My Controller

Изучаем микроконтроллеры STM32

STM32 FatFs. Обзор библиотечных функций

Итак, карту памяти мы подключили, инициализация ее прошла успешно. В предыдущей статье были рассмотрены функции, которые нужны для работы с картой памяти. Теперь, прежде чем начать работу с FAT, рассмотрим некоторые функции, которые предоставляет библиотека FatFs:

- FRESULT **f_mount** (BYTE, FATFS*) — для регистрации диска;
- FRESULT **f_open** (FIL*, const char*, BYTE) — открывает/создает файл;
- FRESULT **f_close** (FIL*) — закрытие файла;
- FRESULT **f_read** (FIL*, void*, UINT, UINT*) — чтение данных из файла;
- FRESULT **f_write** (FIL*, const void*, UINT, UINT*) — запись данных в файл;
- FRESULT **f_lseek** (FIL*, DWORD) — навигация по открытому файлу;
- FRESULT **f_truncate** (FIL*) — уменьшение размера файла;
- FRESULT **f_sync** (FIL*) — перезапись кешированных данных на диск;
- FRESULT **f_stat** (const char*, FILINFO*) — для получения данных о файле;
- FRESULT **f_opendir** (DIR*, const char*) — открывает директорию (папку);
- FRESULT **f_readdir** (DIR*, FILINFO*) — чтение информации о папке;
- FRESULT **f_unlink** (const char*) — удалить файл/папку;
- FRESULT **f_mkdir** (const char*) — создать папку;
- FRESULT **f_chmod** (const char*, BYTE, BYTE) — изменить атрибуты файла/папки;
- FRESULT **f_utime** (const char*, const FILINFO*) — изменяет время создания файла/папки;
- FRESULT **f_rename** (const char*, const char*) — переименовывает/перемещает файл/папку;
- FRESULT **f_getfree** (const char*, DWORD*, FATFS**) – для определения свободного кол-ва кластеров;
- FRESULT **f_mkfs** (BYTE, BYTE, WORD) — форматирование диска

Как видим, возможности очень даже приличные. Можно работать как с файлами, так и с папками, имеется возможность переименовывать, перемещать файлы, удалять их и прочее.

Присмотревшись к приведенным функциям можем увидеть то, что их объединяет – возвращаемый результат имеет тип **FRESULT** . Это перечисленный тип данных, который описан в файле ff.h и может принимать следующие значения:

- FR_OK — операция завершена успешно;
- FR_DISK_ERR — ошибка при обращении к диску;
- FR_INT_ERR — ошибка при работе самой библиотеки;
- FR_NOT_READY — невозможно работать с диском из-за его неготовности;
- FR_NO_FILE — указанный файл не найден;
- FR_NO_PATH — указанный путь не найден;
- FR_INVALID_NAME — недопустимое имя;
- FR_DENIED — доступ отклонен (диск переполнен или что-то другое);
- FR_EXIST — попытка создать объект с уже существующим именем;
- FR_INVALID_OBJECT —
- FR_WRITE_PROTECTED – запись невозможна, так как диск защищен от записи;
- FR_INVALID_DRIVE — недопустимый номер диска;
- FR_NOT_ENABLED — обратились к диску, который не зарегистрирован функцией f_mount;
- FR_NO_FILESYSTEM — диск не форматирован в FAT;
- FR_MKFS_ABORTED — ошибка при форматировании;
- FR_TIMEOUT — выполнение ф-ии прервано из-за таймаута

Если кого-то не устраивает приведенное выше описание значений, обращайтесь к оригиналу – там все точно и без ошибок. Я привел его для общего обзора.

Ниже рассмотрим некоторые основные функции.

FRESULT **f_mount** (BYTE *vol*, FATFS **fs*) – связывает диск (vol) со структурой, на которую указывает fs.

Без обращения к этой функции “кина не будет”. Прежде чем начать работать с FAT, вызываем эту функцию. Также вызываем ее если нужно отменить связывание. Делаем это так:

```
1 FATFS fs;                 //структура, с которой будем связывать диск                    /
2 f_mount(0, &fs);          //выполняем связывание диска с структурой
3    .......                //работаем с FAT
4 f_mount(0, NULL);         //отменяем связывание
```

FRESULT **f_open** (FIL **fp*, const char **path*, BYTE *mode*) — открывает файл или создает, если он не существует. Немного об аргументах:

FIL **fp* — указатель на структуру, в которой будут храниться данные об открытом файле. Формат этой структуры можно посмотреть в ff.h . В ней, среди прочего, имеются данные о размере файла, номере первого кластера файла, как файл открыт – для чтения, записи и т.д. После обращения к этой функции

создается файловый объект (структура, на которую указывает *fp), посредством которого осуществляется вся дальнейшая работа с файлом: при чтении/записи данных используется уже не имя файла, а указатель на эту структуру.

const char *path – указатель на имя файла. Имя файла может выглядеть приблизительно так: "0:mydir/proba.txt" 0 – это имя диска.

BYTE *mode* – указывает на способ открытия файла и тип доступа к нему. Может состоять из набора следующих флагов:

- FA_OPEN_EXISTING 0×00 — открытие уже имеющегося файла;
- FA_READ 0×01 - открытие файла для чтения;
- FA_WRITE 0×02 - открытие файла для записи;
- FA_CREATE_NEW 0×04 - создание нового файла (если уже существует, ф-я вернет ошибку — FR_EXIST);
- FA_CREATE_ALWAYS 0×08 – создает новый файл. Если он уже есть – создает по новой;
- FA_OPEN_ALWAYS 0×10 — открывает имеющийся файл, если его нет – прежде создает

После того, как файл был успешно открыт, с ним можно работать с помощью функций f_read, f_write, f_lseek, f_truncate, f_sync. По окончанию работы с файлом его можно закрыть функцией

FRESULT **f_close** (FIL *fp), которой в качестве аргумента передаем указатель на структуру открытого файла. Процесс открытия/закрытия файла выглядит следующим образом:

```
FATFS   fs;           //структура, с которой будем связывать диск                          /
FIL     file;         //здесь будут храниться данные об открытом файле
FRESULT res;          //для анализа возвращаемого функциями результата

f_mount(0, &fs);      //выполняем связывание диска со структурой fs

//пытаемся открыть существующи файл для чтения и записи
res = f_open(&file, "0:mydir/proba.txt", FA_OPEN_EXISTING | FA_READ | FA_WRITE);
if(res){};            //обрабатываем ошибку открытия файла, если таковая возникла
  ..............      //что-то делаем с файлом
f_close(&file);       //закрываем открытый файл

f_mount(0, NULL);     //отменяем связывание диска со структурой fs
```

В приведенном примере mydir – это папка, в которой расположен файл proba.txt

Дальше рассмотрим функции, посредством которых можно работать с открытым файлом.

FRESULT **f_read** (FIL *fp, void *buff, UINT btw, UINT *bw) – читает данные из открытого файла.

- FIL *fp — указатель на структуру, в которой храняться данные об открытом файле;
- void *buff — указатель на буфер, в который будут прочитаны данные.
- UINT btr — количество запрашиваемых байт;
- UINT *br – указатель на переменную, в которую будет записано количество реально прочитанных байт из файла.

FRESULT **f_write** (FIL *fp, const void *buff, UINT btw, UINT *bw) – записывает данные в открытый файл.

- FIL *fp — указатель на структуру, в которой храняться данные об открытом файле;
- const void *buff- указатель на буфер, с которого будут прочитаны данные для записи в файл.
- UINT btw — количество байт для записи;
- UINT *bw – указатель на переменную, в которую будет записано количество реально записанных в файл байт

При работе с функциями чтения/записи возникает вопрос – а с какого места будет производиться эта операция. Для навигации по файлу предназначен специальный указатель, который после открытия файла указывает на первый байт. Поэтому операция чтения/записи будет производиться с первого байта. Для перемещения указателя в нужное место используется следующая функция:

FRESULT **f_lseek** (FIL *fp, DWORD ofs), где

- FIL *fp — указатель на структуру, в которой храняться данные об открытом файле;
- DWORD ofs – значение смещения в файле относительно его начала

FRESULT **f_truncate** (FIL *fp) – обрезает файл в положении указателя, о котором говорилось выше. Например, если указатель переместили на 10 байт относительно начала, а затем вызвали данную функцию, то размер файла станет равным десяти.

FRESULT **f_sync** (FIL *fp) – похоже на f_close, но сохраняет файл открытым. Она просто записывает данные из буфера (кеша) на диск. Дело в то, что размер сектора 512 байт. Если записать данные функцией f_write, то они реально на карту перепишутся, когда будет достигнут конец сектора. Если до этого произойдет сброс контроллера, то данные будут потеряны. Чтобы этого не произошло, их можно периодически сохранять данной функцией.

Для “затравки” напишем код, с помощью которого в корневом каталоге форматированной карты будет создано два текстовых файла, в которые запишем по одной строке:

```
//*****************************************************************************************
//function создает текстовый файл и записывает в него строку "Hello, Word"                //
//argument имя файла                                                                      //
//*****************************************************************************************
FRESULT create_file(const char *FileName)
{
  FRESULT res;              //для возвращаемого функциями результата
  FIL file;                 //файловый объект
  UINT len;                 //для хранения количества реально записанных байт
  char str[]="Hello, Word"; //записываемая строка

  //создаем файл с именем FileName и открываем его для для записи
  res = f_open(&file, FileName, FA_WRITE | FA_CREATE_ALWAYS);
  if(res) return res;    //если произошла ошибка
  //записываем строку в файл
  res = f_write(&file, str, sizeof(str), &len);
  if(res) return res;    //если произошла ошибка
  //закрываем файл
  return f_close(&file);
}

```

```
//*****************************************************************************************
//                                        main                                           //
//*****************************************************************************************
int main (void)
{
  FATFS   fs;                   //структура, с которой будем связывать диск

  ......                        //что-то делаем

  disk_initialize(0);           //инициализируем карту памяти
  f_mount(0, &fs);              //выполняем связывание диска со структурой fs
  create_file ("0:proba.txt");  //создаем первый файл
  create_file ("0:proba2.txt"); //создаем второй файл
  f_mount(0, NULL);             //отменяем связывание диска со структурой fs

  ......                        //что-то делаем
}
```

Когда открываем файл, можно без особого труда узнать его размер. Он указан в файловой переменной (в приведенном выше примере file)

Для этого достаточно обратиться к полю структуры: LenFile = file.fsize;

Если нужна полная информация о файле, необходимо обратиться к функции f_stat, которая позволяет получить состояние файла. Она имеет следующий вид:

FRESULT **f_stat** (const char **path*, FILINFO **fno*), где

- const char **path* – имя файла;
- FILINFO **fno* — указатель на структуру FILINFO, в которую будут записаны данные о файле

Структура FILINFO выглядит следующим образом:

```
typedef struct _FILINFO {
        DWORD fsize;            //размер файла
        WORD fdate;             //последнее изменение даты
        WORD ftime;             //последнее изменение времени
        BYTE fattrib;           //аттрибуты
        char fname[13];         //короткое имя (8.3 format)
        char *lfname;           //указатель на LFN buffer (буфер, где содержится длинное имя) /
        int lfsize;             //размер буфера LFN
} FILINFO;
```

Пришло время поработать с каталогами. Небольшой ликбез в отношении понятия *каталог*.

Каталог (англ. directory) предназначен для организации файлов, для их упорядочивания. Чем отличается каталог от папки? А ничем. Это одно и то же. Просто обычно на экране каталог изображается в виде папки, поэтому чаще говорят не каталог, а папка. Есть такое понятие, как **корневой каталог** - он включает в себя все остальные каталоги. В Windows корневой каталог обозначается буквой (A, B, C и т.д), после которой следует двоеточие. В библиотеке FatFs корневой каталог обозначается цифрой с последующим двоеточием.

Рассмотрим функции, которые обеспечивают работу с каталогами.

FRESULT **f_opendir** (DIR *dj, const char *path) – открывает каталог. Назначение аргументов:

- DIR **dj* – указатель на структуру, где будут храниться данные о каталоге;
- const char **path* – имя открываемого каталога

Для чего открывать каталог? Чтобы можно было его использовать посредством других функций, например f_readdir.

FRESULT **f_readdir** (DIR **dj*, FILINFO **fno*) – для чтения содержимого каталога

- DIR **dj* – указатель на структуру, где хранятся данные об открытом каталоге;
- FILINFO **fno* – указатель на структуру, в которой будет сохраняться информация о найденном элементе каталога.

Что делает эта функция? В каталоге могут содержаться файлы и/или другие каталоги. При каждом вызове этой функции в структуру **fno* будут занесены данные об очередном элементе. В этой структуре есть поле fname. В него записывается имя очередного элемента. Если перебор закончен, это имя будет равно NULL. Как видим, эта функция является прекрасным инструментом для того, чтобы “пошарить” в папке. Если нужен пример использования этой функции, можно посмотреть здесь: http://microsin.net/programming/ARM/fatfs-read-dir.html

FRESULT **f_mkdir** (const char **path*) – создает каталог с указанным именем.

Функция не сложная. Для примера создадим в корневом каталоге папку ‘mydir’ а в ней папку ‘mydir2’ :

```
f_mkdir("0:mydir");                     //создать папку ‘mydir’                             /
f_mkdir("0:mydir/mydir2");              //в папке ‘mydir’ создать папку ‘mydir2’
```

FRESULT **f_unlink** (const char **path*) – удаление файла или папки. Файл удаляет на раз (главное, чтобы он не был открыт и чтобы не был установлен атрибут “только чтение”). А вот перед удалением папки ее необходимо полностью освободить, т.е. удалить все находящиеся в ней объекты.

FRESULT **f_rename** (const char **path_old*, const char **path_new*) – переименовывает и/или перемещает файл/папку. Объект не должен быть открыт и не забываем за атрибуты.

На этом пока остановлюсь. В процессе работы с FatFs буду добавлять статьи с рабочими примерами.

Рубрика: МИКРОКОНТРОЛЛЕРЫ STM32, Работа с FatFs

Комментарии (7) на “STM32 FatFs. Обзор библиотечных функций”