Главная › Программирование AVR › AVR Урок 33. SPI. Карта SD. FAT. Часть 4

AVR Урок 33. SPI. Карта SD. FAT. Часть 4

Posted on Январь 11, 2017 by Narod Stream Опубликовано в Программирование AVR — Нет комментариев ↓

Урок 33
Часть 4

SPI. Карта SD. FAT

Продолжаем работать с флеш-картой **SD**. В **прошлой части** нашего урока мы работали с блоками. Нам удалось считать блок памяти величиной 512 байт с карты, а также удалось такой же блок на карту записать. Только обычно, как мы знаем, с данной картой не рабоатют блоками и байтами, как с EEPROM, а работают с файлами и каталогами при помощи файловых систем.

Поэтому нам тоже ничего не остаётся делать, как заняться этим вопросом и попробовать написать что-то по работе с файловой системой.

Только рутинную работу по написанию библиотеки для именно дисково-секторных процедур мы на себя брать не будем, так как для этого есть универсальная бесплатная готовая библиотека Petit FAT File System Module, которая размещена на сайте http://elm-chan.org/ с лицензией Creative Commons (указание автора).

Библиотека есть двух видов, мы воспользуемся упрощённым именно petit. Более сложный тип умеет больше, но более сложен в использоании.

Также ещё было принято решение, начиная именно с этой части урока осуществить переход на среду програмирования Atmel Studio 7, так как некоторые вопросы уже не решаются на шестой студии. Скачивается она на сайте Atmel, установка простейшая, поэтому данный вопрос мы затрагивать не будем.

Свежие комментарии

- Сергей к записи PIC Урок 3. Бегущие огни
- Narod Stream к записи PIC Урок 5. Таймеры
- Артем к записи PIC Урок 5. Таймеры
- Narod Stream к записи AVR Урок 13. ШИМ. Мигаем светодиодом плавно. Часть 1
- Narod Stream к записи STM Урок 10. HAL. Изучаем PWM (ШИМ). Мигаем светодиодами плавно

Форум. Последние ответы

- Narod Stream в Программирование МК STM32
  2 дн., 3 час. назад
- Zandy в Программирование МК STM32
  2 дн., 10 час. назад
- Narod Stream в Программирование МК STM32
  1 неделя, 6 дн. назад
- Narod Stream в Программирование МК STM32
  1 неделя, 6 дн. назад
- fireweb в Программирование МК STM32
  2 нед., 2 дн. назад

Январь 2018

Пн	Вт	Ср	Чт	Пт	Сб	Вс
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

« Дек

Архивы

- Январь 2018
- Декабрь 2017
- Ноябрь 2017
- Октябрь 2017
- Сентябрь 2017
- Август 2017
- Июль 2017
- Июнь 2017
- Май 2017

Мета

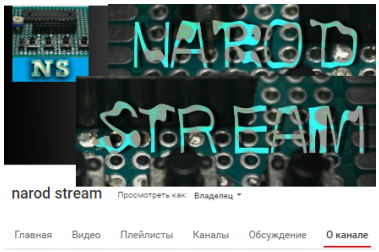
- Регистрация
- Войти
- RSS записей
- RSS комментариев
- WordPress.org

искать здесь …
Фильтровать

Уроки по программированию МК

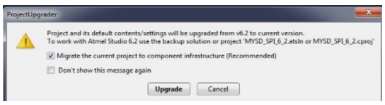
Программирование МК AVR
Программирование МК STM32
Программирование МК PIC
Тесты устройств и аксессуаров

Заходите на канал Narod Stream



- Март 2017
- Февраль 2017
- Январь 2017
- Декабрь 2016
- Ноябрь 2016

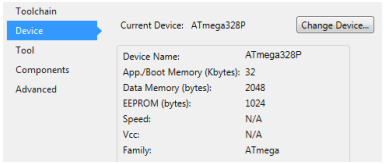
Проект у нас из прошлой части. Чтобы его преобразовать в проект для седьмой студии, его достаточно с помощью неё открыть и согласиться с преобразованием



В следующем окне выбираем в выпадающем списке наш проект и соглашаемся.

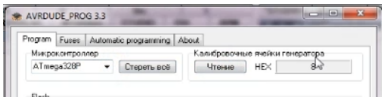
Второе принятое решение — это вместо Atmega8 в отладочную плату поместить контроллер Atmega328. Основное отличие — большая оперативная память, необходимая для полноценной работы с файлами и большая память под прошивку.

Поэтому в свойствах проекта также меняем контроллер

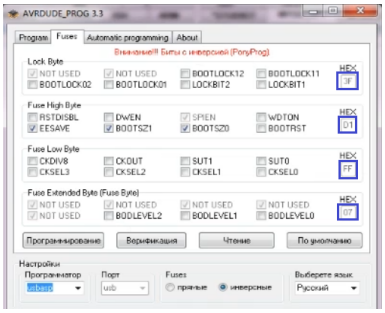


Больше ничего не меняем, в том числе и в коде. Возможно, если бы был у нас аппаратный SPI, то изменения бы какие-то были, врать не буду. Если был бы USART, то были бы точно.

В программе для прошивания мы также меняем контроллер и считываем калибровочные ячейки



Затем переходим во фьюзы и выстанавливаем их вот так при условии, что на нашей плате мы ничего не трогали кроме замены контроллера, что тот же резонатор у нас отсался



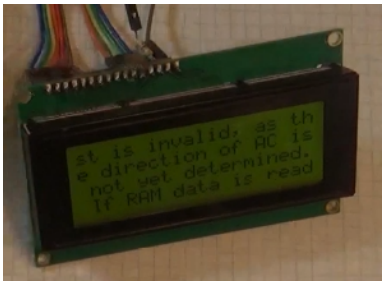
Выставляете их точно также, и жмёте кнопку "Программирование".

Возвращаемся на первую закладку, выбираем проект и прошиваем контроллер.

Посмотрим, всё ли у нас по прежнему работает

Рубрики

- 1-WIRE (3)
- ADC (6)
- DAC (4)
- GPIO (25)
- I2C (19)
- SPI (13)
- USART (8)
- Программирование AVR (131)
- Программирование PIC (6)
- Программирование STM32 (211)
- Тесты устройств и аксессуаров (1)



Теперь вернёмся в проект.

Скачаем сначала файлы с сайта. Заходим на вышеуказанный сайт, жмём там ссылочку Softwares, и в таблице выбираем **Petit FatFs Module**

Откроете ссылку, может там же и скачать библиотеку, и почитать назначение функций.

Ну, в принципе, Вы можете не скачивать, у меня в архиве с проектом будет уже всё настроенное. Дело в том что данная библиотека предназначена не только под флеш-карты, а ещё и под диски и прочие носители информации, и поэтому инициализацию и функции доступа к периферии и передачи данных нужно подкладывть туда свои. Например, функция **disk_initialize** там пустая, нужно писать свою. Но дело выше, можете скачать и настроить самостоятельно.

Также на странице, которая открылась по ссылке, существует ссылка для скачивания самой библиотеки, а также ссылка на пример

Мы будем скачивать именно пример, там также есть вся библиотека, только более адаптированная к карте SD.

В архиве будет папка avr, из которой мы возьмём следующие файлы: **pff.c, mmc.c, diskio.h, pff.h, integer.h**. Скопируем мы их в наш проект и подключим затем к нему. Данные файлы у меня уже исправлены — и мною и не только мною, поэтому советую просто всё взять именно из моего проекта. Можно взять также с сайти и сравнить, что изменено.

Вообщем, подключили мы всё в проект из архива моего же другого сохранённого проекта, где всё настроено.

Также подключим данную библиотеку в файле main.h, который после всех подключений будет иметь вот такой вид

```
#ifndef MAIN_H_
#define MAIN_H_
#define F_CPU 8000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "pff.h"
#include "diskio.h"
#include "integer.h"
#include "lcd.h"
#endif /* MAIN_H_ */
```

Из функции main() удалим вот эти переменные

```
unsigned int i;
unsigned char result;
```

А добавим другие

```
int main(void)
{
  FATFS fs; //FatFs объект
  FRESULT res; //Результат
выполнения
  WORD s1;
```

Всё, что находится после вывода четырёх тестовых строк на диспелей и задержки, закомментируем до бесконечного цикла.

Вызовем функцию из подключенной библиотеки в main(), вставив до вызова на всякий случай пустой цикл, отобразив результат

```
clearlcd();//очистим дисплей
asm("nop");
res=pf_mount(&fs); //Монтируем FAT
sprintf(str,"%d",res);
setpos(0,0);
str_lcd(str);
```

Зайдём в файл библиотеки **mmc.c**. Я внёс в нём определённые изменения, например добавил макросы для ножек SPI нашего контроллера

```
// Definitions for MMC/SDC
connection
#define SD_DI 3
#define SD_DO 4
#define SD_CLK 5
#define SD_CS 2
```

Далее определены команды SD, которые нам потребуются

```
// Definitions for MMC/SDC command
#define CMD0 (0x40+0) //
GO_IDLE_STATE
#define CMD1 (0x40+1) //
SEND_OP_COND (MMC)
#define ACMD41 (0xC0+41) //
SEND_OP_COND (SDC)
#define CMD8 (0x40+8) //
SEND_IF_COND
```

```
#define CMD16 (0x40+16) //
SET_BLOCKLEN
#define CMD17 (0x40+17) //
READ_SINGLE_BLOCK
#define CMD24 (0x40+24) //
WRITE_BLOCK
#define CMD55 (0x40+55) // APP_CMD
#define CMD58 (0x40+58) // READ_OCR
```

Разберёмся немного с функцией **disk_initialize**

Здесь мы сделали на всякий случай 100 циклов передачи, так же как и в нашей функции с прошлых занятий

```
#endif
  for (n = 100; n; n–) rcv_spi(); //
Dummy clocks
```

Затем обнулили переменную

```
for (n = 100; n; n–) rcv_spi(); //
Dummy clocks
ty = 0;
```

Дальше передаём команду CMD0 и при успешной передачи попадаем в тело условия

```
ty = 0;
if (send_cmd(CMD0, 0) == 1) { //
Enter Idle state
  }
```

Дальше передаём команду CMD8 и при удачном результате попадаем в тело следующего условия

```
if (send_cmd(CMD0, 0) == 1) { //
Enter Idle state
  if (send_cmd(CMD8, 0x1AA) == 1) {
// SDv2
  } else { // SDv1 or MMCv3
  }
  }
```

В качестве аргумента у нас 0x1AA.

Данная команда возвращает результат типа R7

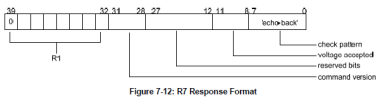


Figure 7-12: R7 Response Format

При успешном выполнении команды мы попадём в тело условия. А успешное выполнение нам даёт право судить о том. что у нас карта формата SD v2, а если ошибка — то SD v1 либо MMC.

Подключим библиотеку для lcd в файл mmc.c

```
#include "diskio.h"
#include "lcd.h"
```

Продолжим нашу функцию.

В теле, в которое мы попали, примем результат типа R7 и отобразим его на дисплее, применив хитрые форматы для отображения шестнадцатеричных величин

```
if (send_cmd(CMD8, 0x1AA) == 1) { // SDv2
setpos(0,3);
  str_lcd("SDv2");
  for (n = 0; n < 4; n++) ocr[n] = rcv_spi(); // Get trailing return value of R7 resp
  setpos(8,0);
  sprintf(str,"%02X",ocr[3]);
  str_lcd(str);
  setpos(11,0);
  sprintf(str,"%02X",ocr[2]);
  str_lcd(str);
  setpos(14,0);
  sprintf(str,"%02X",ocr[1]);
  str_lcd(str);
  setpos(17,0);
  sprintf(str,"%02X",ocr[0]);
  str_lcd(str);
```

Дальше в этом же теле будет ещё одно условие

```
str_lcd(str);
if (ocr[2] == 0x01 && ocr[3] == 0xAA) { // The card can work at vdd range of 2.7-3.6V
}
```

То есть здесь мы уже начинаем исследовать возвращённый результат. Начнём писать тело данного условия, отобразив в случае попадания сюда на дисплее AA01

```
if (ocr[2] == 0x01 && ocr[3] == 0xAA) { // The card can work at vdd range of 2.7-3.6V
  setpos(5,3);
  str_lcd("AA01");
```

Продолжим писать тело условия. Передадим там следующую команду и также отобразим результат. Смысл команды виден в комментариях

```
  str_lcd("AA01");
  for (tmr = 12000; tmr && send_cmd(ACMD41, 1UL << 30); tmr--) ; // Wait for leaving idle state (ACMD41 with HCS bit)
  if (tmr && send_cmd(CMD58, 0) == 0) { // Check CCS bit in the OCR
  setpos(10,3);
  str_lcd("CCS");
  for (n = 0; n < 4; n++) ocr[n] = rcv_spi();
  setpos(8,1);
  sprintf(str,"%02X",ocr[3]);
  str_lcd(str);
  setpos(11,1);
  sprintf(str,"%02X",ocr[2]);
  str_lcd(str);
  setpos(14,1);
  sprintf(str,"%02X",ocr[1]);
  str_lcd(str);
  setpos(17,1);
  sprintf(str,"%02X",ocr[0]);
```

```
  str_lcd(str);
  ty = (ocr[0] & 0x40) ? CT_SD2 |
CT_BLOCK : CT_SD2; // SDv2 (HC or
SC)
}
```

Теперь в теле условия проверки карты на версию, только в теле невыполнения условия, мы также узнаем, какая именно у нас карта — SD или MMC.

В принципе, результат отображать незачем, мы не будем работать с такими картами скорее всего, а если и будем, то увидим, что у нас не будет отображения предыдущих результатов, которые мы отображали в случае выполнения условия.

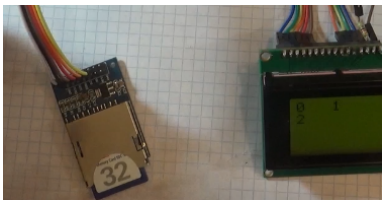
```
  } else { // SDv1 or MMCv3
      if (send_cmd(ACMD41, 0) <= 1)
{
      ty = CT_SD1; cmd = ACMD41; //
SDv1
    } else {
      ty = CT_MMC; cmd = CMD1; //
MMCv3
    }
    for (tmr = 25000; tmr &&
send_cmd(cmd, 0); tmr--) ; // Wait
for leaving idle state
    if (!tmr || send_cmd(CMD16, 512)
!= 0) // Set R/W block length to 512
    ty = 0;
  }
}
```

Выйдем из тела и допишем функцию, отобразив на дисплее тип карты в виде целочисленной величины.

По окончанию прогоним пустой цикл по SPI и вернём результат типа карты

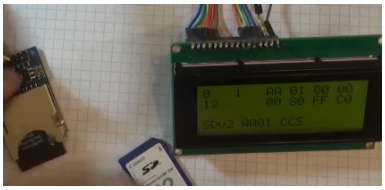
```
  }
  CardType = ty;
  sprintf(str,"%d",CardType);
  setpos(0,1);
  str_lcd(str);
  release_spi();
  return ty ? 0 : STA_NOINIT;
}
```

Соберём код, прошьём контроллер и посмотрим, что у нас определится



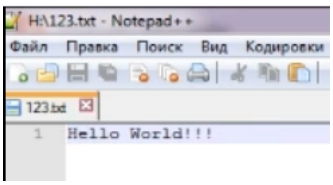
Определился ноль, означающий, что карта у нас самого простого типа, так как она у нас всего на 32 мегабайта.

Вставим карту на 4 гигабайта, перезагрузим контроллер и посмотрим результат



Здесь мы уже получили версию вторую. То же самое мы получим, если воспользуемся флешкартой на 32 гигабайта.

Поместим карту на 32 мегабайта в картовод и мы посмотрим, что там создан текстовый файл "123.txt" с определённым содержимым



Воткнём карту назад в наш картоприёмник и в main() вызовем следующую функцию из подключенной библиотеки и отобразим результат выполнения

```
str_lcd(str);
_delay_ms(2000);
clearlcd();//очистим дисплей
res=pf_open("/123.txt");
sprintf(str,"%d",res);
setpos(0,0);
str_lcd(str);
```

Я думаю, по имени функции можно определить, чем она занимается.

Мы здесь пытаемся открыть файл и отобразить результат на дисплее. Но только это будет результат не в виде содержимого файла, а в виде типа результата. Слеш должен быть обязательно. Он означает, что мы читаем файл из текущего каталога.

Соберём код, прошьём контроллер и посмотрим результат.

У нас на дисплее ноль, что означает, что мы файл открыли.

Теперь проверим это, вставив в картоприёмник остальные карты памяти, так как на них также записан данный файл. Если в этих случаях мы также видим нули, то мы всё сделали правильно.

А уже содержимое файла мы увидим в **следующей части** нашего урока.

Предыдущая часть
Программирование МК AVR
Следующая часть

Техническая документация на Secure Digital

Программатор, модуль SD и дисплей можно приобрести здесь:

Программатор (продавец надёжный)

USBASP USBISP 2.0

Модуль карты SD SPI

Дисплей LCD 20×4

Смотреть ВИДЕОУРОК (нажмите на картинку)

Post Views: 316

‹ AVR Урок 33. SPI. Карта SD. FAT. Часть 3

AVR Урок 33. SPI. Карта SD. FAT. Часть 5 ›

Добавить комментарий

Ваш e-mail не будет опубликован.
Обязательные поля помечены *

Комментарий

Имя *

E-mail *

Сайт

− 2 =

Отправить комментарий