文件。这是程序的执行的入口文件。在上电启动时，主要完成了对堆栈的初始设置，设置中断向量表，以及跳转到最终指向 main（）函数的 C 库。

APP 组项:



App.c 里包含了任务的建立、各任务的原型以及 ucOSII 内核的启动。

Stm32f10x_it.c 里包含了各个中断服务程序。在这个例程中，只用到了两个中断，一个是 systick 时钟节拍中断，一个是串口 1 接收中断。Systick 时钟节拍中断为 ucOSII 内核提供了 10ms 的时钟节拍。

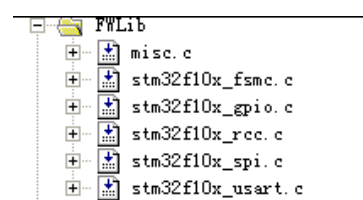
Fun.C 里包含了基于 ucGUI 的应用程序

BSP 组项:



Bsp.c 包含了对所用外设的初始化。

FWLIB 组项:



这个组项里包含了例程所用的到的 STM32 的各外设固件库。

Misc.c 是和中断设置有关的固件库

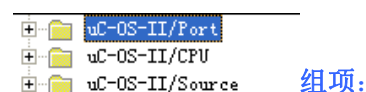
Stm32f10x_gpio.c 是和通用端口有关的库

Stm32f10x_rcc.c 是和外设时钟有关的库

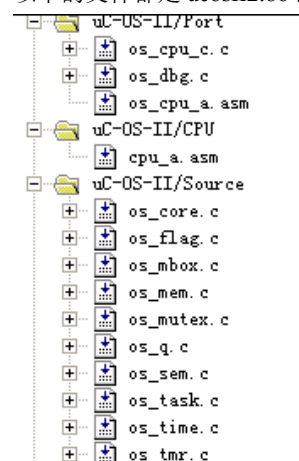
Stm32f10x_fsmc.c 是和显示接口有关的库

Stm32f10x_spi.c 是和 SPI 接口有关的库

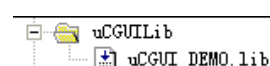
Stm32f10x_usart.c 是和串口有关的库



以下的文件都是 ucosII2.86 源码。这些文件已经移植到 STM32 平台下，可以直接包含到工程里使用。

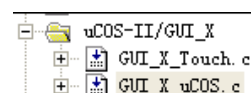


uCGUILib 组项:



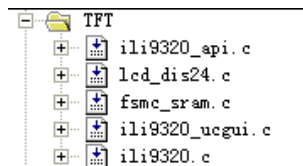
uCGUI_DEMO.lib 包含了和 ucgui 各部件资源有关的库，在这里包含库文件，而不是包含源文件，是为了在开发时，编译速度更快捷。如果要修改显示屏的分辨率的参数，及触摸屏原点初始值，请在光盘里的 uCGUI_LIB 横版显示库或者 uCGUI_LIB 竖版显示库目录下的工程里进行修改。并重新编译为 lib。将这个 lib 文件，拷贝到应用程序的 uCGUILib 目录下替换以前的旧 lib 文件。uCGUI_LIB 横版显示库目录下的工程是用于奋斗板显示屏的横板 ucgui 例程显示。uCGUI_LIB 竖版显示库目录下的工程是用于奋斗板显示屏的竖板 ucgui 例程显示。

uCOS-II/GUI_X 组项:



这个组项下的文件和在 ucOSII 内核调度下的触摸操作有关，也和 ucOSII 内核调度下的 ucgui 任务有关。

TFT 组项:



Ili9320_api.c 是 ucGUI 部件所用到的底层函数原型定义

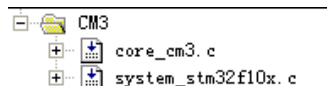
Lcd_dis24.c 包含了奋斗板显示屏模块硬件驱动程序

Fsmc.sram.c 是和 FSMC 显示接口有关的定义

Ili9320_ucgui.c 是和 ucgui 画图函数有关系的接口函数定义

Ili9320.c 是 ucgui 与 fsmc 有关系的接口函数库

CM3 组项:



Core_cm3.c 包含了 Coretex-M3 内核的外设访问层源文件。

System_stm32f10x.c 包含了和 Coretex-M3 系统时钟有关系的外设访问层源文件。

例程所用到的 uCosII 概念解释:

任务: 任务通常是一个无限的循环，返回参数必须定义为 void。当任务开始执行时，会有一个参数传递给用户任务代码。uCOSII 可以管理 64 个任务，其中系统保留了 8 个任务。开放给用户的有 56 个任务，每个任务的优先级都不同，任务的优先级号越低，任务的优先级越高，在这个版本的 uCosII 中，任务的优先级号就是任务编号。

任务的状态一定是以下 5 种之一：

睡眠态: 就是任务没有交给 ucosII 调度的，也就是没用经过建立的任务。只保存在存储器空间里。

就绪态: 任务一旦建立，任务就进入就绪态。任务可以通过调用 OSTasjDel() 返回到睡眠态。

运行态: 任何时刻只能有一个任务处于运行态。

等待状态: 正在运行的任务可以通过调用以下 2 个函数之一，将自身延迟一段时间。这 2 个函数是 OSTimeDly() 或 OSTimeDlyHMSM()。这个任务于是进入等待状态，一直到函数中定义的延迟时间到。正在运行的任务可能在等待某一事件的发生，可以通过调用以下函数之一实现：OSFlagPend()，OSSemPend(), OSMutexPend(), OSMboxPend() 或 OSQPend()。如果某事件并未发生，调用上述函数的任务就进入了等待状态，直到等待的事件发生了。当任务因等待事件被挂起时，下一个优先级最高的就绪任务就得到了 CPU 的使用权。当时间发生了或等待延时超时，被挂起的任务就进入就绪态。

中断服务态: 正在运行的任务是可以被中断的，被中断的任务于是进入了中断服务态，响应中断时，正在执行的任务被挂起，中断服务程序控制了 CPU 的使用权。从中断服务程序返回前，uCOSII 要判定被中断的任务是否是当前就绪任务里优先级最高的，如果不是，就执行优先级最高的那个任务。如果是，就执行被中断的这个任务。

消息邮箱: 这是 uCosII 中的一种通信机制，可以使一个任务或者中断服务程序向另一个任务发送一个指针型的变量，通常该指针指向了一个包含了消息的特定数据结构。在例程中需要建立邮箱，用到了函数 OSMboxCreate()，串口 1 中断服务程序用到了向邮箱发送一则消息的函数 OSMboxPost()，串口接收任务用到了等待邮箱中消息的函数 OSMboxPend()。

例程所用到的 uCGUI 函数原型的说明:

在例程里用到了窗体的建立，以及各窗体部件的建立。以及用于响应窗体动作变化的回调函数的编写。下面列出所用到的函数原型。

窗体的建立:

```
WM_HWIN GUI_CreateDialogBox(const GUI_WIDGET_CREATE_INFO* paWidget,
                             int NumWidgets, WM_CALLBACK* cb,
                             WM_HWIN hParent,int x0, int y0);
```

Parameter	Meaning
paWidget	Pointer to resource table defining the widgets to be included in the dialog.
NumWidgets	Total number of widgets included in the dialog.
cb	Pointer to an application-specific callback function (dialog procedure).
hParent	Handle of parent window (0 = no parent window).
x0	X-position of the dialog relative to parent window.
y0	Y-position of the dialog relative to parent window.

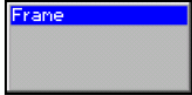
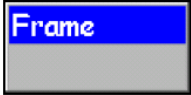
回调函数

```
void callback(WM_MESSAGE* pMsg);
```

Parameter	Meaning
pMsg	Pointer to a data structure of type WM_MESSAGE.

窗体标题栏字体设置

FRAMEWIN_SetFont()

Before	After
	

Description

Sets the title font.


Prototype

```
void FRAMEWIN_SetFont(FRAMEWIN_Handle hObj, const GUI_FONT* pfont);
```

Parameter	Meaning
hObj	Handle of frame window.
pFont	Pointer to the font.

窗体背景色设置

FRAMEWIN_SetClientColor()

Before	After
	

Description

Sets the color of the client window area of a specified frame window.

Prototype

```
void FRAMEWIN_SetClientColor(FRAMEWIN_Handle hObj, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of frame window.
Color	Color to be set.

获得对话框里某某项目的句柄

WM_GetDialogItem()

Description

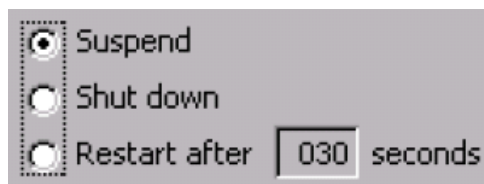
Returns the window handle of a dialog box item (widget).

Prototype

```
WM_HWIN WM_GetDialogItem(WM_HWIN hDialog, int Id);
```

Parameter	Meaning
hDialog	Handle of dialog box.
Id	Window Id of the widget.

TEXT: Text widget



文本框字体设置

TEXT_SetFont()

Description

Sets the font to be used for a specified text widget.

Prototype

```
void TEXT_SetFont(TEXT_Handle hObj, const GUI_FONT* pFont);
```

Parameter	Meaning
hObj	Handle of text widget.
pFont	Pointer to the font to be used.

文本框字体颜色设置

TEXT_SetTextColor()

Description

Sets the text color of a specified text widget.

Prototype

```
void TEXT_SetTextColor(TEXT_Handle pObj, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of text widget.
Color	New text color.

EDIT: Edit widget



文本编辑框字体设置

EDIT_SetFont()**Description**

Sets the edit field font.

Prototype

```
void EDIT_SetFont(EDIT_Handle hObj, const GUI_FONT* pfont);
```

Parameter	Meaning
<code>hObj</code>	Handle of edit field.
<code>pFont</code>	Pointer to the font.

设置文本编辑框里的文本内容

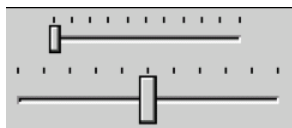
EDIT_SetText()**Description**

Sets the text to be displayed in the edit field.

Prototype

```
void EDIT_SetText(EDIT_Handle hObj, const char* s)
```

Parameter	Meaning
<code>hObj</code>	Handle of edit field.
<code>s</code>	Text to display.

SLIDER: Slider widget

设置滑动条的滑动取值范围

SLIDER_SetRange()**Description**

Sets the range of the slider.

Prototype

```
void SLIDER_SetRange(SLIDER_Handle hObj, int Min, int Max);
```

Parameter	Meaning
<code>hObj</code>	Handle of slider widget.
<code>Min</code>	Minimum value.
<code>Max</code>	Maximum value.

Additional information

After creating a slider widget the default range is set to 0 - 100.

Examples

If a value should be modified in the range of 0 - 2499 set the range as follows:

```
SLIDER_SetRange(hSlider, 0, 2499);
```

If a value should be modified in the range of 100 - 499 set the range as follows:

```
SLIDER_SetRange(hSlider, 100, 499);
```

If a value should be modified in the range of 0 to 5000 and the slider bar should change the value in steps of 250 set the range and the tick marks as follows. The result returned by `SLIDER_GetValue()` should be multiplied with 250:

```
SLIDER_SetRange(hSlider, 0, 20);
SLIDER_SetNumTicks(hSlider, 21);
```

设置滑动条的当前位置

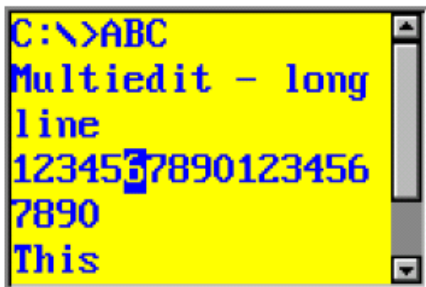
SLIDER_SetValue()

Description
Sets the current value of the slider bar.

Prototype
void SLIDER_SetValue(SLIDER_Handle hObj, int v);

Parameter	Meaning
hObj	Handle of slider widget.
v	Value to be set.

多行编辑框



在窗体上建立多行编辑框

MULTIEDIT_Create()

(Obsolete, MULTIEDIT_CreateEx should be used instead)

Description
Creates a MULTIEDIT widget of a specified size at a specified location.

Prototype
MULTIEDIT_HANDLE MULTIEDIT_Create(int x0, int y0, int xsize, int ysize, WM_HWIN hParent, int Id, int Flags, int ExFlags, const char * pText, int MaxLen);

Parameter	Meaning
x0	Leftmost pixel of the multiedit widget (in parent coordinates).
y0	Topmost pixel of the multiedit widget (in parent coordinates).
xsize	Horizontal size of the multiedit widget (in pixels).
ysize	Vertical size of the multiedit widget (in pixels).
hParent	Parent window of the multiedit widget.
Id	ID of the multiedit widget.
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 15: "The Window Manager" for a list of available parameter values).
ExFlags	(see table below)
pText	Text to be used.
MaxLen	Maximum number of bytes for text and prompt.

Permitted values for parameter ExFlags	
MULTIEDIT_CF_AUTOSCROLLBAR_H	Automatic use of a horizontal scrollbar.
MULTIEDIT_CF_AUTOSCROLLBAR_V	Automatic use of a vertical scrollbar.
MULTIEDIT_CF_INSERT	Enables insert mode.
MULTIEDIT_CF_READONLY	Enables read only mode.

设置多行编辑框的字体

MULTIEDIT_SetFont()

Description

Sets the font used to display the text and the prompt.

Prototype

```
void MULTIEDIT_SetFont(MULTIEDIT_HANDLE hObj, const GUI_FONT * pFont);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
pFont	Pointer to font to be used.

设置最大字符数量

MULTIEDIT_SetMaxNumChars()

Description

Sets the maximum number of characters used by text and prompt.

Prototype

```
void MULTIEDIT_SetMaxNumChars(MULTIEDIT_HANDLE hObj, unsigned MaxNumChars);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
MaxNumChars	Maximum number of characters.

设置文字对齐方式

MULTIEDIT_SetTextAlign()

Description

Sets the text alignment for the given MULTIEDIT widget.

Prototype

```
void MULTIEDIT_SetTextAlign(MULTIEDIT_HANDLE hObj, int Align);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
Align	(see table below)

Permitted values for parameter [Align](#)

GUI_TA_LEFT	Left text align.
GUI_TA_RIGHT	Right text align.

设置文字字体颜色

MULTIEDIT_SetTextColor()

Description

Sets the text color.

Prototype

```
void MULTIEDIT_SetTextColor(MULTIEDIT_HANDLE hObj,
                             unsigned int Index, GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
Index	(See table below)
Color	Text color to be used.

Permitted values for parameter [Index](#)

MULTIEDIT_CI_EDIT	Edit mode.
MULTIEDIT_CI_READONLY	Read only mode.

设置文字回绕

MULTIEDIT_SetWrapWord()

Description

Enables the word wrapping mode.

Prototype

```
void MULTIEDIT_SetWrapWord(MULTIEDIT_HANDLE hObj);
```

Parameter	Meaning
hObj	Handle of multiedit widget.

在多行文本框里追加内容

MULTIEDIT_AddText()

Description

Adds the given text at the current cursor position.

Prototype

```
int MULTIEDIT_AddText(MULTIEDIT_HANDLE hObj, const char * s);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
s	Pointer to a NULL terminated text to be added.

获得当前的内容的长度

MULTIEDIT_GetTextSize()

Description

Returns the buffer size used to store the current text (and prompt).

Prototype

```
int MULTIEDIT_GetTextSize(MULTIEDIT_HANDLE hObj);
```

Parameter	Meaning
hObj	Handle of multiedit widget.

设置背景颜色

MULTIEDIT_SetBkColor()

Description

Sets the background color of the given multiedit widget.

Prototype

```
void MULTIEDIT_SetBkColor(MULTIEDIT_HANDLE hObj,
                           unsigned int Index,
                           GUI_COLOR Color);
```

Parameter	Meaning
hObj	Handle of multiedit widget.
Index	(See table below)
Color	Background color to be used.

Permitted values for parameter Index	
MULTIEDIT_CI_EDIT	Edit mode.
MULTIEDIT_CI_READONLY	Read only mode.

设置字体

MULTIEDIT_SetFont()

Description

Sets the font used to display the text and the prompt.

Prototype

```
void MULTIEDIT_SetFont(MULTIEDIT_HANDLE hObj, const GUI_FONT * pFont);
```

Parameter	Meaning
<code>hObj</code>	Handle of multiedit widget.
<code>pFont</code>	Pointer to font to be used.

设置光标位置

MULTIEDIT_SetCursorOffset()

Description

Sets the cursor position to the given character.

Prototype

```
void MULTIEDIT_SetCursorOffset(MULTIEDIT_HANDLE hObj, int Offset);
```

Parameter	Meaning
<code>hObj</code>	Handle of multiedit widget.
<code>Offset</code>	New cursor position.

Additional information

The number of characters used for the prompt has to be added to the parameter Offset. If a prompt is used the value for parameter Offset should not be smaller than the number of characters used for the prompt.

程序流程讲解:

经过启动文件的初始化,首先在 C 代码里,执行 main(),在启动 ucosII 内核前先禁止 CPU 的中断 -CPU_IntDis(),防止启动过程中系统崩溃,然后对 ucosII 内核进行初始化 OSInit(),以上两个函数都不用特意修改,在 STM32 平台上已经被移植好了,对板上的一些用到的外设进行初始化设置 BSP_Init(),这个函数包含了对系统时钟的设置 RCC_Configuration(),将系统时钟设置为 72MHz,对 LED 闪烁控制端口进行设置 GPIO_Configuration(),对中断源进行配置 NVIC_Configuration(),对触摸控制进行配置 tp_config(),对显示器接口 FSMC 进行配置,板子上的外设初始化完毕

```

/*****
* 名称: void BSP_Init(void)
* 功能: 奋斗板初始化函数
* 入口参数: 无
* 出口参数: 无
* 说明:
* 调用方法: 无
*****/
void BSP_Init(void)
{
    RCC_Configuration();           //系统时钟初始化及端口外设时钟使能
    NVIC_Configuration();          //中断源配置
    GPIO_Configuration();          //状态LED1的初始化
    USART_Config(USART1,115200);   //串口1初始化
    tp_Config();                   //SPI1 触摸电路初始化
    FSMC_LCD_Init();               //FSMC TFT接口初始化
}
    
```

建立主任务，该任务是为了在内核启动后，建立另外 2 个用户任务，并清 0 节拍计数器，启动 ucOSII 内核。

```
//建立主任务， 优先级最高 建立这个任务另外一个用途是为了以后使用统计任务
os_err = OSTaskCreate((void *) (void *) App_TaskStart, //指向任务代码的指针
                      (void *) 0, //任务开始执行时，传递给任务的参数的指针
                      (OS_STK *) &App_TaskStartStk[APP_TASK_START_STK_SIZE - 1], //分配给任务的堆栈的栈顶指针 从顶向下递减
                      (INT8U) APP_TASK_START_PRIO); //分配给任务的优先级

OSTimeSet(0); //ucosII的节拍计数器清0 节拍计数器是0-4294967295
OSStart(); //启动ucosII内核
```

主任务的任务名为 App_TaskStart，主任务有自己的堆栈，堆栈尺寸为 APP_TASK_START_STK_SIZE*4(字节)，然后执行 ucosII 内部函数 OSTimeSet(0)，将节拍计数器清 0，节拍计数器范围是 0-4294967295，对于节拍频率 100hz 时，每隔 497 天就重新计数，调用内部函数 OSStart()，启动 ucosII 内核，此时 ucosII 内核开始运行。对任务表进行监视，主任务因为已经处于就绪状态，于是开始执行主任务 App_TaskStart()，ucOSII 的任务结构规定必须为无返回的结构，也就是无限循环模式。

```
/* *****
 * 名称: static void App_TaskStart(void* p_arg)
 * 功能: 开始任务建立
 * 入口参数: 无
 * 出口参数: 无
 * 说明:
 * 调用方法: 无
 * *****
static void App_TaskStart(void* p_arg)
{
    (void) p_arg;
    //初始化ucosII时钟节拍
    OS_CPU_SysTickInit();

    //使能ucos 的统计任务
    #if (OS_TASK_STAT_EN > 0)

        OSStatInit(); //----统计任务初始化函数
    #endif

    App_TaskCreate(); //建立其他的任务

    while (1)
    {
        OSTimeDlyHMSM(0, 0, 1, 100);
    }
}
```

```

/*****
* 名称: static void App_TaskCreate(void)
* 功能: 建立其余任务的函数
* 入口参数: 无
* 出口参数: 无
* 说明:
* 调用方法: 无
*****/
static void App_TaskCreate(void)
{
    Com1_MBOX=OSMboxCreate((void *) 0);          //建立串口1中断的邮箱
    /* 建立用户界面任务 */
    OSTaskCreateExt(AppTaskUserIF,
        (void *)0,
        (OS_STK *) &AppTaskUserIFStk[APP_TASK_USER_IF_STK_SIZE-1],
        APP_TASK_USER_IF_PRIO,
        APP_TASK_USER_IF_PRIO,
        (OS_STK *) &AppTaskUserIFStk[0],
        APP_TASK_USER_IF_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CLR);

    /* 建立触摸驱动任务 */
    OSTaskCreateExt(AppTaskKbd,
        (void *)0,
        (OS_STK *) &AppTaskKbdStk[APP_TASK_KBD_STK_SIZE-1],
        APP_TASK_KBD_PRIO,
        APP_TASK_KBD_PRIO,
        (OS_STK *) &AppTaskKbdStk[0],
        APP_TASK_KBD_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CLR);

    //串口1接收及发送任务
    OSTaskCreateExt(Task_Com1,(void *)0,(OS_STK *) &Task_Com1Stk[Task_Com1_STK_SIZE-1],Task_Com1_PRIO,Task_Com1_PRIO,
        (OS_STK *) &Task_Com1Stk[0],
        Task_Com1_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CLR);

    //LED1 闪烁任务
    OSTaskCreateExt(Task_Led1,(void *)0,(OS_STK *) &Task_Led1Stk[Task_Led1_STK_SIZE-1],Task_Led1_PRIO,Task_Led1_PRIO,
        (OS_STK *) &Task_Led1Stk[0],
        Task_Led1_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CLR);

    //LED2 闪烁任务
    OSTaskCreateExt(Task_Led2,(void *)0,(OS_STK *) &Task_Led2Stk[Task_Led2_STK_SIZE-1],Task_Led2_PRIO,Task_Led2_PRIO,
        (OS_STK *) &Task_Led2Stk[0],
        Task_Led2_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CLR);

    //LED3 闪烁任务
    OSTaskCreateExt(Task_Led3,(void *)0,(OS_STK *) &Task_Led3Stk[Task_Led3_STK_SIZE-1],Task_Led3_PRIO,Task_Led3_PRIO,
        (OS_STK *) &Task_Led3Stk[0],
        Task_Led3_STK_SIZE,
        (void *)0,
        OS_TASK_OPT_STK_CHK|OS_TASK_OPT_STK_CLR);
}

```

6 个用户任务各自有自己的堆栈空间。

```

/*****
* 名称: static void AppTaskUserIF (void *p_arg)
* 功能: 用户界面任务
* 入口参数: 无
* 出口参数: 无
* 说明:
* 调用方法: 无
*****/
static void AppTaskUserIF (void *p_arg)
{
    (void)p_arg;
    GUI_Init();          //ucgui初始化
    while(1)
    {
        Fun();          //界面主程序
    }
}

/*****
* 名称: static void AppTaskKbd (void *p_arg)
* 功能: 触摸屏坐标获取
* 入口参数: 无
* 出口参数: 无
* 说明:
* 调用方法: 无
*****/
static void AppTaskKbd (void *p_arg)
{
    (void)p_arg;
    while(1)
    {
        /* 延时10ms会读取一次触摸坐标 */
        OSTimeDlyHMSM(0,0,0,10);
        GUI_TOUCH_Exec();
    }
}

```

```

/*****
* 名称: static void Task_Led1(void* p_arg)
* 功能: LED1闪烁任务
* 入口参数: 无
* 出口参数: 无
* 说明:
* 调用方法: 无
*****/
static void Task_Led1(void* p_arg)
{
    (void) p_arg;
    while (1)
    {
        LED_LED1_ON();
        OSTimeDlyHMSM(0, 0, 0, milsec1);

        LED_LED1_OFF();
        OSTimeDlyHMSM(0, 0, 0, milsec1);
    }
}

/*****
* 名称: static void Task_Led2(void* p_arg)
* 功能: LED2闪烁任务
* 入口参数: 无
* 出口参数: 无
* 说明:
* 调用方法: 无
*****/
static void Task_Led2(void* p_arg)
{
    (void) p_arg;
    while (1)
    {
        LED_LED2_ON();
        OSTimeDlyHMSM(0, 0, 0, milsec2);

        LED_LED2_OFF();
        OSTimeDlyHMSM(0, 0, 0, milsec2);
    }
}

/*****
* 名称: static void Task_Led3(void* p_arg)
* 功能: LED3闪烁任务
* 入口参数: 无
* 出口参数: 无
* 说明:
* 调用方法: 无
*****/
static void Task_Led3(void* p_arg)
{
    (void) p_arg;
    while (1)
    {
        LED_LED3_ON();
        OSTimeDlyHMSM(0, 0, 0, milsec3);

        LED_LED3_OFF();
        OSTimeDlyHMSM(0, 0, 0, milsec3);
    }
}

/*****
* 名称: static void Task_Com1(void *p_arg)
* 功能: 串口1任务
* 入口参数: 无
* 出口参数: 无
* 说明:
* 调用方法: 无
*****/
static void Task_Com1(void *p_arg) {
    INTSU err;
    unsigned char * msg;
    (void)p_arg;
    while(1){
        msg=(unsigned char *)OSMboxPend(Com1_MBOX,0,&err); //等待串口接收指令成功的信号量
        if(msg[0]=='L' && msg[1]==0x31){
            milsec1=atoi(&msg[3]); //LED1 的延时毫秒 (mini and V3)
            USART_OUT(USART1, "\r\n");
            USART_OUT(USART1, "LED1: %d ms 间隔闪烁", milsec1);
        }
        else if(msg[0]=='L' && msg[1]==0x32){
            milsec2=atoi(&msg[3]); //LED2 的延时毫秒 (only V3)
            USART_OUT(USART1, "\r\n");
            USART_OUT(USART1, "LED2: %d ms 间隔闪烁", milsec2);
        }
        else if(msg[0]=='L' && msg[1]==0x33){
            milsec3=atoi(&msg[3]); //LED3 的延时毫秒 (only V3)
            USART_OUT(USART1, "\r\n");
            USART_OUT(USART1, "LED3: %d ms 间隔闪烁", milsec3);
        }
    }
}

```

void USART1_IRQHandler(void)为串口 1 接收中断响应程序

```

void USART1_IRQHandler(void)
{
    unsigned int i;
    unsigned char msg[50];
    OS_CPU_SR cpu_sr;

    OS_ENTER_CRITICAL(); //保存全局中断标志,关总中断// Tell uc/OS-II that we are starting an ISR
    OSIntNesting++;      //用于中断嵌套
    OS_EXIT_CRITICAL();   //恢复全局中断标志

    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET) //判断读寄存器是否非空
    {
        msg[RxCounter1++] = USART_ReceiveData(USART1); //将读寄存器的数据缓存到接收缓冲区里
        if(msg[RxCounter1-1]=='L'){msg[0]='L'; RxCounter1=1;} //判断起始标志
        if(msg[RxCounter1-1]=='F') //判断结束标志是否是"F"
        {
            for(i=0; i< RxCounter1; i++){
                TxBuffer1[i] =msg[i]; //将接收缓冲器的数据转到发送缓冲区,准备转发
            }
            rec_f=1; //接收成功标志
            TxBuffer1[RxCounter1]=0; //接收缓冲区终止符
            RxCounter1=0;
            OSMboxPost(Com1_MBOX, (void *) &msg);
        }
    }
    if(USART_GetITStatus(USART1, USART_IT_TXE) != RESET)
    {
        USART_ITConfig(USART1, USART_IT_TXE, DISABLE);
    }
    OSIntExit(); //在os_core.c文件里定义,如果有更高优先级的任务就绪了,则执行一次任务切换
}

```

定义对话框资源列表

```

/* 定义对话框资源列表 */
static const GUI_WIDGET_CREATE_INFO aDialogCreate[] = {
    //建立窗体, 大小是240X320 原点在0,0
    { FRAMEWIN_CreateIndirect, "LED Flash Config", 0, 0, 0, 240, 400, FRAMEWIN_CF_ACTIVE },

    //建立TEXT控件, 起点是窗体的10, 20, 大小180X30 文字左对齐
    { TEXT_CreateIndirect, "Led Flash Rate", GUI_ID_TEXT3, 10, 20, 180, 30, TEXT_CF_LEFT },

    //建立TEXT控件, 起点是窗体的200, 20, 大小39X30 文字左对齐
    { TEXT_CreateIndirect, "ms", GUI_ID_TEXT4, 200, 20, 39, 30, TEXT_CF_LEFT },

    //建立EDIT控件, 起点是窗体的191, 60, 大小47X25 文字右对齐 4个字符宽度
    { EDIT_CreateIndirect, "", GUI_ID_EDIT0, 191, 60, 47, 25, EDIT_CF_RIGHT, 4 },
    //建立EDIT控件, 起点是窗体的191, 110, 大小47X25 文字右对齐 4个字符宽度
    { EDIT_CreateIndirect, "", GUI_ID_EDIT1, 191, 110, 47, 25, EDIT_CF_RIGHT, 4 },
    //建立EDIT控件, 起点是窗体的191, 160, 大小47X25 文字右对齐 4个字符宽度
    { EDIT_CreateIndirect, "", GUI_ID_EDIT2, 191, 160, 47, 25, EDIT_CF_RIGHT, 4 },

    //建立TEXT控件, 起点是窗体的5, 60, 大小50X55 文字右对齐
    { TEXT_CreateIndirect, "Led1", GUI_ID_TEXT0, 5, 60, 50, 55, TEXT_CF_RIGHT },
    //建立TEXT控件, 起点是窗体的5, 110, 大小50X105 文字右对齐
    { TEXT_CreateIndirect, "Led2", GUI_ID_TEXT1, 5, 110, 50, 105, TEXT_CF_RIGHT },
    //建立TEXT控件, 起点是窗体的5, 160, 大小50X155 文字右对齐
    { TEXT_CreateIndirect, "Led3", GUI_ID_TEXT2, 5, 160, 50, 155, TEXT_CF_RIGHT },

    //建立滑动条控件, 起点是窗体的60, 60, 大小130X25
    { SLIDER_CreateIndirect, NULL, GUI_ID_SLIDERO, 60, 60, 130, 25, 0, 0 },
    //建立滑动条控件, 起点是窗体的60, 110, 大小130X25
    { SLIDER_CreateIndirect, NULL, GUI_ID_SLIDER1, 60, 110, 130, 25, 0, 0 },
    //建立滑动条控件, 起点是窗体的60, 160, 大小130X25
    { SLIDER_CreateIndirect, NULL, GUI_ID_SLIDER2, 60, 160, 130, 25, 0, 0 },
};

```

void Fun(void) 函数是显示界面的主函数。特别说明一下,本例程增加了 13 点阵的汉字字库,这个字库不是 ucgui 的标准字库,因此需要在 gui.h 里声明一下。所以在 gui.h 里做了如下的修改,红色部分就是增加的声明该字体的语句,声明后,就可以在 ucgui 的字体设置里进行应用。

```

font
+---+
+---+ SimSun_13.c

```


Gui.c

```
/* Digits */  
  
extern GUI_CONST_STORAGE GUI_FONT GUI_FontD24x32;  
  
extern GUI_CONST_STORAGE GUI_FONT GUI_FontD32;  
  
extern GUI_CONST_STORAGE GUI_FONT GUI_FontD36x48;  
  
extern GUI_CONST_STORAGE GUI_FONT GUI_FontD48;  
  
extern GUI_CONST_STORAGE GUI_FONT GUI_FontD48x64;  
  
extern GUI_CONST_STORAGE GUI_FONT GUI_FontD64;  
  
extern GUI_CONST_STORAGE GUI_FONT GUI_FontD60x80;  
  
extern GUI_CONST_STORAGE GUI_FONT GUI_FontD80;  
  
/* Comic fonts */  
  
extern GUI_CONST_STORAGE GUI_FONT GUI_FontComic18B_ASCII, GUI_FontComic18B_1;  
extern GUI_CONST_STORAGE GUI_FONT GUI_FontComic24B_ASCII, GUI_FontComic24B_1;  
  
extern GUI_CONST_STORAGE GUI_FONT GUI_FontHZ_SimSun_13;
```

引用方法例如程序里的用法

```
/* 设置 multiedit 部件的字体 */  
MULTIEDIT_SetFont(hmultiedit,&GUI_FontHZ_SimSun_13);
```

```
void Fun(void) {
    unsigned char edit_cur;
    GUI_CURSOR_Show();
    WM_SetCreateFlags(WM_CF_MEMDEV); /* Automatically use memory devices on all windows */
    /* 建立窗体, 包含了资源列表, 资源数目, 并指定回调函数 */
    hWin = GUI_CreateDialogBox(aDialogCreate, GUI_COUNTOF(aDialogCreate), _cbCallback, 0, 0, 0);

    /* 设置窗体字体 */
    FRAMEWIN_SetFont(hWin, &GUI_FontComic24B_1);

    /* 获得TEXT 部件的句柄 */
    text0 = WM_GetDialogItem(hWin, GUI_ID_TEXT0);
    text1 = WM_GetDialogItem(hWin, GUI_ID_TEXT1);
    text2 = WM_GetDialogItem(hWin, GUI_ID_TEXT2);
    text3 = WM_GetDialogItem(hWin, GUI_ID_TEXT3);
    text4 = WM_GetDialogItem(hWin, GUI_ID_TEXT4);

    /* 获得slider部件的句柄 */
    slider0 = WM_GetDialogItem(hWin, GUI_ID_SLIDERO);
    slider1 = WM_GetDialogItem(hWin, GUI_ID_SLIDER1);
    slider2 = WM_GetDialogItem(hWin, GUI_ID_SLIDER2);

    /* 获得edit 部件的句柄 */
    edit0 = WM_GetDialogItem(hWin, GUI_ID_EDIT0);
    edit1 = WM_GetDialogItem(hWin, GUI_ID_EDIT1);
    edit2 = WM_GetDialogItem(hWin, GUI_ID_EDIT2);

    /* 设置TEXT部件的字体 */
    EDIT_SetFont(edit0, &GUI_FontComic18B_1);
    EDIT_SetFont(edit1, &GUI_FontComic18B_1);
    EDIT_SetFont(edit2, &GUI_FontComic18B_1);

    /* 设置EDIT部件采用10进制 范围50-20000 */
    EDIT_SetDecMode(edit0, milsec1, 50, 2000, 0, 0);
    EDIT_SetDecMode(edit1, milsec2, 50, 2000, 0, 0);
    EDIT_SetDecMode(edit2, milsec3, 50, 2000, 0, 0);

    /* 设置TEXT部件的字体 */
    TEXT_SetFont(text0, pFont);
    TEXT_SetFont(text1, pFont);
    TEXT_SetFont(text2, pFont);
    TEXT_SetFont(text3, pFont);
    TEXT_SetFont(text4, pFont);

    /* 设置TEXT部件的字体颜色 */
    TEXT_SetTextColor(text0, GUI_WHITE);
    TEXT_SetTextColor(text1, GUI_WHITE);
    TEXT_SetTextColor(text2, GUI_WHITE);
    TEXT_SetTextColor(text3, GUI_WHITE);
    TEXT_SetTextColor(text4, GUI_WHITE);

    /* 设置slider部件的取值范围50-2000 */
    SLIDER_SetRange(slider0, 50, 2000);
    SLIDER_SetRange(slider1, 50, 2000);
    SLIDER_SetRange(slider2, 50, 2000);

    /* 设置slider部件的值 */
    SLIDER_SetValue(slider0, milsec1);
    SLIDER_SetValue(slider1, milsec2);
    SLIDER_SetValue(slider2, milsec3);

    /* 在窗体上建立multiedit部件 */
    hmultiedit = MULTIEDIT_Create(5, 230, 230, 160, hWin, GUI_ID_MULTIEDIT0, WM_CF_SHOW, MULTIEDIT_CF_AUTOSCROLLBAR_V,
    "", 500);

    /* 设置multiedit部件的字体 */
    MULTIEDIT_SetFont(hmultiedit, &GUI_FontHZ_SimSun_13);
    /* 设置multiedit部件的背景色 */
    MULTIEDIT_SetBkColor(hmultiedit, MULTIEDIT_CI_EDIT, GUI_LIGHTGRAY);
    /* 设置multiedit部件的字体颜色 */
    MULTIEDIT_SetTextColor(hmultiedit, MULTIEDIT_CI_EDIT, GUI_BLACK);
    /* 设置multiedit部件的文字环绕 */
    MULTIEDIT_SetWrapWord(hmultiedit);
    /* 设置multiedit部件的最大字符数 */
    MULTIEDIT_SetMaxNumChars(hmultiedit, 500);
    /* 设置multiedit部件的字符左对齐 */
    MULTIEDIT_SetTextAlign(hmultiedit, GUI_TA_LEFT);
    /* 获得multiedit部件里光标位置 */
    edit_cur = MULTIEDIT_GetTextSize(hmultiedit);
    /* 设置multiedit部件光标位置 */
    MULTIEDIT_SetCursorOffset(hmultiedit, edit_cur);
    /* 设置multiedit部件的文本内容 */
    MULTIEDIT_AddText(hmultiedit, "奋斗STM32开发板LED闪烁实验");
    while (1)
    {
        if (rec_f == 1) {
            rec_f = 0; //全局变量 rec_f 代表串口有数据接收到
            edit_cur = MULTIEDIT_GetTextSize(hmultiedit); //获得MULTIEDIT 内容的长度
            if (edit_cur < 500) { //显示区域字符长度小于500 继续添加显示
                MULTIEDIT_SetCursorOffset(hmultiedit, edit_cur);
                MULTIEDIT_AddText(hmultiedit, &TxBuffer1[0]); //在内容的最后增加来自于串口的新的内容
            }
            else { //显示区域字符长度大于等于500 清除显示区。继续重新显示
                MULTIEDIT_SetText(hmultiedit, &TxBuffer1[0]);
            }
            if (TxBuffer1[0] == 'L' && TxBuffer1[1] == 0x31) { //读取串口接收到的信息
                milsec1 = atoi(&TxBuffer1[3]); //LED1 的延时毫秒 (mini and V3)
                SLIDER_SetValue(slider0, milsec1); //改变slider0的值
                EDIT_SetValue(edit0, milsec1); //改变edit0的值
            }
            else if (TxBuffer1[0] == 'L' && TxBuffer1[1] == 0x32) { //读取串口接收到的信息
                milsec2 = atoi(&TxBuffer1[3]); //LED2 的延时毫秒 (mini and V3)
                SLIDER_SetValue(slider1, milsec2); //改变slider1的值
                EDIT_SetValue(edit1, milsec2); //改变edit1的值
            }
            else if (TxBuffer1[0] == 'L' && TxBuffer1[1] == 0x33) { //读取串口接收到的信息
                milsec3 = atoi(&TxBuffer1[3]); //LED3 的延时毫秒 (mini and V3)
                SLIDER_SetValue(slider2, milsec3); //改变slider2的值
                EDIT_SetValue(edit2, milsec3); //改变edit3的值
            }
        }
        WM_Exec(); //屏幕刷新
    }
}
```

static void _cbCallback()定义了窗体动作响应的回调函数

```

/*****
* 名    称: static void _cbCallback(WM_MESSAGE * pMsg)
* 功    能: 窗体回调函数
* 入口参数: 无
* 出口参数: 无
* 说    明:
* 调用方法: 无
*****/
static void _cbCallback(WM_MESSAGE * pMsg) {
    int NCode, Id;
    WM_HWIN hDlg;
    hDlg = pMsg->hWin;
    switch (pMsg->MsgId) {
        case WM_NOTIFY_PARENT:
            Id = WM_GetId(pMsg->hWinSrc); /*获得窗体部件的ID*/
            NCode = pMsg->Data.v; /*动作代码*/
            switch (NCode) {
                case WM_NOTIFICATION_VALUE_CHANGED: /*窗体部件的值被改变*/
                    _OnValueChanged(hDlg, Id);
                    break;
                default:
                    break;
            }
            break;
        default:
            WM_DefaultProc(pMsg);
    }
}

/*****
* 名    称: static void _OnValueChanged(WM_HWIN hDlg, int Id)
* 功    能: 值被改变的动作
* 入口参数: 无
* 出口参数: 无
* 说    明:
* 调用方法: 无
*****/
static void _OnValueChanged(WM_HWIN hDlg, int Id) {

    if ((Id == GUI_ID_SLIDER0)) { /*slider0 的值被改变
                                   //获得slider0的值
                                   //EDIT0 的值被改变
    }
    else if ((Id == GUI_ID_SLIDER1)) { /*slider1 的值被改变
                                   //获得slider1的值
                                   //EDIT1 的值被改变
    }
    else if ((Id == GUI_ID_SLIDER2)) { /*slider2 的值被改变
                                   //获得slider2的值
                                   //EDIT2 的值被改变
    }
}

```

运行过程

按  编译工程，完成后会提示如下。

```

compiling bsp.c...
linking...
Program Size: Code=52452 RO-data=29576 RW-data=608 ZI-data=15552
FromELF: creating hex file...
"..\\ObjFlash\\STM32-FD-ucgui.axf" - 0 Error(s), 1 Warning(s).

```

将代码下载到开发板里，并执行。按照界面提示进行操作。

显示效果



例程目录名：STM32奋斗板-LED闪烁-ucgui ucos

2011 年 12 月 22 日 西安

奋斗嵌入式开发工作室