## 实验一、STM32的开发环境与简单工程

### 一、实验目的

1、熟悉STM32开发板的开发环境；

2、熟悉MDK创建和配置STM32工程项目的基本流程；

3、熟悉STM32官方库的应用；

4、规范编程格式。

### 二、实验内容

本次实验配置MDK集成开发环境，新建一个简单的工程文件，添加STM32官方库并配置工程，编译运行这个工程文件。下载已经编译好的文件到开发板中运行。学会在程序中设置断点，观察系统内存和变量，为调试应用程序打下基础。

### 三、预备知识

基本单片机硬件知识、单片机软件编程语言、程序创建和调试的基本方法。

### 四、实验设备及工具

硬件：STM32开发平台

软件：STM32官方库；PC机操作系统Windows 98、Windows 2000或Windows XP；KEIL MDK集成开发环境；串口转usb驱动。

### 五、实验步骤

1、在准备存放工程文件的目录下创建一新文件夹，命名为Proj\_GPIO；在Proj\_GPIO文件夹里面分别再创建四个文件夹：CMSIS、USER、LIB、OBJ。如图1。

其中CMSIS（Cortex Microcontroller Software Interface Standard）用于存放Cortex-M 处理器系列的与供应商无关的软件抽象层和启动相关的代码文件；

USER用于存放我们自己编写的代码文件（含自己移植的底层驱动），还有MDK工程；

LIB存放所有的官方底层驱动库文件；

OBJ用于工程输出的过程文件和最终的二进制文件。

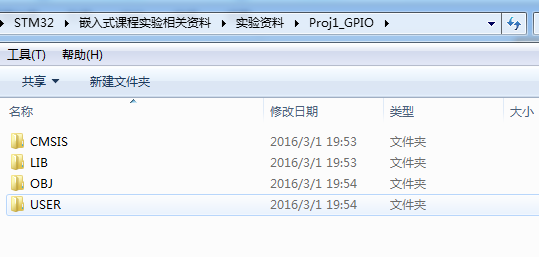


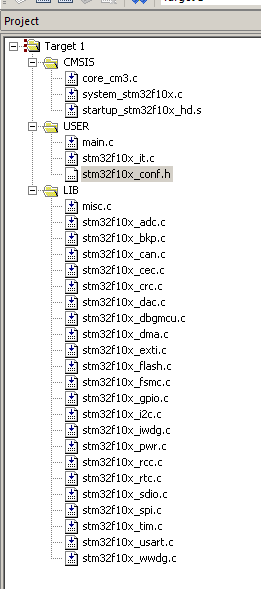
图1

2、将官方库STM32F10x\_StdPeriph\_Lib\_V3.5.0.rar解压。

1）把STM32F10x\_StdPeriph\_Lib\_V3.5.0\Libraries\CMSIS\CM3\CoreSupport下的所有文件和STM32F10x\_StdPeriph\_Lib\_V3.5.0\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x下的所有文件都到第一步所创建的CMSIS文件夹中；

2）把STM32F10x\_StdPeriph\_Lib\_V3.5.0\Libraries\STM32F10x\_StdPeriph\_Driver目录下的文件（目录inc和scr）复制到第一步创建的LIB文件夹中；

3)把STM32F10x\_StdPeriph\_Lib\_V3.5.0\Project\STM32F10x\_StdPeriph\_Template目录下的stm32f10x\_conf.h、stm32f10x\_it.c、stm32f10x\_it.h三个文件复制到USER文件夹中。

3、打开MDK软件，新建一个工程Proj\_GPIO保存到Proj\_GPIO/USER中。CPU选择STM32F103ZE，如图2；

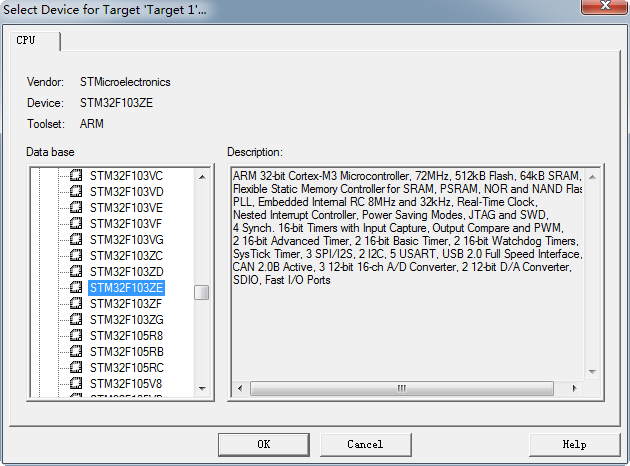


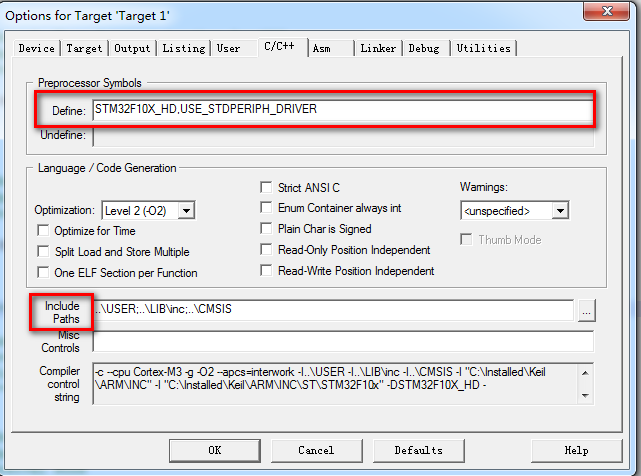
图2

4、新建一个空文档main.c保存到USER中，然后根据绝对路径将文件对应添加到工程中，如右图。

5、配置工程属性，右键点击工程文件中的Target 1选择Options for Target ‘Target 1’打开工程选项对话框。做如下修改：

1）Output选项勾选Create HEX File，然后点击Select Folder for Objects按钮定位输出文件保存目录到工程的OBJ文件；

2）Listing选项，同样点击Select Folder for Listings定位输出文件保存目录到工程的OBJ文件；



3）C/C++选项，Define中填入STM32F10X\_HD, USE\_STDPERIPH\_DRIVER系统的两个基本宏定义；配置Include Paths属性，加入工程中包含头文件的目录；如右图

4）后面Debug和Utilities两个选项如果使用J-link或者其他调试器则需要做对应的修改，否则不用。

6、根据实验任务硬件原理图编写代码实现四个按键控制led灯亮灭。

1）其中用户按键和LED原理图如下：

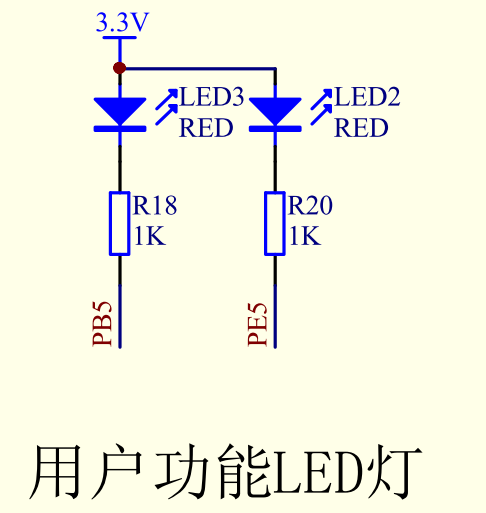
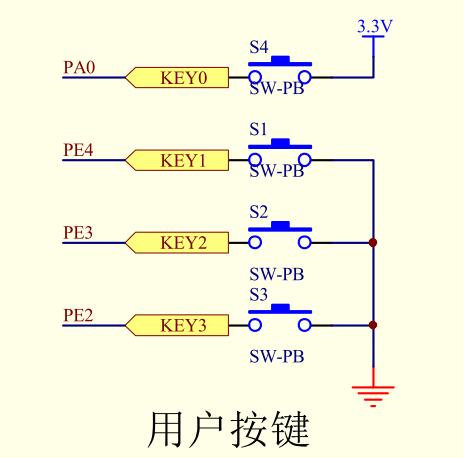


图5

2）根据原理图初始化各IO引脚，编写控制函数，然后主函数中调用。

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* 函数名称：io\_init**

**\* 函数功能：初始化PA0 PE2 PE3 PE4为输入口PE5 PB5为输出口**

**\* 参数: 无**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void io\_init()**

**{**

**GPIO\_InitTypeDef gpin\_init\_struct;**

**//使能每个IO口的时钟**

**RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE);**

**RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB, ENABLE);**

**RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOE, ENABLE);**

**gpin\_init\_struct.GPIO\_Pin = GPIO\_Pin\_0; //PA0引脚**

**gpin\_init\_struct.GPIO\_Speed = GPIO\_Speed\_2MHz; //IO速度为2MHz**

**gpin\_init\_struct.GPIO\_Mode = GPIO\_Mode\_IPD; //下拉模式**

**GPIO\_Init(GPIOA, &gpin\_init\_struct); //A口按键**

**gpin\_init\_struct.GPIO\_Pin = GPIO\_Pin\_5;**

**gpin\_init\_struct.GPIO\_Speed = GPIO\_Speed\_2MHz;**

**gpin\_init\_struct.GPIO\_Mode = GPIO\_Mode\_Out\_PP;**

**GPIO\_Init(GPIOB, &gpin\_init\_struct); //B口led1**

**GPIO\_Init(GPIOE, &gpin\_init\_struct); //E口led2**

**gpin\_init\_struct.GPIO\_Pin = GPIO\_Pin\_2|GPIO\_Pin\_3|GPIO\_Pin\_4;**

**gpin\_init\_struct.GPIO\_Speed = GPIO\_Speed\_2MHz;**

**gpin\_init\_struct.GPIO\_Mode = GPIO\_Mode\_IPU; //上拉模式**

**GPIO\_Init(GPIOE, &gpin\_init\_struct); //E口按键**

**}**

、

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* 函数名称: key\_led**

**\* 函数功能: 根据按键情况控制led灯的亮灭 s1 s2控制led1；s3 s4控制led2**

**\* 参数: 无**

**\* 其他：其中LED1\_ON LED1\_OFF LED2\_ON LED2\_OFF为全局宏定义**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void key\_led()**

**{**

**if(GPIO\_ReadInputDataBit(GPIOE, GPIO\_Pin\_4)==0) LED1\_ON;**

**if(GPIO\_ReadInputDataBit(GPIOE, GPIO\_Pin\_3)==0) LED1\_OFF;**

**if(GPIO\_ReadInputDataBit(GPIOE, GPIO\_Pin\_2)==0) LED2\_ON;**

**if(GPIO\_ReadInputDataBit(GPIOA, GPIO\_Pin\_0)==1) LED2\_OFF;**

**}**

**#include "stm32f10x.h"**

**#define LED2\_ON GPIO\_ResetBits(GPIOB, GPIO\_Pin\_5)**

**#define LED2\_OFF GPIO\_SetBits(GPIOB, GPIO\_Pin\_5)**

**#define LED1\_ON GPIO\_ResetBits(GPIOE, GPIO\_Pin\_5)**

**#define LED1\_OFF GPIO\_SetBits(GPIOE, GPIO\_Pin\_5)**

**void io\_init();**

**void key\_led();**

**int main()**

**{**

**io\_init();**

**while(1)**

**{**

**key\_led();**

**}**

**}**

3）代码编写完成后编译链接，通过下载软件将hex文件下载到实验板中运行验证。

## 实验二、ucos操作系统的移植

### 一、实验目的

1、掌握ucos到STM32的移植方法；

2、熟悉通过C语音处理底层寄存器的技巧；

3、了解STM32在ucos任务切换时的处理；

4、规范编程格式。

### 二、实验内容

本次实验通过用C语言编写6个操作系统相关函数和用汇编语言编写4个与处理器相关的函数，将实时操作系统µC/OS-Ⅱ移植到芯片STM32F103ZE中，并创建两个用户任务点亮led。学会操作系统的移植方法，深入掌握嵌入式操作系统µC/OS-Ⅱ在初始化任务和任务调度的实现方法，直观了解实时操作系统的体系结构和设计思想。

### 三、预备知识

基本单片机硬件知识、单片机软件编程语言、程序创建和调试的基本方法、ucos操作系统的任务调度原理、STM32的寄存器架构。

### 四、实验设备及工具

硬件：STM32开发平台

软件：STM32官方库；PC机操作系统Windows 98、Windows 2000或Windows XP；KEIL MDK集成开发环境；串口转usb驱动。

### 五、实验步骤

1、按实验一的步骤新建一个工程Proj2\_ucos。

2、将移植好的ucos源代码整个拷贝到我们的新建的工程目录中，然后按对应的文件路径添加到keil工程里（os\_dbg.c和os\_dbg\_r.c只添加一个），如图6和图7。

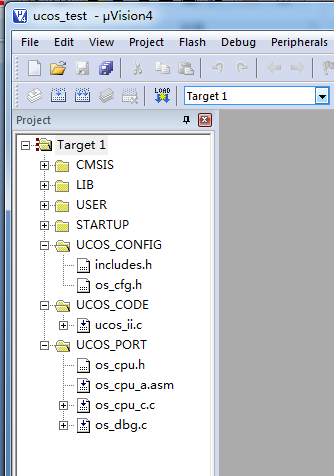
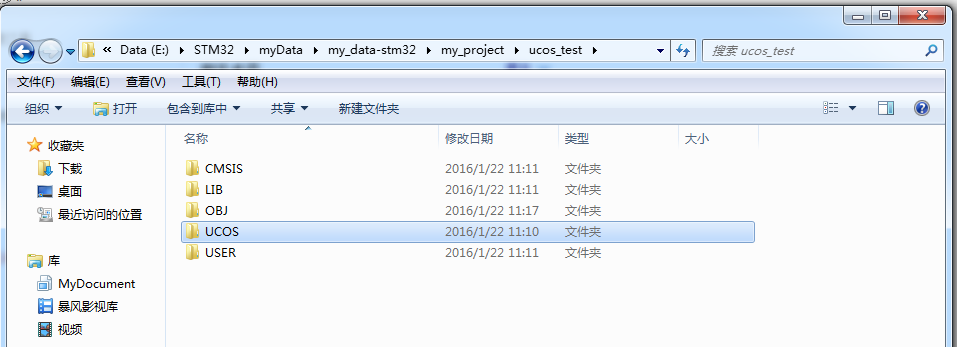


图6 图7

3、配置工程相关属性（与实验一库应用配置一样），记得将ucos的头文件路径添加到工程的include path上面去，找到stm2f10x\_it.c的PendSV\_Handler函数注释掉，main文件中将includes.h包含进来，主函数为空，编译通过。

4、在main.c中建立sys\_init()函数编写操作系统运行需要的硬件相关初始化（见代码），然后修改stm32f10x\_it.c中的系统心跳中断函数并加入头文件includes.h（见代码，）。

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* 函数名称：sys\_init**

**\* 函数功能：嵌套向量中断优先级分组，时钟心跳定时初始化1/200 s**

**\* 参数: 无**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void sys\_init()**

**{**

**NVIC\_PriorityGroupConfig(NVIC\_PriorityGroup\_2);**

**SysTick\_Config(SystemCoreClock/200);**

**}**

**/\*\***

**\* @brief This function handles SysTick Handler.**

**\* @param None**

**\* @retval None**

**\*/**

**void SysTick\_Handler(void)**

**{**

**OSIntEnter();**

**OSTimeTick();**

**OSIntExit();**

**}**

5、编写用户任务函数和相关的初始化。

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* 函数名称: led\_init**

**\* 函数功能: 初始化LED**

**\* 参 数:**

**\* 其 他：**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void led\_init()**

**{**

**GPIO\_InitTypeDef gpin\_init\_struct;**

**RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB|RCC\_APB2Periph\_GPIOE, ENABLE);**

**gpin\_init\_struct.GPIO\_Pin = GPIO\_Pin\_5;**

**gpin\_init\_struct.GPIO\_Speed = GPIO\_Speed\_2MHz;**

**gpin\_init\_struct.GPIO\_Mode = GPIO\_Mode\_Out\_PP;**

**GPIO\_Init(GPIOB, &gpin\_init\_struct); //B口led1**

**GPIO\_Init(GPIOE, &gpin\_init\_struct); //E口led2**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* 函数名称: my\_task**

**\* 函数功能: 用户任务**

**\* 参 数:**

**\* 其 他：**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void my\_task()**

**{**

**sys\_init();**

**led\_init();**

**OSStatInit();**

**while(1)**

**{**

**LED1\_ON;**

**LED2\_OFF;**

**OSTimeDly(50);**

**LED1\_OFF;**

**LED2\_ON;**

**OSTimeDly(100);**

**}**

**}**

6、最后在主函数中创建任务并启动操作系统。编译并调试运行。

**#include "stm32f10x.h"**

**#include "includes.h"**

**#define LED2\_ON GPIO\_ResetBits(GPIOB, GPIO\_Pin\_5)**

**#define LED2\_OFF GPIO\_SetBits(GPIOB, GPIO\_Pin\_5)**

**#define LED1\_ON GPIO\_ResetBits(GPIOE, GPIO\_Pin\_5)**

**#define LED1\_OFF GPIO\_SetBits(GPIOE, GPIO\_Pin\_5)**

**OS\_STK task\_stk[128];**

**void my\_task(void);**

**void led\_init(void);**

**void sys\_init(void);**

**int main()**

**{**

**OSInit();**

**OSTaskCreate(my\_task, (void \*)0, (OS\_STK \*)&task\_stk[127], 10);**

**OSStart();**

**while(1);**

**}**

## 实验三、ucos操作系统的任务通信

### 一、实验目的

1、掌握ucos任务间通信的各种方法；

2、了解嵌入式操作系统任务间可能出现的资源冲突；

3、掌握ucos中信号量和邮箱的应用；

4、规范编程格式。

### 二、实验内容

本次实验通过用C语言编写基于ucos操作系统下的多个用户任务，并且实现任务间的正确通信。通过信号量处理公共资源的等待和使用，通过邮箱实现任务间的多个数据变量的传递。建立正确的操作系统编程理念，深入掌握µC/OS-Ⅱ中信号量与邮箱的创建和使用，了解各任务间通信方法的大概原理及优缺点。

### 三、预备知识

基本单片机硬件知识、单片机软件编程语言、程序创建和调试的基本方法、ucos操作系统的任务调度原理。

### 四、实验设备及工具

硬件：STM32开发平台。

软件：STM32官方库；PC机操作系统Windows 98、Windows 2000或Windows XP；KEIL MDK集成开发环境；串口转usb驱动。

### 五、实验步骤

1、建立基于操作系统µC/OS-Ⅱ的工程Proj3\_ucos1，并配置好工程各项属性。

2、创建用户任务文件task.c和task.h两个文件，详细代码如下。

**//task.c文件代码**

**#include "task.h"**

**INT32U NUM\_COUNT1;**

**INT32U NUM\_COUNT2;**

**void test\_sem(void);**

**void my\_task()**

**{**

**while(1)**

**{**

**test\_sem();**

**}**

**}**

**void test\_sem()**

**{**

**INT8U sem\_err;**

**INT32U tmp;**

**tmp = NUM\_COUNT1;**

**OSTimeDlyHMSM(0,0,0,50);//用延时代替任务的其他任务**

**tmp++;**

**NUM\_COUNT1 = tmp;**

**OSSemPend(NUM\_COUNT, 500, &sem\_err);**

**tmp = NUM\_COUNT2;**

**OSTimeDlyHMSM(0,0,0,50);//用延时代替任务的其他任务**

**tmp++;**

**NUM\_COUNT2 = tmp;**

**OSSemPost(NUM\_COUNT);**

**}**

**//task.h文件代码**

**#ifndef \_MY\_TASK\_H**

**#define \_MY\_TASK\_H**

**#include "includes.h"**

**extern OS\_EVENT \*NUM\_COUNT;**

**extern void my\_task(void);**

**#endif**

代码中，在task1.c中定义两个全局变量NUM\_COUNT1和NUM\_COUNT2，两个变量在my\_task函数里用于记录该任务循环的总次数，区别在与NUM\_COUNT1直接操作，而NUM\_COUNT2则通过信号量的方法进行操作。

3、在main函数中初始化操作系统后创建一个信号量NUM\_COUNT，然后用OSTaskCreate创建两个任务，优先级分别是8和9，两个任务的入口地址都是task1.c文件中的my\_task函数。

**NUM\_COUNT = OSSemCreate(1); //创建一个信号量**

**OSTaskCreate(my\_task, (void \*)0, (OS\_STK \*)&task1\_stk[SIZE\_STK-1], 9);**

**OSTaskCreate(my\_task, (void \*)0, (OS\_STK \*)&task2\_stk[SIZE\_STK-1], 8);**

4、编译通过后调试，在debug界面中，将NUM\_COUNT1，NUM\_COUNT2两个全局变量加入到变量观察窗口watch1中，然后在task.c的my\_task函数里面设置断点，运行并观察两个变量的变化，理解信号量的使用。

5、关于操作系统邮箱的应用由例子工程Proj3\_ucos2实现，请自行理解代码。

## 实验四、ucos操作系统的文件系统实现

### 一、实验目的

1、了解文件系统FATFS的基本实现原理；

2、掌握FATFS的移植步骤；

3、掌握基于ucos中FATFS的应用；

4、掌握STM32对SD卡的读写操作；

5、规范编程格式。

### 二、实验内容

本实验实现将FAT文件系统移植到ucos中，并基于ucos的文件系统编写用户任务进行测试，通过对文件系统的移植实验深入掌握对嵌入式相关驱动移植的基本步骤，建立正确的嵌入式操作系统驱动移植理念，并且熟悉掌握ucos操作系统下的文件系统的基本应用，了解其优缺点。

### 三、预备知识

基本单片机硬件知识、单片机软件编程语言、程序创建和调试的基本方法、ucos操作系统的任务调度原理、STM32单片机中的SD卡驱动、文件系统的操作函数、FAT文件系统的基础知识。

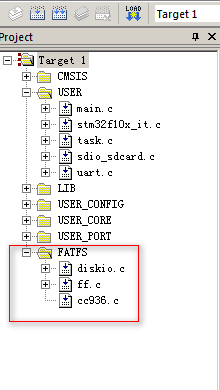
### 四、实验设备及工具

硬件：STM32开发平台。

软件： PC机操作系统Windows 98、Windows 2000或Windows XP；KEIL MDK集成开发环境；串口转usb驱动；STM32官方库；FATFS的官方库；STM32的uart驱动和SD卡驱动源码。

### 五、实验步骤

1、拷贝实验三的所有代码到新的目录地下，删除USER和OBJ文件夹里工程相关的文件，重新建立实验四的工程并命名为Proj4\_FATFS，配置好工程各项属性。

2、在USER文件夹里面创建文件夹myapp，将串口驱动uart和SD卡驱动sdio\_sdcard的源代码拷贝到该文件夹里面，并添加到工程中USER。

3、拷贝FAT文件系统源码到myapp文件夹。在keil的工程中增加一个文件夹FATFS并添加diskio.c、ff.c、cc936.c三个文件到里面，如右图。

4、尝试编译字符表文件cc936.c可能发现出错：#error directive: This file is not needed in current configuration. 这是因为在ff.h中\_CODE\_PAGE这个宏的定义值不是936和长文件名宏定义\_USE\_LFN的值不是1导致的。修改成对应的值后该文件的编译即可通过（如果用其他字符表文件同理）。

5、移植FAT文件系统需要做的事情。

1）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。FATFS源代码的获取，可以到官网下载<http://elm-chan.org/fsw/ff/00index_e.html>； 解压文件会得到两个文件夹，一个是doc文件夹，保存FATFS的一些使用文档和说明。另一个是src文件夹，保存代码源文件。FAT文件系统源码和sd卡读写的底层驱动。

2）移植FAT文件系统只需根据硬件中可能接入的物理存储模块对文件系统的底层IO驱动（diskio.c）进行修改，具体需要修改的函数有：

**DSTATUS disk\_initialize (BYTE);**//存储介质的初始化

**DSTATUS disk\_status (BYTE);**//读取存储介质的状态

**DRESULT disk\_read (BYTE, BYTE\*, DWORD, BYTE);**//读取数据

**DRESULT disk\_write (BYTE, const BYTE\*, DWORD, BYTE);**//写入数据

**DRESULT disk\_ioctl (BYTE, BYTE, void\*);**//读取存储介质文件系统相关信息

**DWORD get\_fattime (void);**//返回文件系统的时间

3）diskio.c文件中添加头文件sdio\_sdcard.h、ff.h，增加SD卡的驱动号0如下：

**/\* Correspondence between physical drive number and physical drive. \*/**

**#define SD\_CARD 0**

**#define ATA 1**

**#define MMC 2**

**#define USB 3**

4）diskio.c五个函数具体代码：

**DSTATUS disk\_initialize (**

**BYTE drv /\* Physical drive nmuber (0..) \*/**

**)**

**{**

**switch(drv)**

**{**

**case SD\_CARD://SD卡**

**if(SD\_Init() == SD\_OK)**

**{**

**return RES\_OK;**

**}**

**else**

**{**

**return RES\_NOTRDY;**

**}**

**}**

**return RES\_NOTRDY;**

**}**

**DSTATUS disk\_status (**

**BYTE drv /\* Physical drive nmuber (0..) \*/**

**)**

**{**

**switch(drv)**

**{**

**case SD\_CARD://SD卡**

**if(SD\_GetStatus() == SD\_TRANSFER\_ERROR)**

**{**

**return STA\_NOINIT;**

**}**

**else**

**{**

**return STA\_PROTECT;**

**}**

**default:**

**return STA\_NODISK;**

**}**

**}**

**DRESULT disk\_read (**

**BYTE drv, /\* Physical drive nmuber (0..) \*/**

**BYTE \*buff, /\* Data buffer to store read data \*/**

**DWORD sector, /\* Sector address (LBA) \*/**

**BYTE count /\* Number of sectors to read (1..255) \*/**

**)**

**{**

**switch (drv) {**

**case SD\_CARD :**

**for(;count>0;count--)**

**{**

**if(SD\_ReadBlock(buff, sector<< 9, \_MAX\_SS) != SD\_OK)**

**{**

**break;**

**}**

**sector++;**

**buff+=\_MAX\_SS;**

**/\* Wait until end of DMA transfer \*/**

**SD\_WaitReadOperation();**

**/\* Check if the Transfer is finished \*/**

**while(SD\_GetStatus() != SD\_TRANSFER\_OK);**

**}**

**if(count == 0)**

**{**

**return RES\_OK;**

**}else{**

**return RES\_ERROR;**

**}**

**default:**

**return RES\_NOTRDY;**

**}**

**}**

**#if \_READONLY == 0**

**DRESULT disk\_write (**

**BYTE drv, /\* Physical drive nmuber (0..) \*/**

**const BYTE \*buff, /\* Data to be written \*/**

**DWORD sector, /\* Sector address (LBA) \*/**

**BYTE count /\* Number of sectors to write (1..255) \*/**

**)**

**{**

**switch (drv) {**

**case SD\_CARD :**

**for(;count>0;count--)**

**{**

**if(SD\_WriteBlock((uint8\_t \*)buff,sector<<9, \_MAX\_SS) != SD\_OK)**

**{**

**break;**

**}**

**sector++;**

**buff+=\_MAX\_SS;**

**/\* Wait until end of DMA transfer \*/**

**SD\_WaitWriteOperation();**

**/\* Check if the Transfer is finished \*/**

**while(SD\_GetStatus() != SD\_TRANSFER\_OK);**

**}**

**if(count == 0)**

**{**

**return RES\_OK;**

**}else{**

**return RES\_ERROR;**

**}**

**default:**

**return RES\_NOTRDY;**

**}**

**}**

**#endif /\* \_READONLY \*/**

**// disk\_ioctl 和 get\_fattime 两个函数暂时可以不实现 直接返回**

**DRESULT disk\_ioctl (**

**BYTE drv, /\* Physical drive nmuber (0..) \*/**

**BYTE ctrl, /\* Control code \*/**

**void \*buff /\* Buffer to send/receive control data \*/**

**)**

**{**

**return RES\_NOTRDY;**

**}**

**#if !\_FS\_READONLY**

**/\* 31-25: Year(0-127 org.1980), 24-21: Month(1-12), 20-16: Day(1-31) \*/**

**/\* 15-11: Hour(0-23), 10-5: Minute(0-59), 4-0: Second(0-29 \*2) \*/**

**DWORD get\_fattime (void)**

**{**

**return 0;**

**}**

**#endif**

5）对于最底层（物理层）对SD卡的具体初始化和读写操作，STM32有SPI和SDIO两种方式实现，实际应用中根据硬件电路条件选择。在没有SPI外设的单片机中也可以通过IO口模拟SPI协议。（具体实现代码略，详见工程文件中的 sdio\_sdcard.c 和 sdio\_sdcard.h ）

6）修改task.c进行文件系统的测试应用。在测试FAT文件系统之前，首先应该确保对底层SD卡的读写驱动正确。所以第一步通过直接调用SD卡驱动函数实现对SD卡进行初始化检测和基本的读写操作，具体代码如下：

**void test\_SD()**

**{**

**SD\_CardInfo SDCardInfo;**

**INT16U i;**

**printf("SD卡驱动函数测试程序……\n\r");**

**while(SD\_Init()) //内存卡检测错误**

**{**

**printf("内存卡检测错误\n\r");**

**OSTimeDly(200);**

**}**

**printf("内存卡检测成功！\n\r");**

**SD\_GetCardInfo(&SDCardInfo);**

**printf("sdcard size is: %u\n\r", SDCardInfo.CardCapacity>>22);**

**if(SD\_ReadBlock(RxBuffer, 0, 512) == SD\_OK)**

**{**

**for(i=0; i<512; i++)**

**{**

**printf("%X ", RxBuffer[i]);**

**}**

**}**

**}**

7）如果上一步能初始化并检测出SD卡的大小，读出前512个字节的数据，则说明SD卡和驱动函数都正常。下一步设计一个FAT测试函数：实现检测TEST.txt文件是否存在，存在则读取文件中的前100个字节内容并串口输出；然后打开TEST2.txt文件（不存在则创建）写入读取到的内容（没有内容则写入“is no file TEST.txt!”）；打开TEST2.txt读取文件中的前100个字节内容并串口输出。具体代码如下。

**void test\_FAT()**

**{**

**FATFS fs;**

**FIL f\_1, f\_2;**

**INT32U read\_count=21;**

**FRESULT error;**

**error = f\_mount(0, &fs); //挂载SD卡**

**error = f\_open(&f\_1, "TEST.txt", FA\_OPEN\_EXISTING|FA\_READ);**

**if(error==FR\_OK)**

**{**

**error = f\_read(&f\_1, RxBuffer, BUFFER\_SIZE, &read\_count);**

**printf("\n\rread the file: TEST.txt \n\r");**

**printf("read 100 bytes, then return %d bytes! \n\r", read\_count);**

**printf("the data is :\n\r");**

**printf((char\*)RxBuffer);**

**error = f\_close(&f\_1);**

**}**

**error = f\_open(&f\_2, "TEST2.txt", FA\_OPEN\_ALWAYS|FA\_WRITE);**

**printf("\n\rwrite test2.txt %d words!\n\r", read\_count);**

**error = f\_write(&f\_2, RxBuffer, read\_count, &read\_count);**

**if(error==FR\_OK)**

**{**

**printf("write OK ! %d words are writed~\n\r", read\_count);**

**}else{**

**printf("write error\n\r");**

**}**

**error = f\_close(&f\_2);**

**error = f\_open(&f\_1, "TEST2.txt", FA\_OPEN\_EXISTING|FA\_READ);**

**if(error==FR\_OK)**

**{**

**error = f\_read(&f\_1, RxBuffer, BUFFER\_SIZE, &read\_count);**

**printf("\n\rread the file: TEST2.txt \n\r");**

**printf("read 100 bytes, then return %d bytes! \n\r", read\_count);**

**printf("the data is :\n\r");**

**printf((char\*)RxBuffer);**

**error = f\_close(&f\_1);**

**}**

**}**

8）至此，我们已经将FATFS移植成功了，下面我们设计一个基于ucos的文件系统应用实验，实现在用户中检测是否有SD卡插入，如果有则任务没循环一次在SD卡中的文件mytask.txt的末尾写入一句“这是本任务的第xx次循环”，然后在主函数中创建两个以上的该任务的线程（这里需要注意实验三的资源冲突问题！）。

9）具体代码，详见实例工程！

\*扩展知识1：在本工程中的SD底层驱动中，因为用32位存储SD卡的容量大小，所以最大只能识别到4GB，如果采用大于4G的卡进行测试则读出的容量是错误的。

\*扩展知识2：对于FAT32的MBR区不一定是在0扇区中。

\*扩展知识3：配置ff.h（可能是ffconf.h）文件中 \_USE\_STRFUNC 宏定义为1，可以使用f\_putc、f\_puts、f\_printf、f\_gets、f\_eof、f\_error 等函数

## 实验五、基于ucos操作系统的网络模块

### 一、实验目的

1、了解W5500网络模块的基本实现原理；

2、掌握ucos下网络底层库的移植步骤；

3、掌握基于ucos的网络编程TCP、UDP思路；

4、了解嵌入式软件设计的分层思路；

### 二、实验内容

本实验通过STM32的SPI协议读写W5500网络模块，移植网络协议驱动到ucos嵌入式操作系统，并通过编写ucos操作系统的应用任务实现实验板和上位机的网络通信。了解基于网络通信的嵌入式软件设计的思路。

### 三、预备知识

本实验要求学生具备基本的C语言读写能力、基本单片机软硬件知识、了解SPI协议、了解基础的网络协议应用、ucos操作系统的用户任务设计能力

### 四、实验设备及工具

硬件：STM32开发平台、PC机

软件： keilMDK编程环境、串口转usb驱动、STM32官方库、移植好的ucos源码、基于STM32的SPI驱动库、基于STM32的uart驱动库、W5500网络模块驱动库、

### 五、实验步骤

1、了解网络编程中socket、UDP的基本实现原理。

2、了解W5500网络模块的基本功能和官方驱动库提供的代码架构。

W5500 是一款全硬件 TCP/IP 嵌入式以太网控制器，为嵌入式系统提供了更加简易的互联网连接方案。W5500 集成了 TCP/IP 协议栈，10/100M 以太网数据链路层（MAC） 及物理层（PHY），使得用户使用单芯片就能够在他们的应用中拓展网络连接。原厂公司位于韩国，中文官网：<http://www.iwiznet.cn/>



W5500的驱动可在官网中找到下载，下载解压后的目录Ethernet为W5500的官方驱动库。其中W5500.c和W5500.h提供W5500的硬件接口函数，实现对W5500寄存器的读写驱动；socket.c和socket.h提供网络协议socket的API实现和用户接口函数；wizchip\_conf.c和wizchip.h提供对芯片应用驱动和配置接口，例如芯片的复位、IP地址的初始化，注册用户的SIP驱动函数等。整体架构如右图：

3、基于STM32的移植。

由上图我们知道对于官方驱动库，我们需要提供STM32的SPI协议驱动接口。具体需要我们手动实现的函数是：

**wizchip\_cris\_enter** ：进入临界区的函数，可以不管

**wizchip\_cris\_exit** ：退出临界区的函数，也可以不管

**wizchip\_cs\_select** ：输出有效片选信号的函数，控制CS输出低电平的函数，必须实现

**wizchip\_cs\_deselect** ：控制CS输出高电平的函数，必须实现

**wizchip\_bus\_readbyte** ：SPI总线读取一字节数据函数，必须实现

**wizchip\_bus\_writebyte** ：SPI总线写一字节数据函数，必须实现

正常情况下我们需要找到驱动库中对应的函数原型进行修改，但是因为官方库中提供了一个用户接口，可以将我们已有对应的SPI函数入口注册并替换掉上述函数，所以我们可以独立编写文件SPI.c和SPI.h，然后初始化时调用“注册函数”（注册函数在wizchip）。

工程中我们创建文件confForW5500.c和confFroW5500.h实现对W5500的配置和各项初始化，代码详见工程文件。

4、用户测试任务。

根据网络编程基础知识，我们编写一个回环测试任务，对网络模块的TCP/IP功能进行基础测试。具体分为实现socket功能和UDP两个功能，流程图如下：



根据上述流程图，我们创建文件loopback.c和loopback.h实现对应的各项功能函数。然后基于ucos操作系统下建立用户任务my\_task() 进行各项功能的顺序调用，主函数中启动操作系统并创建该任务。

5、上位机连接测试。

将工程编译调试通过并烧写到实验板，用网线连接PC机和W5500网络模块，实验板上电启动，根据我们编写的用户任务STM32将会启动一个SOCKET服务端并绑定实验板的IP和程序中配置的端口。

PC上机设置各项网络参数，让PC机和实验板处于同一网段（局域网），然后打开命令提示符窗口CMD，输入ping命令：ping 192.168.xxx.xxx （实验板IP），观察是否能够收到实验板反馈的数据信息。如下图：

PC机打开调试软件NetAssist.exe，选择TCP Client，服务器IP地址设置成实验板IP，服务端口设置成实验板中打开的socket端口，点击连接看能否成功连接实验板socket服务，连接成功后在软件的发送数据区随便输入发送内容，点击发送，查看接收到的反馈信息，如下图

6、自行设计编写应用任务，实现实验板网络方面新的需求。