My Controller

Изучаем микроконтроллеры STM32

- Home
- О сайте
- СОФТ

« STM32 Подключаем энкодер
STM32 FatFs. Обзор библиотечных функций »

STM32 Работа с библиотекой FatFs

Столкнулся с необходимостью прикрутить к контроллеру карту памяти и работать с ней посредством файловой системы. Для этого необходима специальная библиотека. Конечно, ее возможно написать самому, но уж очень это трудоемкая работа. Поэтому решил воспользоваться чужими наработками (говорят что одна из причин технического прогресса – разделение труда).

Существуют свободно распространяемые библиотеки для работы с FAT. Одна из них – FatFs. Больше информации о ней на русском языке можно найти здесь http://microsin.net/programming/ARM/fatfs-file-system.html

Возможности данной библиотеки достаточно большие. В ней содержатся функции для работы с файлами, папками, можно читать данные из файла, сохранять данные в файле, создавать файлы, переименовать файл и др.

Библиотеку можно использовать для 8-ми битных контроллеров, а для 32-х разрядных и подавно. Имеется возможность работы с длинными именами файлов, в кодировке Unicode.

Есть смысл познакомиться с ней поближе.

Для начала я скачал по указанной ссылке пример проекта с использованием FatFs, адаптированный под микроконтроллеры STM32. В этом примере работа с картой ведется через SPI. Имеется два варианта – без использования ДМА и с использованием оного.

Я пока остановился на первом варианте с перспективой перехода на второй. Правда мне пришлось немного изменить его, т.к. в оригинале используется SPI1, а у меня карта подключена к модулю SPI2.

Сделаю небольшой обзор необходимых файлов:

- **ff.c** – именно здесь содержатся функции для работы с FAT;
- **ff.h** – содержит объявления функций для работы с FAT, описание различных структур, позволяет задавать режимы работы (например, разрешить/запретить использование длинных имен, максимальную длину этих имен) а также содержит некоторые макроопределения;
- **fattime.c** и **fattime.h** – самые простые файлы. Нужны для возврата текущего времени, нужного при создании файлов. В принципе, можно возвращать не текущее время, а какое-то фиксированное;
- **integer.c** – для определения размеров целочисленных типов;
- **diskio.c** и **diskio.h** – для связи с картой памяти. Именно diskio.c необходимо корректировать для конкретного контроллера, вся остальная часть является платформенно независимой.

Есть еще файлы, которые позволяют работать с кодировкой Unicode, но я его пока не включал в проект, так как для первого ознакомления он не особо нужен, а памяти занимает много. Чтобы компилятор не ругался на отсутствие этого файла, пришлось закоментировать строки 1146 и 1186 в файле ff.c

Файл diskio.c я подкорректировал, так как у меня используется второй SPI

Помимо указанных выше файлов в проект нужно включить следующие файлы из библиотеки StdPeriph_Lib:

- stm32f10x_gpio.h и stm32f10x_gpio.c;
- stm32f10x_rcc.h и stm32f10x_rcc.c;
- stm32f10x_spi.h и stm32f10x_spi.h;

Эти файлы нужны для работы с соответствующей периферии микроконтроллера.

Если кому интересно, указанные файлы можно скачать здесь: **скачать**

Краткий обзор некоторых функций

Для работы с картой памяти файлах diskio.c и diskio.h имеются следующие функции:

- DSTATUS **disk_initialize** (BYTE);
- DSTATUS **disk_status** (BYTE);
- DRESULT **disk_read** (BYTE, BYTE*, DWORD, BYTE);
- DRESULT **disk_write**(BYTE, const BYTE*, DWORD, BYTE);
- DRESULT **disk_ioctl** (BYTE, BYTE, void*);

DSTATUS **disk_initialize** (BYTE drv) – используется для инициализации диска. Она определяет тип карты, присутствует ли она вообще, включена ли защита от записи и т.д. Если ты только подключил карту и хочешь проверить результат своего творчества, то нужно использовать именно эту функцию. В качестве аргумента передаем ноль.

В качестве результата возвращается состояние диска. Результат также сохраняется в глобальной переменной Stat. В ней задействованы три младших разряда, которые являются отдельными флагами. В файле diskio.h для этого имеются следующие макроопределения:

- STA_NOINIT 0×01 — диск не инициализирован;
- STA_NODISK 0×02 — нет карты памяти;
- STA_PROTECT 0×04 — включена защита от записи

Анализируя результат функции disk_initialize можем судить о состоянии диска (читай *карты памяти*). Если результат равен нулю, значит все окей, если установлен один из трех младших разрядов или их комбинация – значит есть какая-то бяка. Анализируя каждый разряд поймем в чем проблема.

DSTATUS **disk_status** (BYTE drv) – позволяет в любое время прочитать значение переменной Stat, указывающей на состояние диска. Аргумент – номер диска(0).

DRESULT **disk_read** (BYTE drv, BYTE* buff, DWORD sector, BYTE count) – позволяет прочитать нужное количество секторов (count = 1..128) начиная с указанного сектора (sector) в указанный буфер (buff). С аргументом drv мы уже знакомы.

При работе с FAT к этой функции обращаться не придется, но она очень интересна, т.к. ее (а также следующую функцию) можно использовать для своих нужд без использования FAT.

Возвращаемое значение имеет тип *DRESULT,* который представляет собой перечисленный тип, который описан в diskio.h:

```
typedef enum
{
  RES_OK = 0,    //операция выполнена нормально
  RES_ERROR,     //какая-то аппаратная ошибка (проблема с картой и т.д.)
  RES_WRPRT,     //включена защита от записи (используется при записи секторов)
  RES_NOTRDY,    //диск не был инициализирован
  RES_PARERR     //передали неверный параметр (например count = 0)
} DRESULT;
```

DRESULT **disk_write**(BYTE drv , const BYTE* buff, DWORD sector, BYTE count) – позволяет записать нужное количество секторов на карту, начиная с указанного сектора. Передаваемые аргументы и возвращаемый результат такие же как и в предыдущей функции.

Если нет необходимости записывать данные на диск, то данную функцию можно отключить макросом _READONLY в файле diskio.h присвоив ему значение, отличное от нуля.

Чуть не забыл самое главное. Чтобы перечисленные выше ф-ии нормально работали, необходимо настроить один из таймеров контроллера на прерывание через каждые 10мс, а в каждом прерывании необходимо вызывать функцию **disk_timerproc**. Это нужно для всяких таймаутов. Короче, нужно.

Описание функций библиотеки FatFs приведу в следующей статье. А пока подключаем карту памяти, выполняем инициализацию таймера, в прерывании которого обращаемся к disk_timerproc. Затем в функции main вызываем disk_initialize. Если она возвращает ноль – радостно хлопаем в ладоши, иначе – чешим затылок и думаем: *“а чё оно не работает”*

Рубрика: МИКРОКОНТРОЛЛЕРЫ STM32, Работа с FatFs