

Cortex-M4原理与实践实验报告

题目： MicroSD 卡操作实验

**作者姓名 林宇航**

**指导教师**  **郭方洪**

**专业班级 自动化1901**

**学 院 信息工程学院**

**提交日期** 2021年12月14日

**一、实验目的**

1. 了解 FAT 文件系统原理。

2. 了解 MicroSD 卡的原理与控制方式。

3. 学习 TM4C129x Series Cortex-M4 的同步串口 QSSI 操作方式。

4. 学习 QSSI 相关库函数的使用。

**二、实验设备**

1.计算机一台，操作系统为WindosXP或Windos7，装有CCSv11.0软件。

2.EK-TM4C1294XL实验开发板一块。

3.USB连接线一条。

**三、实验原理和流程**

**1. FAT 文件系统原理简介**

FAT 文件起源于 70 年代末 80 年代初，用于微软的 MS-DOS 操作系统。它开始被设计 成一个简单的文件系统用于小于 500K 的软件盘。后来被功能被大大增强用于支持越来越大 的媒质。现在的文件系统有 FAT12，FAT16 和 FAT32 三种子类。本实验所用的 FatFs 是用于 小型嵌入式系统的通用 FAT 文件系统。FatFs 服从 ANSIC 标准，并且完全和磁盘 IO 脱离。 能编入资源有限的小型微处理器，像 AVR，8051，PIC，ARM 等等。

FatFs 模块拥有以下主要特性：

 兼容 Windows 的 FAT 文件系统；

 包括 DBCS 在内的多种 ANSI/OEM 字符集和字符编码方案；

 在 ANSI/OEM 或 Unicode 下，支持长文件名；

 支持实时时钟；

 支持多种扇区大小。

图示

描述已自动生成FatFs 模块是一个文件系统层，完全和诸如存储卡，硬盘以及其他任何种类的存储器件 分离下图 1 显示了文件层和其他层之间的关系。

图1 文件层和其他层关系

**2. MicroSD 卡的原理与控制方式简介**

Micro SD Card 原名 TF 卡（Trans-flash Card）2004 年正式更名为 Micro SD Card，由 SanDisk（闪迪）公司发明，被广泛应用于手机，PDA，数码相机，MP3 等等移动电子产品。 MicroSD 卡体积小，容量大，相比传统 Flash 存储元件，MicroSD 卡在设计与制造上有诸多优势，例如：

相同的尺寸大小，可以有多种不同的容量；

采用片上控制器，不需要主机的内存管理，并且将软件开发从 ECC 和磨损均衡中分离；

无缝内存组件技术的转变；

采用行业标准串口，将风险最小化（增加向下兼容）；

空前的可升级性和灵活性；

节省片上资源。

MicroSD 卡有 8 个引脚，2 种工作方式，分别为 SD 模式，和 SPI 模式，本实验采用相 对简单的 SPI 工作模式，电路原理图如下图 2 所示。SD 卡可以通过 SPI 接口或者 GPIO 模 拟 SPI 方式和大多数微控制器相连，适用于低开销嵌入式应用。在 SPI 模式下，信号线上的 数据方向是固定的，并且数据以字节为单位传输。从主机到 TF 卡的命令帧是一个固定长度 的数据包，占 6 个字节，如下图 3 所示，一般叫做 CMDx 或 ACMDx。当将 D0 置高，TF 卡准备好接收命令帧，接收完成后，从卡返回一个应答信号。命令参数占 4 个字节，不是所 有的命令都有参数。每次发送完一个命令后，SD 卡都会有回应，回应有多种格式，1 字节 的 R1，2 字节的 R2 等，一般在 SPI 模式下，我们只用到 R1。当 R1 返回的值为 0x00 时， 说明操作成功，否则，出现对应的错误，R1 对应日程表

描述已自动生成错误如下图 4 所示。

图2图示, 示意图

描述已自动生成 SD卡电脑原理图

图3图示

低可信度描述已自动生成 SD命令帧格式

图4 R1返回对应错误

对 SD 卡操作要注意几点：

1) SD 卡的 SPI 总线，在读入数据时 SD 卡的 SPI 是 CLK 的上升沿输入锁存，输出数 据也是在上升沿；

2) 写 CMD 命令过程为先拉低 CS 使能 SD 卡，其次在 SD 卡的 DIN 写入指令，并附 加 8 个填充时钟，使 SD 卡完成内部操作，之后在 SD 卡的 DOUT 上接受回应，回 应接受完毕使 CS 为低电平，再附加 8 个填充时钟；

3) 在 SD 卡的 DIN 没有数据写入时，应使 DIN 保持高电平。

**3. TM4C129x Series Cortex-M4 的同步串口 QSSI 工作原理与方式**

1294 有 4 个同步串行接口(QSSI, Quad-Synchronous Serial Interface)模块，所有 4 个模块 都支持单 SSI，双 SSI 接口，也支持 4SSI 接口增强，以提供更快的数据吞吐能力。QSSI 模 块作为与外设同步串口通信的主或从接口，例如 Freescale SPI，或者 TI 同步串行接口。发送 和接收采用内部独立 FIFO 缓冲器，允许存储最多 8 个 16 位的值。CPU 能获取这些数据， 也能获取 QSSI 的控制和状态信息。QSSI 模块也提供了 DMA 接口，允许在 uDMA 模式下 设置发送和接收 FIFO 作为源或目的地址。

QSSI 模块有以下主要特性：

 4 路 QSSI 通道，都支持单 SSI，双 SSI 和 4SSI 功能；

 可编程的时钟频率和预分频器；

 数据帧大小从 4 位到 16 位可编程；

 标准 FIFO 中断和发送结束中断；

 采用 μDMA 控制器高效地传输数据；

输出时钟频率定义为：SSInClk = SysClk / (CPSDVSR ✕ (1 + SCR))，其中，SysCLK 或 ALTCLK 作为 SSInCLK 的时钟源，CPSDVSR 为预分频寄存器（SSICPSR）中的值，从 2 到 254，SCR 是 SSI 控制寄存器 0（SSICR0）中的值，从 0 到 255。图 5 为 SSI 模块框图。

图5图示, 示意图

描述已自动生成 SSI模块框图

**4. 实验用到的主要库函数简介**

void SSIConfigSetExpClk(uint32\_t ui32Base,uint32\_t ui32SSIClk,

uint32\_t ui32Protocol,uint32\_t ui32Mode,

uint32\_t ui32BitRate，

uint32\_t ui32DataWidth)

函数用于配置 SSI 的工作模式，ui32Base 为 SSI 模块的基地址；ui32SSIClk 是 SSI 模块的时钟频率；ui32Protocol 指定传输协议例如 SSI\_FRF\_MOTO\_MODE\_0、SSI\_FRF\_TI； ui32Mode 指定 SSI 模块工作于主模式还是从模式；ui32BitRate 指定传输的时钟频率； ui32DataWidth 指定传输数据帧的数据宽度，可以为 4~16 中间的数字。

void SSIEnable(uint32\_t ui32Base)函数用于使能 SSI 模块，ui32Base 为 SSI 模块的基地址。

void SSIDataPut(uint32\_t ui32Base,uint32\_t ui32Data)函数用于输出一个数据到 FIFO。 ui32Base 为 SSI 模块的基地址；ui32Data 为需要输出的数据。

void SSIDataGet(uint32\_t ui32Base,uint32\_t \*pui32Data)函数用于从接收 FIFO 中获取一 个数据。ui32Base 为 SSI 模块的基地址；\*pui32Data 为指向数据接收的地址。

**5. 简化的 SSI 模块配置步骤**



**流程图：**



**四、实验代码、注释**

**实验代码：**

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

//

// sd\_card.c - Example program for reading files from an SD card.

//

// Copyright (c) 2011 Texas Instruments Incorporated. All rights reserved.

// Software License Agreement

//

// Texas Instruments (TI) is supplying this software for use solely and

// exclusively on TI's microcontroller products. The software is owned by

// TI and/or its suppliers, and is protected under applicable copyright

// laws. You may not combine this software with "viral" open-source

// software in order to form a larger program.

//

// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.

// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT

// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY

// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL

// DAMAGES, FOR ANY REASON WHATSOEVER.

//

// This is part of revision 8049 of the EK-LM4F232 Firmware Package.

//

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

#include <string.h>

#include <stdint.h>

#include <stdbool.h>

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "driverlib/fpu.h"

#include "driverlib/gpio.h"

#include "driverlib/interrupt.h"

#include "driverlib/rom.h"

#include "driverlib/sysctl.h"

#include "driverlib/systick.h"

#include "driverlib/pin\_map.h"

#include "grlib/grlib.h"

#include "utils/cmdline.h"

#include "utils/uartstdio.h"

#include "third\_party/fatfs/src/ff.h"

#include "third\_party/fatfs/src/diskio.h"

#include "driverlib/ssi.h"

#include "driverlib/rom\_map.h"

#include "TFTinit/TFT\_400x240\_OTM4001A\_16bit.h"

#include "EPIinit/EPIinit.h"

#ifndef M\_PI

#define M\_PI 3.14159265358979323846F

#endif

uint32\_t g\_ui32SysClock;

#define PATH\_BUF\_SIZE 80

#define CMD\_BUF\_SIZE 64

static char g\_cCwdBuf[PATH\_BUF\_SIZE] = "/";

static char g\_cTmpBuf[PATH\_BUF\_SIZE];

static char g\_cCmdBuf[CMD\_BUF\_SIZE];

static FATFS g\_sFatFs;

static DIR g\_sDirObject;

static FILINFO g\_sFileInfo;

static FIL g\_sFileObject;

typedef struct

{

FRESULT fresult;

char \*pcResultStr;

}

tFresultString;

#define FRESULT\_ENTRY(f) { (f), (#f) }

tFresultString g\_sFresultStrings[] =

{

FRESULT\_ENTRY(FR\_OK),

FRESULT\_ENTRY(FR\_DISK\_ERR),

FRESULT\_ENTRY(FR\_INT\_ERR),

FRESULT\_ENTRY(FR\_NOT\_READY),

FRESULT\_ENTRY(FR\_NO\_FILE),

FRESULT\_ENTRY(FR\_NO\_PATH),

FRESULT\_ENTRY(FR\_INVALID\_NAME),

FRESULT\_ENTRY(FR\_DENIED),

FRESULT\_ENTRY(FR\_EXIST),

FRESULT\_ENTRY(FR\_INVALID\_OBJECT),

FRESULT\_ENTRY(FR\_WRITE\_PROTECTED),

FRESULT\_ENTRY(FR\_INVALID\_DRIVE),

FRESULT\_ENTRY(FR\_NOT\_ENABLED),

FRESULT\_ENTRY(FR\_NO\_FILESYSTEM),

FRESULT\_ENTRY(FR\_MKFS\_ABORTED),

FRESULT\_ENTRY(FR\_TIMEOUT),

FRESULT\_ENTRY(FR\_LOCKED),

FRESULT\_ENTRY(FR\_NOT\_ENOUGH\_CORE),

FRESULT\_ENTRY(FR\_TOO\_MANY\_OPEN\_FILES),

FRESULT\_ENTRY(FR\_INVALID\_PARAMETER)

};

#define NUM\_FRESULT\_CODES (sizeof(g\_sFresultStrings) / sizeof(tFresultString))

tContext g\_sContext;

const char \*

StringFromFresult(FRESULT fresult)

{

unsigned int uIdx;

for(uIdx = 0; uIdx < NUM\_FRESULT\_CODES; uIdx++)

{

if(g\_sFresultStrings[uIdx].fresult == fresult)

{

return(g\_sFresultStrings[uIdx].pcResultStr);

}

}

return("UNKNOWN ERROR CODE");

}

void

SysTickHandler(void)

{

disk\_timerproc();

}

int

Cmd\_ls(int argc, char \*argv[])//显示文件列表

{

unsigned long ulTotalSize;

unsigned long ulFileCount;

unsigned long ulDirCount;

FRESULT fresult;

FATFS \*pFatFs;

fresult = f\_opendir(&g\_sDirObject, g\_cCwdBuf);

if(fresult != FR\_OK)

{

return(fresult);

}

ulTotalSize = 0;

ulFileCount = 0;

ulDirCount = 0;

UARTprintf("\n");

for(;;)

{

fresult = f\_readdir(&g\_sDirObject, &g\_sFileInfo);

if(fresult != FR\_OK)

{

return(fresult);

}

if(!g\_sFileInfo.fname[0])

{

break;

}

if(g\_sFileInfo.fattrib & AM\_DIR)

{

ulDirCount++;

}

else

{

ulFileCount++;

ulTotalSize += g\_sFileInfo.fsize;

}

UARTprintf("%c%c%c%c%c %u/%02u/%02u %02u:%02u %9u %s\n",

(g\_sFileInfo.fattrib & AM\_DIR) ? 'D' : '-',

(g\_sFileInfo.fattrib & AM\_RDO) ? 'R' : '-',

(g\_sFileInfo.fattrib & AM\_HID) ? 'H' : '-',

(g\_sFileInfo.fattrib & AM\_SYS) ? 'S' : '-',

(g\_sFileInfo.fattrib & AM\_ARC) ? 'A' : '-',

(g\_sFileInfo.fdate >> 9) + 1980,

(g\_sFileInfo.fdate >> 5) & 15,

g\_sFileInfo.fdate & 31,

(g\_sFileInfo.ftime >> 11),

(g\_sFileInfo.ftime >> 5) & 63,

g\_sFileInfo.fsize,

g\_sFileInfo.fname);

}

UARTprintf("\n%4u File(s),%10u bytes total\n%4u Dir(s)",

ulFileCount, ulTotalSize, ulDirCount);

fresult = f\_getfree("/", &ulTotalSize, &pFatFs);

if(fresult != FR\_OK)

{

return(fresult);

}

UARTprintf(", %10uK bytes free\n", ulTotalSize \* pFatFs->csize / 2);

return(0);

}

int

Cmd\_cd(int argc, char \*argv[])

{

unsigned int uIdx;

FRESULT fresult;

strcpy(g\_cTmpBuf, g\_cCwdBuf);

if(argv[1][0] == '/')

{

if(strlen(argv[1]) + 1 > sizeof(g\_cCwdBuf))

{

UARTprintf("Resulting path name is too long\n");

return(0);

}

else

{

strncpy(g\_cTmpBuf, argv[1], sizeof(g\_cTmpBuf));

}

}

else if(!strcmp(argv[1], ".."))

{

uIdx = strlen(g\_cTmpBuf) - 1;

while((g\_cTmpBuf[uIdx] != '/') && (uIdx > 1))

{

uIdx--;

}

g\_cTmpBuf[uIdx] = 0;

}

else

{

if(strlen(g\_cTmpBuf) + strlen(argv[1]) + 1 + 1 > sizeof(g\_cCwdBuf))

{

UARTprintf("Resulting path name is too long\n");

return(0);

}

else

{

if(strcmp(g\_cTmpBuf, "/"))

{

strcat(g\_cTmpBuf, "/");

}

strcat(g\_cTmpBuf, argv[1]);

}

}

fresult = f\_opendir(&g\_sDirObject, g\_cTmpBuf);

if(fresult != FR\_OK)

{

UARTprintf("cd: %s\n", g\_cTmpBuf);

return(fresult);

}

else

{

strncpy(g\_cCwdBuf, g\_cTmpBuf, sizeof(g\_cCwdBuf));

}

return(0);

}

int

Cmd\_pwd(int argc, char \*argv[])

{

UARTprintf("%s\n", g\_cCwdBuf);

return(0);

}

int

Cmd\_cat(int argc, char \*argv[])//读取文本文件的内容，打印到控制台

{

FRESULT fresult;

unsigned int usBytesRead;

if(strlen(g\_cCwdBuf) + strlen(argv[1]) + 1 + 1 > sizeof(g\_cTmpBuf))

{

UARTprintf("Resulting path name is too long\n");

return(0);

}

strcpy(g\_cTmpBuf, g\_cCwdBuf);

if(strcmp("/", g\_cCwdBuf))

{

strcat(g\_cTmpBuf, "/");

}

strcat(g\_cTmpBuf, argv[1]);

fresult = f\_open(&g\_sFileObject, g\_cTmpBuf, FA\_READ);

if(fresult != FR\_OK)

{

return(fresult);

}

do

{

fresult = f\_read(&g\_sFileObject, g\_cTmpBuf, sizeof(g\_cTmpBuf) - 1,

&usBytesRead);

if(fresult != FR\_OK)

{

UARTprintf("\n");

return(fresult);

}

g\_cTmpBuf[usBytesRead] = 0;

UARTprintf("%s", g\_cTmpBuf);

}

while(usBytesRead == sizeof(g\_cTmpBuf) - 1);

return(0);

}

int

Cmd\_help(int argc, char \*argv[])//帮助命令

{

tCmdLineEntry \*pEntry;

UARTprintf("\nAvailable commands\n");

UARTprintf("------------------\n");

pEntry = &g\_psCmdTable[0];

while(pEntry->pcCmd)

{

UARTprintf("%s%s\n", pEntry->pcCmd, pEntry->pcHelp);

pEntry++;

}

return(0);

}

tCmdLineEntry g\_psCmdTable[] =

{

{ "help", Cmd\_help, " : Display list of commands" },

{ "h", Cmd\_help, " : alias for help" },

{ "?", Cmd\_help, " : alias for help" },

{ "ls", Cmd\_ls, " : Display list of files" },

{ "chdir", Cmd\_cd, ": Change directory" },

{ "cd", Cmd\_cd, " : alias for chdir" },

{ "pwd", Cmd\_pwd, " : Show current working directory" },

{ "cat", Cmd\_cat, " : Show contents of a text file" },

{ 0, 0, 0 }

};

#ifdef DEBUG

void

\_\_error\_\_(char \*pcFilename, unsigned long ulLine)

{

}

#endif

void

ConfigureUART(void)

{

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

GPIOPinConfigure(GPIO\_PA0\_U0RX);

GPIOPinConfigure(GPIO\_PA1\_U0TX);

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

UARTStdioConfig(0, 115200, g\_ui32SysClock);

}

int

main(void)

{

int nStatus;

FRESULT fresult;

/\*应该启用FPU，因为有些编译器将使用浮点寄存器，即使对于非浮点代码也是如此。

\* 如果未启用FPU，则会导致故障。这也确保了浮点操作可以添加到此应用程序中，

\* 并且可以正常工作并使用硬件浮点单元。最后，为中断处理程序启用了延迟堆栈。

\* 这允许在中断处理程序中使用浮点指令，但要牺牲额外的堆栈使用。\*/

FPUEnable();

FPULazyStackingEnable();

// 从锁相环以120兆赫的频率运行。

g\_ui32SysClock = SysCtlClockFreqSet((SYSCTL\_XTAL\_25MHZ |

SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_PLL |

SYSCTL\_CFG\_VCO\_480), 120000000);

//启用此示例使用的外围设备。

//为100赫兹中断配置SysTick。

ROM\_SysTickPeriodSet(g\_ui32SysClock / 100);

ROM\_SysTickEnable();

ROM\_SysTickIntEnable();

//启用中断

ROM\_IntMasterEnable();

ConfigureUART();//UART初始化

UARTprintf("\n\nSD Card Example Program\n");

UARTprintf("Type \'help\' for help.\n");

//使用逻辑磁盘0装载文件系统。

fresult = f\_mount(0, &g\_sFatFs);

if(fresult != FR\_OK)

{

UARTprintf("f\_mount error: %s\n", StringFromFresult(fresult));

return(1);

}

while(1)

{

// 向控制台打印提示。显示CWD。

UARTprintf("\n%s> ", g\_cCwdBuf);

// 从用户处获取一行文本。

UARTgets(g\_cCmdBuf, sizeof(g\_cCmdBuf));

//将行从用户传递到命令处理器。它将被解析并执行有效的命令。

nStatus = CmdLineProcess(g\_cCmdBuf);

//处理命令错误的情况。

if(nStatus == CMDLINE\_BAD\_CMD)

{

UARTprintf("Bad command!\n");

}

//处理参数过多的情况。

else if(nStatus == CMDLINE\_TOO\_MANY\_ARGS)

{

UARTprintf("Too many arguments for command processor!\n");

}

//否则命令被执行。如果返回错误代码，则打印错误代码。

else if(nStatus != 0)

{

UARTprintf("Command returned error code %s\n",

StringFromFresult((FRESULT)nStatus));

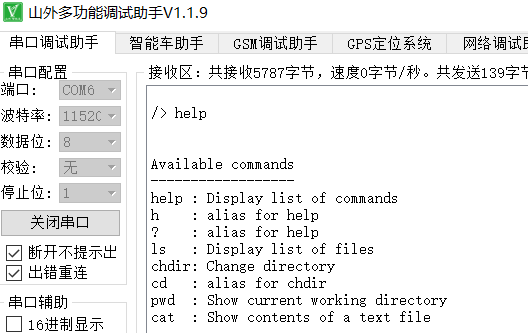
}

}

}

**实验现象：**

程序运行后，在串口调试助手显示相应的字符串，输入对应的字符，会显示不同的内容，显示结果如下：



**五、思考题**

**1、尝试编写尝试将发送到 UART 上的信息，显示在液晶屏上。**

**实验代码：**

int Cmd\_ls(int argc, char \*argv[])

{

unsigned long ulTotalSize;

unsigned long ulFileCount;

unsigned long ulDirCount;

FRESULT fresult;

FATFS \*pFatFs;

//

// Open the current directory for access.

//

fresult = f\_opendir(&g\_sDirObject, g\_cCwdBuf);

//

// Check for error and return if there is a problem.

//

if(fresult != FR\_OK)

{

return(fresult);

}

ulTotalSize = 0;

ulFileCount = 0;

ulDirCount = 0;

//

// Give an extra blank line before the listing.

//

UARTprintf("\n");

zz+=16;

//

// Enter loop to enumerate through all directory entries.

//

for(;;)

{

//

// Read an entry from the directory.

//

fresult = f\_readdir(&g\_sDirObject, &g\_sFileInfo);

//

// Check for error and return if there is a problem.

//

if(fresult != FR\_OK)

{

return(fresult);

}

//

// If the file name is blank, then this is the end of the listing.

//

if(!g\_sFileInfo.fname[0])

{

break;

}

//

// If the attribue is directory, then increment the directory count.

//

if(g\_sFileInfo.fattrib & AM\_DIR)

{

ulDirCount++;

}

//

// Otherwise, it is a file. Increment the file count, and add in the

// file size to the total.

//

else

{

ulFileCount++;

ulTotalSize += g\_sFileInfo.fsize;

}

//

// Print the entry information on a single line with formatting to show

// the attributes, date, time, size, and name.

//

UARTprintf("%c%c%c%c%c %u/%02u/%02u %02u:%02u %9u %s\n",

(g\_sFileInfo.fattrib & AM\_DIR) ? 'D' : '-',

(g\_sFileInfo.fattrib & AM\_RDO) ? 'R' : '-',

(g\_sFileInfo.fattrib & AM\_HID) ? 'H' : '-',

(g\_sFileInfo.fattrib & AM\_SYS) ? 'S' : '-',

(g\_sFileInfo.fattrib & AM\_ARC) ? 'A' : '-',

(g\_sFileInfo.fdate >> 9) + 1980,

(g\_sFileInfo.fdate >> 5) & 15,

g\_sFileInfo.fdate & 31,

(g\_sFileInfo.ftime >> 11),

(g\_sFileInfo.ftime >> 5) & 63,

g\_sFileInfo.fsize,

g\_sFileInfo.fname);

TFTLCD\_ShowChar(0,zz, ((g\_sFileInfo.fattrib & AM\_DIR) ? 'D' : '-'),CYAN,LIGHTBLUE);

TFTLCD\_ShowChar(0+8,zz, ((g\_sFileInfo.fattrib & AM\_RDO) ? 'R' : '-'),CYAN,LIGHTBLUE);

TFTLCD\_ShowChar(0+16,zz, ((g\_sFileInfo.fattrib & AM\_HID) ? 'H' : '-'),CYAN,LIGHTBLUE);

TFTLCD\_ShowChar(0+24,zz, ((g\_sFileInfo.fattrib & AM\_SYS) ? 'S' : '-'),CYAN,LIGHTBLUE);

TFTLCD\_ShowChar(0+32,zz, ((g\_sFileInfo.fattrib & AM\_ARC) ? 'A' : '-'),CYAN,LIGHTBLUE);

TFTLCD\_ShowData(0+40,zz, (g\_sFileInfo.fdate >> 9) + 1980,CYAN,LIGHTBLUE);

TFTLCD\_ShowChar(0+80,zz, '/',CYAN,LIGHTBLUE);

TFTLCD\_ShowData(0+88,zz, (g\_sFileInfo.fdate >> 5) & 15,CYAN,LIGHTBLUE);

TFTLCD\_ShowChar(0+128,zz, '/',CYAN,LIGHTBLUE);

TFTLCD\_ShowData(0+136,zz, g\_sFileInfo.fdate & 31,CYAN,LIGHTBLUE);

TFTLCD\_ShowData(0+160,zz, (g\_sFileInfo.ftime >> 11),CYAN,LIGHTBLUE);

TFTLCD\_ShowChar(0+168,zz, ':',CYAN,LIGHTBLUE);

TFTLCD\_ShowData(0+174,zz, (g\_sFileInfo.ftime >> 5) & 63,CYAN,LIGHTBLUE);

zz+=16;

TFTLCD\_ShowData(0,zz, g\_sFileInfo.fsize,CYAN,LIGHTBLUE);

TFTLCD\_ShowString(0+88,zz, g\_sFileInfo.fname,CYAN,LIGHTBLUE);

zz+=16;

}

//

// Print summary lines showing the file, dir, and size totals.

//

UARTprintf("\n%4u File(s),%10u bytes total\n%4u Dir(s)",

ulFileCount, ulTotalSize, ulDirCount);

TFTLCD\_ShowData(0,zz, ulFileCount,CYAN,LIGHTBLUE);

TFTLCD\_ShowString(40,zz, "File(s),",CYAN,LIGHTBLUE);

TFTLCD\_ShowData(96,zz, ulTotalSize,CYAN,LIGHTBLUE);

TFTLCD\_ShowString(136,zz, " bytes total",CYAN,LIGHTBLUE);

zz+=16;

TFTLCD\_ShowData(0,zz, ulDirCount,CYAN,LIGHTBLUE);

TFTLCD\_ShowString(40,zz, " Dir(s)",CYAN,LIGHTBLUE);

//

// Get the free space.

//

fresult = f\_getfree("/", &ulTotalSize, &pFatFs);

//

// Check for error and return if there is a problem.

//

if(fresult != FR\_OK)

{

return(fresult);

}

//

// Display the amount of free space that was calculated.

//

UARTprintf(", %10uK bytes free\n", ulTotalSize \* pFatFs->csize / 2);

TFTLCD\_ShowData(100,zz, ulTotalSize \* pFatFs->csize / 2/10000,CYAN,LIGHTBLUE);

TFTLCD\_ShowData(140,zz, (ulTotalSize \* pFatFs->csize / 2)%10000,CYAN,LIGHTBLUE);

TFTLCD\_ShowString(180,zz, "K bytes free",CYAN,LIGHTBLUE);

zz+=16;

//

// Made it to here, return with no errors.

//

return(0);

}

int

Cmd\_cd(int argc, char \*argv[])

{

unsigned int uIdx;

FRESULT fresult;

//

// Copy the current working path into a temporary buffer so it can be

// manipulated.

//

strcpy(g\_cTmpBuf, g\_cCwdBuf);

//

// If the first character is /, then this is a fully specified path, and it

// should just be used as-is.

//

if(argv[1][0] == '/')

{

//

// Make sure the new path is not bigger than the cwd buffer.

//

if(strlen(argv[1]) + 1 > sizeof(g\_cCwdBuf))

{

UARTprintf("Resulting path name is too long\n");

TFTLCD\_ShowString(0,zz,"Resulting path name is too long",CYAN,LIGHTBLUE);

zz+=16;

return(0);

}

//

// If the new path name (in argv[1]) is not too long, then copy it

// into the temporary buffer so it can be checked.

//

else

{

strncpy(g\_cTmpBuf, argv[1], sizeof(g\_cTmpBuf));

}

}

//

// If the argument is .. then attempt to remove the lowest level on the

// CWD.

//

else if(!strcmp(argv[1], ".."))

{

//

// Get the index to the last character in the current path.

//

uIdx = strlen(g\_cTmpBuf) - 1;

//

// Back up from the end of the path name until a separator (/) is

// found, or until we bump up to the start of the path.

//

while((g\_cTmpBuf[uIdx] != '/') && (uIdx > 1))

{

//

// Back up one character.

//

uIdx--;

}

//

// Now we are either at the lowest level separator in the current path,

// or at the beginning of the string (root). So set the new end of

// string here, effectively removing that last part of the path.

//

g\_cTmpBuf[uIdx] = 0;

}

//

// Otherwise this is just a normal path name from the current directory,

// and it needs to be appended to the current path.

//

else

{

//

// Test to make sure that when the new additional path is added on to

// the current path, there is room in the buffer for the full new path.

// It needs to include a new separator, and a trailing null character.

//

if(strlen(g\_cTmpBuf) + strlen(argv[1]) + 1 + 1 > sizeof(g\_cCwdBuf))

{

UARTprintf("Resulting path name is too long\n");

TFTLCD\_ShowString(0,zz,"Resulting path name is too long",CYAN,LIGHTBLUE);

zz+=16;

return(0);

}

//

// The new path is okay, so add the separator and then append the new

// directory to the path.

//

else

{

//

// If not already at the root level, then append a /

//

if(strcmp(g\_cTmpBuf, "/"))

{

strcat(g\_cTmpBuf, "/");

}

//

// Append the new directory to the path.

//

strcat(g\_cTmpBuf, argv[1]);

}

}

//

// At this point, a candidate new directory path is in chTmpBuf. Try to

// open it to make sure it is valid.

//

fresult = f\_opendir(&g\_sDirObject, g\_cTmpBuf);

//

// If it can't be opened, then it is a bad path. Inform the user and

// return.

//

if(fresult != FR\_OK)

{

UARTprintf("cd: %s\n", g\_cTmpBuf);

TFTLCD\_ShowString(0,zz,"cd: ",CYAN,LIGHTBLUE);

TFTLCD\_ShowString(32,zz,g\_cTmpBuf,CYAN,LIGHTBLUE);

zz+=16;

return(fresult);

}

//

// Otherwise, it is a valid new path, so copy it into the CWD.

//

else

{

strncpy(g\_cCwdBuf, g\_cTmpBuf, sizeof(g\_cCwdBuf));

}

//

// Return success.

//

return(0);

}

int

Cmd\_pwd(int argc, char \*argv[])

{

//

// Print the CWD to the console.

//

UARTprintf("%s\n", g\_cCwdBuf);

TFTLCD\_ShowString(0,zz,g\_cCwdBuf,CYAN,LIGHTBLUE);

zz+=16;

//

// Return success.

//

return(0);

}

int

Cmd\_cat(int argc, char \*argv[])

{

FRESULT fresult;

unsigned int usBytesRead;

//

// First, check to make sure that the current path (CWD), plus the file

// name, plus a separator and trailing null, will all fit in the temporary

// buffer that will be used to hold the file name. The file name must be

// fully specified, with path, to FatFs.

//

if(strlen(g\_cCwdBuf) + strlen(argv[1]) + 1 + 1 > sizeof(g\_cTmpBuf))

{

UARTprintf("Resulting path name is too long\n");

TFTLCD\_ShowString(0,zz,"Resulting path name is too long",CYAN,LIGHTBLUE);

zz+=16;

return(0);

}

//

// Copy the current path to the temporary buffer so it can be manipulated.

//

strcpy(g\_cTmpBuf, g\_cCwdBuf);

//

// If not already at the root level, then append a separator.

//

if(strcmp("/", g\_cCwdBuf))

{

strcat(g\_cTmpBuf, "/");

}

//

// Now finally, append the file name to result in a fully specified file.

//

strcat(g\_cTmpBuf, argv[1]);

//

// Open the file for reading.

//

fresult = f\_open(&g\_sFileObject, g\_cTmpBuf, FA\_READ);

//

// If there was some problem opening the file, then return an error.

//

if(fresult != FR\_OK)

{

return(fresult);

}

//

// Enter a loop to repeatedly read data from the file and display it, until

// the end of the file is reached.

//

do

{

//

// Read a block of data from the file. Read as much as can fit in the

// temporary buffer, including a space for the trailing null.

//

fresult = f\_read(&g\_sFileObject, g\_cTmpBuf, sizeof(g\_cTmpBuf) - 1,

&usBytesRead);

//

// If there was an error reading, then print a newline and return the

// error to the user.

//

if(fresult != FR\_OK)

{

UARTprintf("\n");

zz+=16;

return(fresult);

}

//

// Null terminate the last block that was read to make it a null

// terminated string that can be used with printf.

//

g\_cTmpBuf[usBytesRead] = 0;

//

// Print the last chunk of the file that was received.

//

UARTprintf("%s", g\_cTmpBuf);

TFTLCD\_ShowString(0,zz,g\_cTmpBuf,CYAN,LIGHTBLUE);

int a=strlen(g\_cTmpBuf)/29;

zz+=16\*(a+1);

}

while(usBytesRead == sizeof(g\_cTmpBuf) - 1);

//

// Return success.

//

return(0);

}

int

Cmd\_help(int argc, char \*argv[])

{

tCmdLineEntry \*pEntry;

//

// Print some header text.

//

UARTprintf("\nAvailable commands\n");

TFTLCD\_ShowString(0,zz,"nAvailable commands",CYAN,LIGHTBLUE);

zz+=16;

UARTprintf("------------------\n");

TFTLCD\_ShowString(0,zz,"------------------",CYAN,LIGHTBLUE);

zz+=16;

//

// Point at the beginning of the command table.

//

pEntry = &g\_psCmdTable[0];

//

// Enter a loop to read each entry from the command table. The end of the

// table has been reached when the command name is NULL.

//

while(pEntry->pcCmd)

{

//

// Print the command name and the brief description.

//

UARTprintf("%s%s\n", pEntry->pcCmd, pEntry->pcHelp);

TFTLCD\_ShowString(0,zz,pEntry->pcCmd,CYAN,LIGHTBLUE);

TFTLCD\_ShowString(0+8\*strlen(pEntry->pcCmd),zz,pEntry->pcHelp,CYAN,LIGHTBLUE);

zz+=16;

//

// Advance to the next entry in the table.

//

pEntry++;

}

zz+=16;

//

// Return success.

//

return(0);

}

tCmdLineEntry g\_psCmdTable[] =

{

{ "help", Cmd\_help, " : Display list of commands" },

{ "h", Cmd\_help, " : alias for help" },

{ "?", Cmd\_help, " : alias for help" },

{ "ls", Cmd\_ls, " : Display list of files" },

{ "chdir", Cmd\_cd, ": Change directory" },

{ "cd", Cmd\_cd, " : alias for chdir" },

{ "pwd", Cmd\_pwd, " : Show current working directory" },

{ "cat", Cmd\_cat, " : Show contents of a text file" },

{ 0, 0, 0 }

};

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

//

// The error routine that is called if the driver library encounters an error.

//

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

#ifdef DEBUG

void

\_\_error\_\_(char \*pcFilename, unsigned long ulLine)

{

}

#endif

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

//

// Configure the UART and its pins. This must be called before UARTprintf().

//

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

void

ConfigureUART(void)

{

//

// Enable the GPIO Peripheral used by the UART.

//

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

//

// Enable UART0

//

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

//

// Configure GPIO Pins for UART mode.

//

GPIOPinConfigure(GPIO\_PA0\_U0RX);

GPIOPinConfigure(GPIO\_PA1\_U0TX);

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

//

// Initialize the UART for console I/O.

//

UARTStdioConfig(0, 115200, g\_ui32SysClock);

}

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

//

// The program main function. It performs initialization, then runs a command

// processing loop to read commands from the console.

//

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

int

main(void)

{

int nStatus;

FRESULT fresult;

FPUEnable();

FPULazyStackingEnable();

//

// Run from the PLL at 120 MHz.

//

g\_ui32SysClock = SysCtlClockFreqSet((SYSCTL\_XTAL\_25MHZ |

SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_PLL |

SYSCTL\_CFG\_VCO\_480), 120000000);

//epi初始化

EPIGPIOinit();

//UARTprintf("TFTLCD test\n");

//UARTprintf("EPI Type: host-bus 16-bit interface\n");

TFT\_400x240\_OTM4001Ainit(g\_ui32SysClock);//液晶屏初始化

// Open BackLight.打开背关

GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_PIN\_0);

ROM\_SysTickPeriodSet(g\_ui32SysClock / 100);

ROM\_SysTickEnable();

ROM\_SysTickIntEnable();

//

// Enable Interrupts

//

ROM\_IntMasterEnable();

ConfigureUART();

UARTprintf("\n\nSD Card Example Program\n");

TFTLCD\_ShowString(0,zz,"SD Card Example Program",CYAN,LIGHTBLUE);

zz+=16;

UARTprintf("Type \'help\' for help.\n");

TFTLCD\_ShowString(0,zz,"'help\' for help.",CYAN,LIGHTBLUE);

zz+=16;

fresult = f\_mount(0, &g\_sFatFs);

if(fresult != FR\_OK)

{

UARTprintf("f\_mount error: %s\n", StringFromFresult(fresult));

TFTLCD\_ShowString(0,zz,"f\_mount error:",CYAN,LIGHTBLUE);

TFTLCD\_ShowString(0+14\*8,zz,StringFromFresult(fresult),CYAN,LIGHTBLUE);

zz+=16;

return(1);

}

//

while(1)

{

//

// Print a prompt to the console. Show the CWD.

//

UARTprintf("\n%s> ", g\_cCwdBuf);

TFTLCD\_ShowString(0,zz,g\_cCwdBuf,CYAN,LIGHTBLUE);

zz+=16;

//

// Get a line of text from the user.

//

UARTgets(g\_cCmdBuf, sizeof(g\_cCmdBuf));

//

nStatus = CmdLineProcess(g\_cCmdBuf);

if(nStatus == CMDLINE\_BAD\_CMD)

{

UARTprintf("Bad command!\n");

TFTLCD\_ShowString(0,zz,"Bad command!",CYAN,LIGHTBLUE);

zz+=16;

}

else if(nStatus == CMDLINE\_TOO\_MANY\_ARGS)

{

UARTprintf("Too many arguments for command processor!\n");

TFTLCD\_ShowString(0,zz,"Too many arguments for command processor!",CYAN,LIGHTBLUE);

zz+=16;

}

else if(nStatus != 0)

{

UARTprintf("Command returned error code %s\n",

StringFromFresult((FRESULT)nStatus));

TFTLCD\_ShowString(0,zz,"Command returned error code",CYAN,LIGHTBLUE);

zz+=16;

TFTLCD\_ShowString(0,zz,StringFromFresult((FRESULT)nStatus),CYAN,LIGHTBLUE);

zz+=16;

}

}

}

**实验现象：**



**六、实验体会与心得**

TF卡在日常生活中使用起来非常方便，只需要一个读卡器，即可在电脑上写入和读取数据。TF卡得益于操作系统的封装，用户才可以方便使用，对于没有操作系统的单片机来说，就连TF卡的初始化都十分费劲，复杂的时序图就可以劝退很多人了，而且不同品牌相同类型的卡用同一个初始化函数都不一定可以成功初始化，在我平时的使用中这类模块的时候总是遇到这问题。

本次实验由于TF卡也不能正常得写入，所以并没有深入研究，对于一个正常的嵌入式开发者来说，很多时候都是靠一些不知名的人将底层的函数封装好，我们再在上面进行开发，这就大大提高了效率。