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. назад
- Zandy в Программирование МК STM32
  2 дн., 7 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

AVR Урок 33. SPI. Карта SD. FAT. Часть 2

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

Урок 33
Часть 2

SPI. Карта SD. FAT

В **прошлой части** нашего занятия мы познакомились с флеш-картой **SD**, увидели, что она может работать как по интерфейсу **SPI**, так и по **SDIO**, но решили остановиться на **SPI** и уже начали писать исходный код. Остановились мы на том что написали реализацию передачи байта по интерфейсу **SPI**. Также узнали, что **SPI** у нас будет реализован программно.

Теперь создадим функцию для приёма байта из шины

```
//-------------------
unsigned char SPI_ReceiveByte(void)
{
}
//-------------------
```

Здесь нам уже потребуются две переменные

```
unsigned char SPI_ReceiveByte(void)
{
  unsigned char i, result=0;
```

Также будет цикл

```
unsigned char i, result=0;
for(i=0;i<8;i++)
{
}
```

Мета

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

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

Теперь нам нужно будет наборот не дрыгать ножками, а отслеживать их состояние. Первым делом создадим положительный фронт, выставив высокий уровень на ножке **SCK**

```
for(i=0;i<8;i++)
{
  PORTB|=(1<<SCK);//фронт на лапке
SCK
```

Сдвигаем result влево на 1 пункт, чтобы подготовить место для нового бита. Если это будет самый первый цикл, то ничего страшного, у нас данная переменная все равно в нуле, поэтому не будем сочинять каких-то условий

```
PORTB|=(1<<SCK);//фронт на лапке SCK
result<<=1;//сдвигаем влево байт,
чтобы записать очередной бит
```

Теперь, в случае если у нас уровень на ножке MISO равен 1, то запшем её в правый бит переменной ewsult

```
result<<=1;//сдвигаем влево байт,
чтобы записать очередной бит
if((PINB&(1<<MISO))!=0x00)//запишем
новый бит в младший разряд
  result=result|0x01;//запишем
считанный с лапки порта MISO бит
```

Затем отрицательный фронт на SCK и ждём 1 такт

```
result=result|0x01;//запишем
считанный с лапки порта MISO бит
PORTB&=~(1<<SCK);//спад на лапке SCK
asm("nop");//1 такт подождём
```

Завершим цикл и вернём результат из функции

```
    asm("nop");//1 такт подождём
  }
  return result;//вернем результат
}
```

Теперь нам нужно написать функцию передачи команд в SD.

Откроем пример в даташите

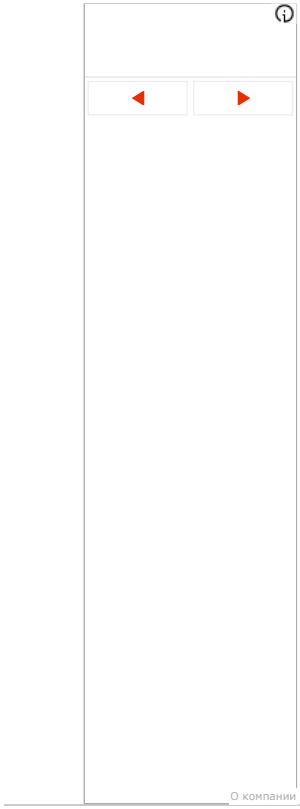
Table 4-15 shows the format of CMD8.

Bit position	47	46	[45:40]	[39:20]	[19:16]	[15:8]	[7:1]	0
Width (bits)	1	1	6	20	4	8	7	1
Value	'0'	'1'	'001000'	'00000h'	x	x	x	'1'
Description	start bit	transmission bit	command index	reserved bits	voltage supplied (VHS)	check pattern	CRC7	end bit

У каждой команды есть индекс. У данной команды индекс 8, так как она именуется CMD8. Также существуют различия в типах команд. Но об этом потом, нас пока интересует именно такой тип, причём именно эту команду нам потом также придется передавать.

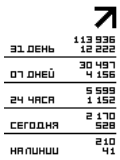
Мы видим что передача команды состоит из 48 бит, то есть из 6 байтов. Первая строка показывает позицию бита в команде, вторая — величину параметра в битах, третья — значение, а четвёртая — разъяснение параметра.

Стартовый бит — всегда 0.



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)



Бит передачи — 1.

индекс команды — в случае данной команды равен 8.

затем идут зарезервированные биты — целых 20 штук, все равные нулю.

Затем идут параметры, 7 бит контрольной суммы и стоповый бит — всегда 1.

Поэтому нам нужно будет написать функцию передачи команды

Напишем её после наших функций передачи и приёма байтов, так как новая функция этими функциями будет пользоваться

```
//————————————-
unsigned char SD_cmd (char dt0,char
dt1,char dt2,char dt3,char dt4,char
dt5)
{
}
//————————————-
```

Вот сколько параметров. Прямо как у нас байтов в команде. Потом мы о них узнаем побольше

Добавим переменные

```
unsigned char SD_cmd (char dt0,char
dt1,char dt2,char dt3,char dt4,char
dt5)
{
  unsigned char result;
  long int cnt;
```

Отправим все параметры в шину

```
long int cnt;
SPI_SendByte(dt0); //индекс
SPI_SendByte(dt1); //Аргумент
SPI_SendByte(dt2);
SPI_SendByte(dt3);
SPI_SendByte(dt4);
SPI_SendByte(dt5); //контрольная
сумма
```

Инициализируем счётчик

```
SPI_SendByte(dt5); //контрольная
сумма
cnt=0;
```

Причём команды, которые мы передаём в карту, могут не только с помощью них чем-то управлять, кое-что могут и возвращать.

Причём возвращаемый результат, мало того, бывает разных типов (я выше говорил про типы команд, это ещё не то) — начинается от типа R1 и т.д.

Вот, например, тип R1

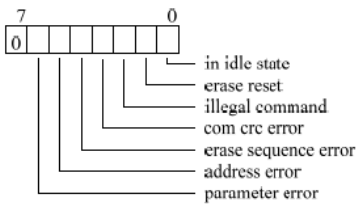


Figure 7-9: R1 Response Format

Вот R2

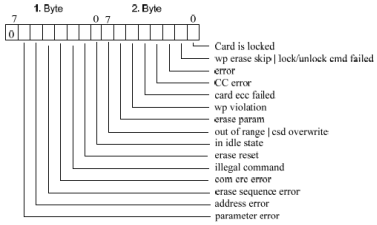


Figure 7-10: R2 Response Format

Пока нас интересует первый тип.

Добавим условный цикл в нашу функцию

```
cnt=0;
do
{ //Ждём ответ в формате R1 (даташит стр 109)
  result=SPI_ReceiveByte();
  cnt++;
} while
(((result&0x80)!=0x00)&&cnt<0xFFFF);
```

Здесь мы ждём пока не прийдёт результат, постоянно принимая байт из шины. Как только байт придёт определённого формата, мы выйдем из функции и его передадим.

То есть должна возникнуть ситуация. когда условие в скобках перестанет выполняться, то есть если самый старший бит перестанет быть у нас не равным нулю, ну или счётчик досчитает до 16535 Это такой своего рода таймаут.

После этого мы вернём результат из нашей функции

```
  } while
(((result&0x80)!=0x00)&&cnt<0xFFFF);
  return result;
}
```

То есть наша функция будет работать только с результатом 1 типа. Другие нам пока не нужны.

Ну и теперь мы наконец-то дошли до интересной функции. Эта функция — инициализация нашей карты. Создадим её ниже нашей только что написанной функции

```
//-----------------
unsigned char SD_Init(void)
{
}
//-----------------
```

Добавим переменные

```
unsigned char SD_Init(void)
{
  unsigned char i,temp;
  long int cnt;
```

Посмотрим рисунок из технической документации

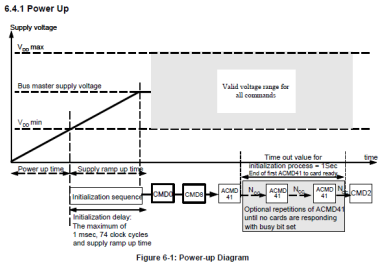


Figure 6-1: Power-up Diagram

Сначала мы какое-то время ждём, затем подаём 74 импульса на шину SCK. Затем отправляем определённые команды.

Пока давайте подадим эти импульсы, их подать несложно

```
long int cnt;
for(i=0;i<10;i++) //80 импульсов (не менее 74) Даташит стр 91
  SPI_SendByte(0xFF);
```

То есть мы отправим 10 байтов FF, тем самым получится 80 единичек. Причём шину SS мы перед передачей не опускаем. Правда импульсов на MOSI я здесь не заметил, будут 80 импульсов на ножке SCK при поднятой MOSI. Но главное работает. Остальное неважно. Вообще всё это дрыганье нужно для того, чтобы таким вот образом карта поняла, что работать мы с ней собираемся именно по SPI и переключилась в соответствующий режим.

А вот после этого только опустим SS

```
for(i=0;i<10;i++) //80 импульсов (не менее 74) Даташит стр 91
  SPI_SendByte(0xFF);
PORTB&=~(1<<SS);//опускаем SS
```

Передадим команду CMD0

```
PORTB&=~(1<<SS);//опускаем SS

temp=SD_cmd(0x40,0x00,0x00,0x00,0x00,0x00,0x95); //CMD0 Даташит стр 102 и 96
```

Как выглядит данная команда, посмотрим на страницах, указанных в комментарии

CMD INDEX	SPI Mode	Argument	Resp	Abbreviation	Command Description
CMD0	Yes	[31:0] stuff bits	R1	GO_IDLE_STATE	Resets the SD Memory Card

Здесь мы видим, что данная коанда для перезагрузки карты, и видим что она должна вернуть нам GO_IDLE_STATE, то есть только нулевой бит должен быть в результате установлен.

Since CMD0 has no arguments, the content of all the fields, including the CRC field, are constants and need not be calculated in run time. A valid reset command is:
0x40, 0x0, 0x0, 0x0, 0x0, 0x95

А это входные аргументы.

Проверим возвращённый результат

```
temp=SD_cmd(0x40,0x00,0x00,0x00,0x00,0x00,0x95); //CMD0 Даташит стр 102 и 96
```

```
if(temp!=0x01) return 1; //Выйти
если ответ не 0x01
```

Пошлём в шину байт FF, чтобы выдавить из сдвигового регистра карты весь мусор, и инициализируем счётчик

```
if(temp!=0x01) return 1; //Выйти
если ответ не 0x01
SPI_SendByte(0xFF);
cnt=0;
```

Передадим теперь ещё команду CMD1, только используя такой же условный цикл, как мы использовали в предыдущей функции, и вернём результат, предварительно отеяв из него ненужное

```
  cnt=0;
  do
  {

temp=SD_cmd(0x41,0x00,0x00,0x00,0x00
,0x95); //CMD1 передаем также,
меняется только индекс
    SPI_SendByte(0xFF);
    cnt++;
  } while
((temp!=0x00)&&cnt<0xFFFF); //Ждёс
ответа R1
  if(cnt>=0xFFFF) return 2;
  return 0;
}
```

Почему же CMD1, а не CMD8? Она же в принципе та же самая ACMD41

Посмотрим вот это дерево

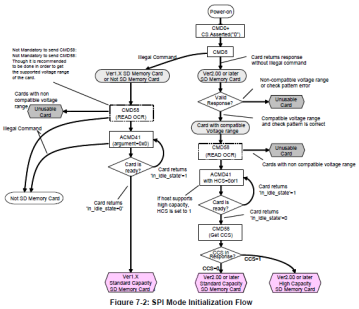


Figure 7-2: SPI Mode Initialization Flow

Хоть тут дерево и большое, но команды CMD8 и CMD58 нам требуются, если мы хотим узнать всё о нашей карте, особенно её тип. Мы пока будем считать, что мы уже знаем тип и нам достаточно будет только ACMD41. Вот так. Вообщем, потом посмотрим, если не будет работать, то мы всё же поработаем с данными командами. Вообще, в будущем они нам ещё будут нужны, когда мы будем уже читать файловую систему с карты.

Добавим строковую переменные в main() для того, чтобы что-то отобразить на дисплее и для результата

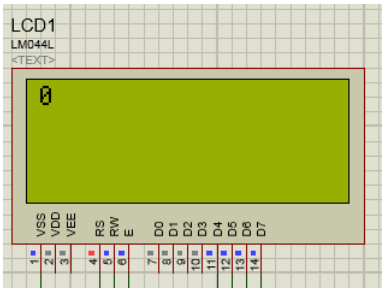
```
unsigned char i;
char str[10];
unsigned char result;
```

Вызовем нашу функцию инициализации, убавив перед ней

немного задержку, и отобразим результат на дисплее

```
_delay_ms(1000);
result=SD_Init();
sprintf(str,"%d",result);
clearlcd();//очистим дисплей
setpos(0,0);
str_lcd(str);
```

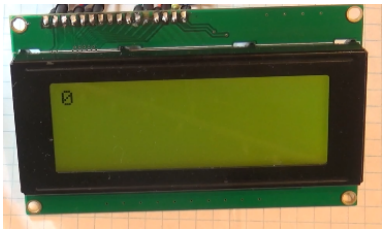
Соберём код и проверим пока результат в протеусе



У нас хороший результат, это очень хорошо, значит виртуальная карта видится. В видеоверсии были ошибки, так что обязательно посмотрите, как мы боремся с ошибками.

Файл с образом карты я также положу в архив с проектом.

Теперь прошьём контроллер и посмотрим результат на живом дисплее с живой картой SD



Отлично! Значит карта у нас инициализировалась и увиделась. А уже читать и писать на ней блоки мы будем в **следующей части** нашего интересного 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)

 AVR SPI. Карта SD. FAT

Post Views: 324

‹ AVR Урок 33. SPI. Карта SD. FAT. Часть 1

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

Добавить комментарий

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

Комментарий

Имя *

E-mail *

Сайт

+ восемь = тринадцать

Отправить комментарий