

ucos 用户手册

何 聪

2012.8.30

目录

一、	系统介绍.....	2
二、	目录结构.....	2
三、	ucos 功能模块图.....	3
四、	系统配置和 Makefile 分析.....	3
五、	编译并且烧写 ucos	5
六、	ucos 启动流程	6
七、	简单 shell.....	6
1.	shell 命令介绍	6
2.	shell 命令添加	10
八、	各模块介绍	11
1.	fatfs	11
2.	spi flash	12
3.	Nand flash	12
4.	sd 驱动.....	13
5.	usb 驱动.....	13
6.	UCGUI	13
7.	AC97	16
九、	有待改善.....	16
1.	usb 驱动	16
2.	AC97 驱动	16
3.	触摸驱动.....	17
十、	参考文献.....	17

一、 系统介绍

系统移植时，使用了如下版本的源码：

ucos 2.6.1	ucosII
lwip 1.11	网络协议栈
fatfs R0.09	文件系统
ucgui 3.90a	图形系统

支持的功能：

- a) 支持 FAT 文件系统
- b) 支持 spi flash 读、写、擦除
- c) 支持 nand flash 读、写、擦除
- d) 支持 TF 卡，读、写和热插拔
- e) 支持 usb 大容量存储设备，读、写和热插拔
- f) 支持 lwip 网络协议栈
- g) 支持 gmac
- h) 支持 ucgui
- i) 支持 lcd
- j) 支持触摸屏
- k) 支持 AC97，录音、播音

基本的信息：

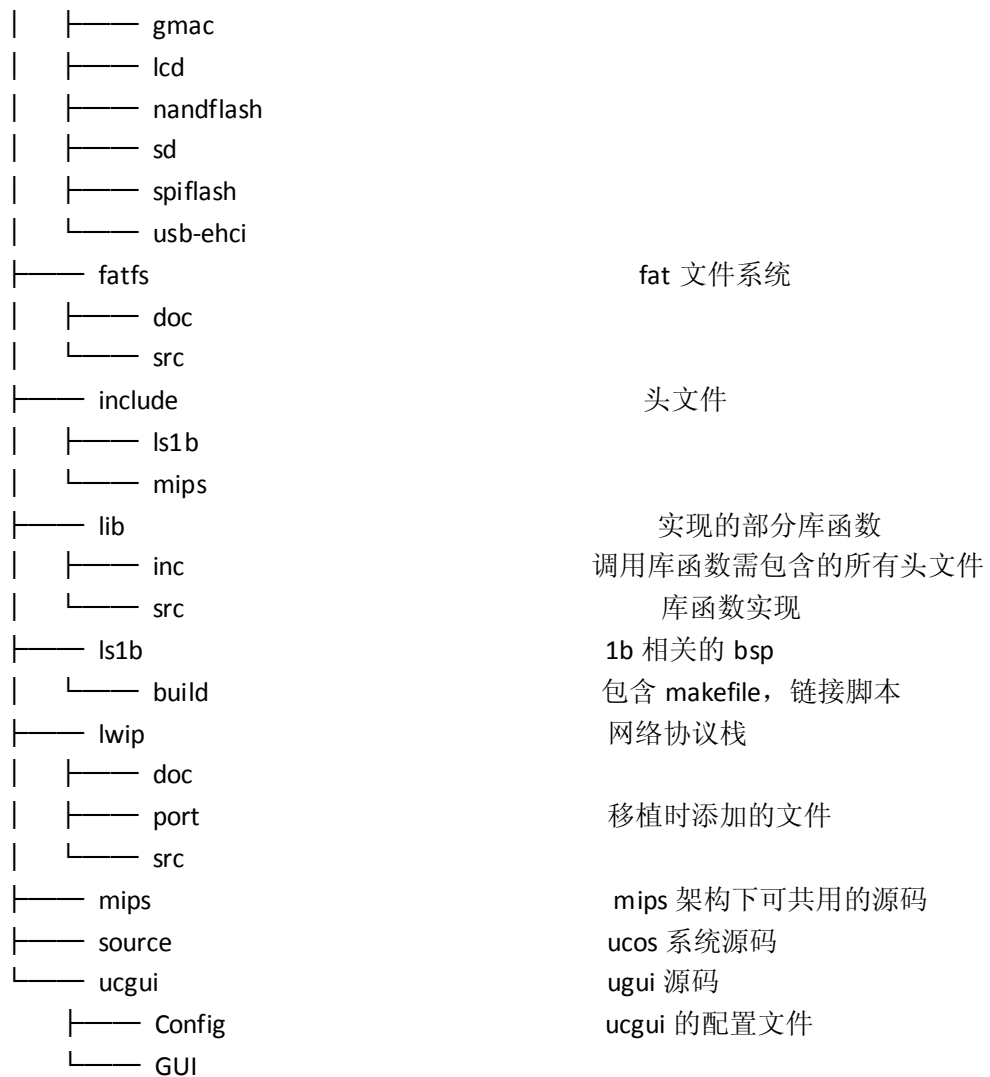
ucos 默认 ip 为 192.168.1.2

tftp 默认 pc 的 ip 为 192.168.1.22

系统实现了一个简单的 httpd，在确保 ping 通的情况下，可在浏览器输入 192.168.1.2 来访问网页。

二、 目录结构

├── app	应用程序或测试程序
├── cmds	go 命令的实现
├── driver	1b 上所有的驱动
│ ├── ac97	
│ └── ads7846	



三、ucos 功能模块图

上层	底层驱动
LWIP	GMAC
FATFS	spi-flash
	NAND-flash
	SD
	USB
UCGUI	LCD
	触摸屏
	AC97

四、系统配置和 Makefile 分析

1. 系统配置

系统根目录下有文件 config.mk, 内容如下:

```
CONFIG_LWIP=1
CONFIG_GMAC=1
```

```

CONFIG_FATFS=1
CONFIG_NORFLASH=1
CONFIG_NANDFLASH=1
CONFIG_SDCCARD=1
CONFIG_UCGUI=1
CONFIG_TOUCH=1
CONFIG_LCD=1
CONFIG_USB_EHCI=1
CONFIG_AC97=1

```

置 1 选中相应模块，置 0 取消该模块。可以直接通过在该文件设置，即可选择需要的模块进行编译。当然，在选择模块时，应该参照上面的功能模块图，有些模块之间是存在依赖性的，比如：LWIP 和 GMAC，网卡驱动和网络协议栈。系统并不能自动检测模块之间的依赖性。

2. Makefile 分析

```

MODULE_CONFIG=../../include/module_config.h
CONFIG_MK=../../config.mk

# -----
config:
    @echo "#ifndef _MODULE_CONFIG_H_">$(MODULE_CONFIG);\
    echo "#define _MODULE_CONFIG_H_">>$(MODULE_CONFIG);\
    if [ "$(grep "CONFIG_LWIP=1" $(CONFIG_MK))" ]; then \
        echo "#define CONFIG_LWIP">>$(MODULE_CONFIG); \
        echo "select LWIP"; \
    fi; \

```

这就是 make config 目标，查找 config.mk 文件，如果对应的模块置 1，就在 include/module_config.h 中加入相应的宏申明。最后，会根据 config.mk 生成文件 module_config.h，这个头文件被用来进行条件编译。

```
include $(CONFIG_MK)
```

包含文件 config.mk，看 config.mk 文件，这就相当于在 Makefile 中定义了 config.mk 内容中的变量。

```

ifeq ($(CONFIG_NANDFLASH),1)
SRCDIR += $(NANDFLASH)
endif
.....

```

很明显，判断变量是否为 1，加入对应模块到编译的源码变量里，SRCDIR 在 Makefile 的后面会被用来提取需要编译的文件。

```
ifeq ($(CONFIG_NANDFLASH),1)
```

```
INCLUDE += -I$(NANDFLASH)
endif
.....
```

判断变量加入头文件的搜索路径。

根据上面的介绍，用户可以自己进行添加需编译的文件夹，需搜索的头文件路径。

五、 编译并且烧写 uc0s

1. 编译

a) 确保你的系统上有 mips 交叉编译工具 gcc-3.4.6-2f（使用 4.3 会出现链接错误），并且在你的 PATH 路径下。

b) 确保 Makefile 里面的

CC = mipsel-linux-gcc

LD = mipsel-linux-ld

OBJCOPY = mipsel-linux-objcopy

OBJDUMP = mipsel-linux-objdump

根据你自己的编译器名字来设定

c) 进入 uc0s/lslb/build 目录下，运行

make clean 删除上次的编译生成的文件

make config 根据 config.mk，生成相应的文件

make all 编译

编译完成后将生成 uc0s-lslb.elf 可执行文件，uc0s-lslb.bin 二进制文件。

2. 烧写

A. 如果你的 spiflash 被更换为 1MB 以上，你将可以直接烧写到 spiflash 中，方法如下：

i. 从 build 目录下 copy uc0s-lslb.bin 到你的 tftp 对应的文件夹下

ii. 确保开发板跟 pc 网线相连，启动开发板到 pmon，在互相能 ping 通的情况下，运行如下命令：

load -r -f bfc00000 tftp://192.168.1.xx/uc0s-lslb.bin (ip 为 pc 上 ip)

iii. 重新启动开发板，即可

B. 烧写到 NANDFLASH

- i. 从 build 目录下 copy `ucos-ls1b.elf` 到你的 tftp 对应的文件夹下
- ii. 确保开发板跟 pc 网线相连, 启动开发板到 pmon, 在互相能 ping 通的情况下, 运行如下命令 (建议烧到 mtd1 或者 mtd2 分区中, 方便测试 Nandflash 功能):

```
devcp tftp://192.168.1.xx/ucos-ls1b.elf /dev/mtd0
```

- iii. 设置 pmon 的环境变量:

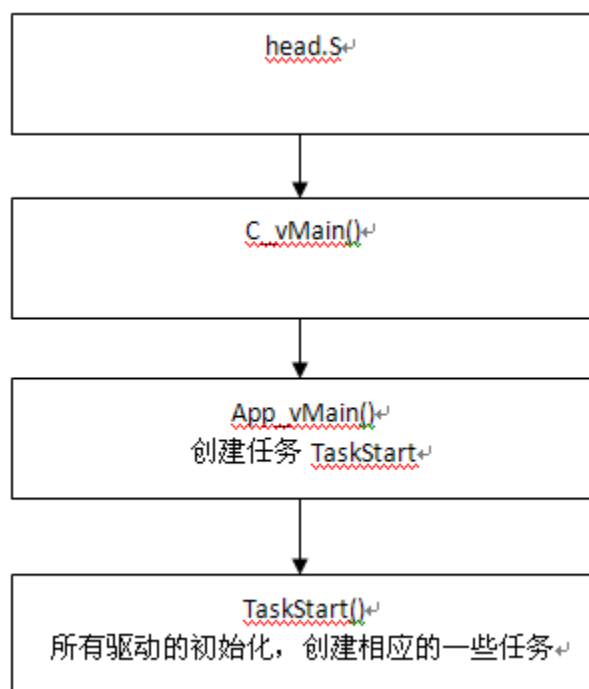
```
set al /dev/mtd0
```

设置 append 为任意值

- iv. 重启开发板, 即可

六、 ucos 启动流程

启动涉及三个文件(全部在 ls1b 目录下): `head.S` `frmwrk.c` `main.c`



七、 简单 shell

1. shell 命令介绍

支持的命令:

tftp	通过 tftp 下载文件到指定的内存
------	--------------------

go	跳到指定的内存去运行
help	列出当前支持的所有命令
fat_test	简单测试各存储设备
sd_usb_test	TF 卡和 u 盘互相拷贝
audio_test	简单音频测试，录音几秒，再播放

1) tftp

格式：

tftp memaddr (内存地址) filename (文件名)

示例： **tftp 0x81000000 uc0s-ls1b-10.bin**

使用该命令之前，设置 pc 上 ip 为 192.168.1.22，确保 pc 能 ping 通开发板，对应的文件在 tftp 目录下。

Document 中提供了三个 uc0s 二进制文件：

uc0s-ls1b-10.bin，起始地址为 0x81000000

uc0s-ls1b-12.bin 起始地址为 0x81200000

uc0s-ls1b-20.bin 起始地址为 0x82000000

这三个文件可用来测试，结合 go 命令，去验证是否下载成功。

2) go

格式：

go memaddr(内存地址)

示例： **go 0x81000000**

如果地址指定错误，或者指定的地址不是代码的起始地址，可能出现异常。

3) help

格式： **help**

简单列出当前支持的所有命令，实现功能较简单，有待完善。

4) fat_test

格式：

fat_test r/w/f (读 / 写 / 格式化) drive (驱动器号)

drive	0	1	2	3	4	5	6
设备	spi flash	Nand flash	TF 卡	第一个 插入的 u 盘	第二个 插入的 u 盘	第三个 插入的 u 盘	第四个 插入的 u 盘

注意：请确保 u 盘或者 TF 卡位 Fat 格式，否则会打印出相应的出错信息，默认写入 loongson, Hello world!-2012\r\n 到 loongson.txt 文件中（硬编码，只为简单测试），在 linux 下用 gedit 打开会提示一些问题，windows 下没问题，因为 linux 和 windows 下回车换行符不一样，一个是\n，一个是\r\n。

示例： fat_test w 1 写 nandflash

5) sd_usb_copy

格式： sd_usb_copy

TF 卡插入和识别正确后，将打印出如下信息：

```
SD Card Init Success
Manufacturer ID (MID) : 0x73
OEM/Application ID (OID) : BG
Product Name (PNM) : NCard
get CSD register...
40 e 0 32 5b 59 0 0 76 89 7f 80 a 40 0
93
get CSD register...
sd card : sector_size=512, sectors=31073280, capacity=15536640KB
```

注意：tf 卡的识别不是特别灵敏（原因不明，似乎 linux 下也有此问题），插入后如果显示： NO SdCard ，那么拔出再重新插入，如果多次不行，就重启再试。

运行前，请插入 1 个 u 盘一个 tf 卡，格式为 fat32，确保先插入的 u 盘中有源文件 LOONGSON.TXT 文件（建议拷贝一个文档，音乐，电影等等，方便运行后检查），可在开机前插入，也可在开机后插入，程序运行时拔掉正在使用的 u 盘或者 TF 卡，将停止 copy。

拷贝过程如下：

```

          u 盘                      TF
LOONGSON.TXT ----->LOONGSON.TXT
test1    <-----
```

```

-----> test1
test2  <-----
-----> test3
....

```

最后结果：每个下面都有 5 个文件 LOONGSON.TXT test test1 test2 test3
 （应该有一个文件是不完整的，因为最后是通过把掉 u 盘来停止程序的）该命令运行后，将不会停止，将会重复拷贝。

运行结果：

```

*****USB          TO          TF*****

```

拷贝方向：usb 到 Tf

u0_to_u1=1,createfile(LOONGSON.TXT))

在该方向上拷贝的总次数，以及在 TF 上创建的文件名

```

=====copy start=====

```

```

.....

```

```

.....

```

```

=====copy success=====

```

Close the file.

交叉拷贝，usb 到 Tf，然后 Tf 到 usb

```

*****TF          TO          USB*****

```

Tf 到 usb

u1_to_u0=1, create file(LOONGSON.TXT)

```

=====copy start=====

```

```

.....

```

```

.....

```

```

=====copy success=====

```

Close the file.

如果快速的打印出许多错误信息，请检查 u 盘或者 tf 卡是否满足测试要求，比如：是否 fat，是否有对应文件，是否识别正确等等。

6) audio_test

格式： audio_test

示例：

```
[ucos-loongson]$audio_test
```

record audio start

.....

Start palying voice ...

.Finish palying voice!

2. shell 命令添加

shell.h

```
struct cmd_t {  
    char name[20];           命令名，也就是在 shell 中使用的  
    unsigned int argc;       命令需要的参数，命令本身算一个参数  
    void (*func)(char **);   函数指针，指向跟此命令相关的函数  
};
```

ls1b/shell.c

```
#define CMD_MAX 6           命令的最大数目
```

```
struct cmd_t cmd_table[] = {  
    [0]={  
        .name = "tftp",  
        .argc = 3,  
        .func = tftp_thread,  
    },  
    [1]={  
        .name = "go",  
        .argc = 2,  
        .func = do_go,  
    },  
    [2]={  
        .name = "fat_test",  
        .argc = 3,  
        .func = fat_test,  
    },  
    [3]={  
        .name = "audio_test",  
        .argc = 1,  
        .func = audio_test,  
    },  
    [4]={  
        .name = "usb_copy_test",  
        .argc = 1,  
        .func = usb_mult_copy,  
    },  
    [5]={  
        .name = NULL,  
    },  
};
```

```

        .argc = 0,
        .func = NULL,
    },
};

```

假如你需要添加一个命令，在[4]那项后面添加

```

    [5]={
        .name=xx,
        .argc = x,
        .func = xxx,
    },
    [6]={
        .name = NULL,
        .argc = 0,
        . func = NULL,
    },

```

最后 CMD_MAX 加 1，变为 7。

注意：你要确保你的函数声明所在的头文件，被 **shell.c** 包含。

八、 各模块介绍

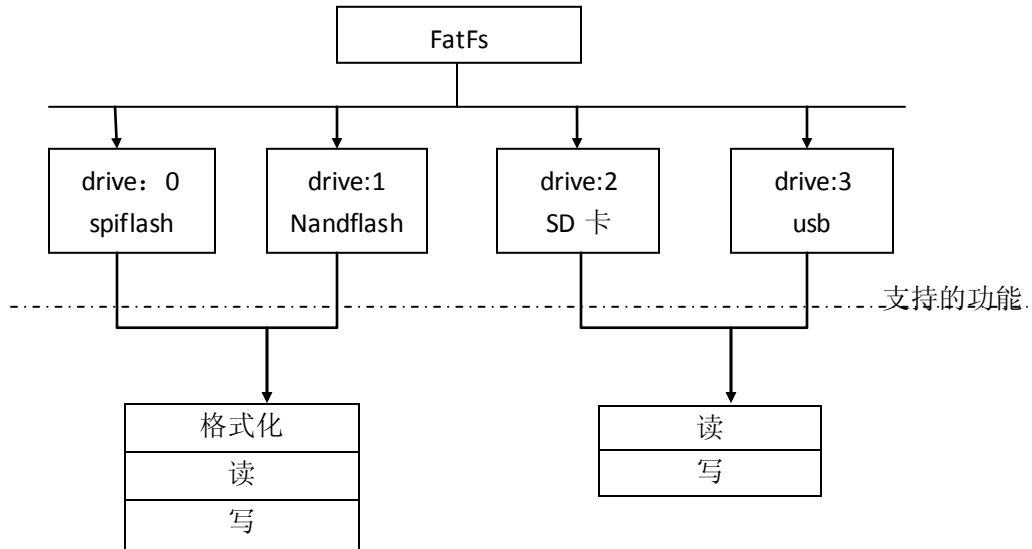
1. fatfs

FatFS FatFs 是一个通用的文件系统模块，用于在小型嵌入式系统中实现 FAT 文件系

统。 FatFs 的编写遵循 ANSI C，因此不依赖于硬件平台。它可以嵌入到便宜的微控制器中，如 8051, PIC, AVR, SH, Z80, H8, ARM 等等，不需要做任何修改。

系统暂时只支持 8.3 文件名格式，不支持长文件名，不支持中文名。8.3 文件名格式，通常指较旧的 Windows 操作系统或 DOS 的文件命名限制，8 指文件名最大长度为 8，3 指扩展名最大长度为 3，若不符合以上限制则会以 "~" 作延长名称如 "Program Files" 会变成 "Progra~1"，若同一文件夹有相似的名称，末端的数值则会自动递增。

fatfs 模块图如下：



具体的 API 使用和移植，请参考 fatfs/doc/00index_e.html，该网页有详细的介绍。

2. spi flash

spi flash 驱动的源码和 pmon 的基本一样，就不罗嗦了。

spi flash 跟 fatfs 的接口函数，在 `nor_flash.c` 中，其中最主要的设置：

```
#define MAX_SECTOR_NUM    128*8
#define SECTOR_SIZE       512
```

和 `nor_flash_ioctl`，详细请参考 fatfs 文档。

3. Nand flash

fatfs 接口文件，`nand_flash.c`，其中的 `read`、`write` 函数并没有判断是否超出范围，

因为这个判断，在 `nand flash` 驱动中已经做了。

`nand flash` 驱动的包含调试信息，可在 `include/lb1b/mtd/mtd.h` 中设置：

```
/*
 * Debugging macro and defines
 */
#define MTD_DEBUG_LEVEL0    (0) /* Quiet */
#define MTD_DEBUG_LEVEL1    (1) /* Audible */
#define MTD_DEBUG_LEVEL2    (2) /* Loud */
#define MTD_DEBUG_LEVEL3    (3) /* Noisy */
#define CONFIG_MTD_DEBUG_VERBOSE 1
#define CONFIG_MTD_DEBUG

#ifdef CONFIG_MTD_DEBUG
```

```
#define DEBUG(n, args...) \
    if (n <= CONFIG_MTD_DEBUG_VERBOSE) {\
        printk(KERN_INFO args);\
    }\
#else /* CONFIG_MTD_DEBUG */\
#define DEBUG(n, args...)\
#endif /* CONFIG_MTD_DEBUG */
```

4. sd 驱动

sd 驱动支持 tf 卡的热插拔,但是有时候已经插进去了,还显示 NO SDCard。好像 1b 上的 linux 也有此问题,有时候插拔检测不准确。

文件系统的接口函数在 sd_card.c, 其中容量大小来源 sd 驱动初始化时,保存到结构体中的信息。

```
extern struct sd_info sf;
#define MAX_SECTOR_NUM    (sf.sector_count)
#define SECTOR_SIZE       (sf.sector_multiply)
```

sf 定义在 sd 卡驱动中,有 sd_get_info()函数通过读取 sd 卡的 csd 字段,计算来填充的。

5. usb 驱动

现在,系统只移植了 EHCI 控制器,支持大容量存储设备,支持热插拔。usb 较复杂,usb 驱动移植时,也是一边学习一边移植,虽然实现了一些功能,但是存在很多缺陷,拷贝速度较慢,未采用中断方式等等。

1b 支持 4 个 usb 接口,usb_storage.c 中 read 、write 通过第一个参数来区分插入的设备。所以在 disk_usb.c 中定义了

```
int usb_nr = 0; /* usb select */
```

读写函数中用到这个变量:

```
usb_stor_read(usb_nr, sector_number, sector_count, buff);
usb_stor_write(usb_nr, sector_number, sector_count, buff);
```

在 disk.c 中:

```
case DN_USB:
    usb_nr=0;
    return usb_initialize();
case DN_USB_1:
    usb_nr=1;
    return usb_initialize();
```

usb 驱动详细的分析,可查看我写的另外一篇<<usb 驱动分析>>。

6. UCGUI

```

ucgui/
├── Config
├── GUI
│   ├── AntiAlias
│   ├── ConvertColor
│   ├── ConvertMono
│   ├── Core
│   ├── Font
│   ├── JPEG
│   │   ├── Doc
│   │   └── Image
│   ├── LCDDriver
│   ├── MemDev
│   ├── MultiLayer
│   ├── Widget
│   └── WM

```

下表显示了所有 ucgui 子目录

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 *

1) ucgui 的配置

所有的配置信息都在 Config 文件夹下：

```

Config/
├── GUIConf.h
├── GUITouchConf.h
└── LCDConf.h

```

具体的配置信息，请参考 ucgui 官方文档。

2) ucgui 和底层驱动的接口

在 GUI 的层次调用中，所有的 API 函数都是最后调用了针对屏的画点函数。因此在进行驱动程序的移植的过程中只需要编写针对次液晶屏的画点和初始化函数即可。用户自己的驱动程序在 gui 文件夹下的 LCDDriver 下的 LCDDummy.c 文件。在这个文件中有 2 个函数分别为：void LCD_L0_SetPixelIndex(int x, int y, int PixelIndex) 和 unsigned int LCD_L0_GetPixelIndex(int x, int y)，而在这 2 个函数中可以分别看到

```
/* Write into hardware ... Adapt to your system */
```

```
{
```

```
/* ... */
```

```
}
```

和

```
/* Read from hardware ... Adapt to your system */
```

```
{
```

```
PixelIndex = 0; /* ... */
```

```
}
```

这段注释的意思就是要求用户添加自己的驱动程序。在下面的 LCD_L0_Init 函数中将 LCD_INIT_CONTROLLER() 改为用户自己的 LCD 初始化函数。

我在 void LCD_L0_SetPixelIndex(int x, int y, int PixelIndex) 中加入了

```
LCDSIM_SetPixelIndex(xPhys, yPhys, PixelIndex);
```

该函数实现：

```
void LCDSIM_SetPixelIndex(int x, int y, int Index, int LayerIndex)
```

```
{
```

```
    lcd_setpixel(x, y, Index);
```

```
}
```

GetPixel 也是一样

```
int LCDSIM_GetPixelIndex(int x, int y, int LayerIndex)
```

```
{
```

```
    unsigned char *fb;
```

```
    int offset;
```

```

    unsigned long video_fb_address = MEM_ptr;
    unsigned short color;
    offset = y * VIDEO_LINE_LEN + x * VIDEO_PIXEL_SIZE;
    fb = (void *)(video_fb_address) + offset;
    color = *(unsigned short *)fb;
    return (int)color;
}

```

Init 函数

```

int LCD_L0_Init(void) {
    lcd_init();
    return 0;
}

```

详细移植请参考网上的文档。

整个 ucgui 框架如下：

上层	ucgui		
底层	LCD 驱动	触摸驱动	ucos 系统支持
ucgui 接口文件	LCDDummy.c	GUI_X_TOUCH.c	GUI_X_uCOS.c
底层驱动	dc.c	ads7846.c	

3) ucgui api

详细的 API 请参阅官方文档。

7. AC97

ac97 驱动，只支持简单的录音和放音功能，支持中断。导出了两个 API：

```
int pcm_read(char *buffer, int count)
```

```
int pcm_write(char *buffer, unsigned int count)
```

分别用于录音和播放。

九、有待改善

1. usb 驱动

usb 驱动是最需要改善的地方，整个框架对于像 bootload 类的程序可以胜任，但是操作系统来说，就会造成很多不足。对于本系统，关键的函数没有确保可重入，数据的发送没有使用中断。

2. AC97 驱动

实现功能较少，稳定性也不够，会被其他任务影响到声音的清晰度。

3. 触摸驱动

有时候触摸成功, 并且已经进入相应中断函数, 但是却读不出相关数据。

另外, 触摸校准也存在一些问题。

十、 参考文献

- 1) <<1B 开发板用户手册>>
- 2) <<嵌入式实时操作系统μ COS-II>>
- 3) <http://webspaces.webring.com/people/wm/michaelanburaj/>
- 4) <<uCOS 平台下的 LWIP 移植笔记>>
- 5) <<LWIP 中文手册>>
- 6) http://elm-chan.org/fsw/ff/00index_e.html
- 7) <<Fatfs 经典资料>>
- 8) <<uCGUI 中文手册>>
- 9) <<UCGUI 移植经验总结>>
- 10) <<ucgui 触摸屏移植>>
- 11) <http://www.cnblogs.com/CodeHXX/archive/2011/07/27/2117659.html>
- 12) <http://www.cnblogs.com/LikeMelody/archive/2008/11/12/1332043.html>
- 13) http://blog.sina.com.cn/s/blog_62e225c30100ku6y.html
- 14) <<SD 卡详细中文资料>>
- 15) <<sd Specification_Ver2.0>>
- 16) ehci 1.0 规范
- 17) usb 2.0 规范
- 18) Mass_Storage_Specification_Overview_v1.4_2-19-2010
- 19) usbmassbulk_10
- 20) <<usb 开发大全>>
- 21) AC97 规范
- 22)