**SPI. Карта SD. FAT**

Порробуем подключить к микроконтроллеру по данной шине карту памяти **SD (Secure Digital)**.

Данная карта может подключаться также по интерфейсу SDIO, но так как такой интерфейс не поддерживается аппаратно нашим контроллером, то в рамках данного занятия мы его касаться не будем. Нам интересен именно тип подключения по шине **SPI**, так как у нас уже есть неплохие накопленные знания по данной теме, а также аппаратная поддержка в контроллере, который мы программируем.

Тем не менее мы посмотрим распиновку ножек карты по обоим типам



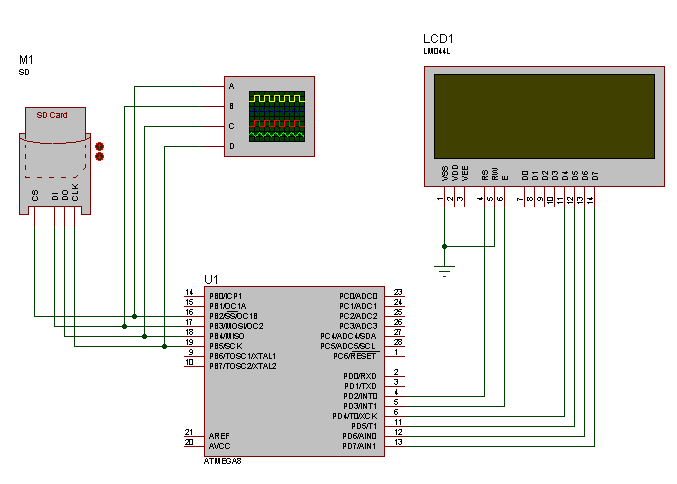
Ну, так как нас интересует второй тип, с ним и будем разбираться.

А разбираться тут особо не в чем. Все эти аббревиатуры нам известны. Здесь все стандартные ножки интерфейса SPI и ничего тут лишнего нет.

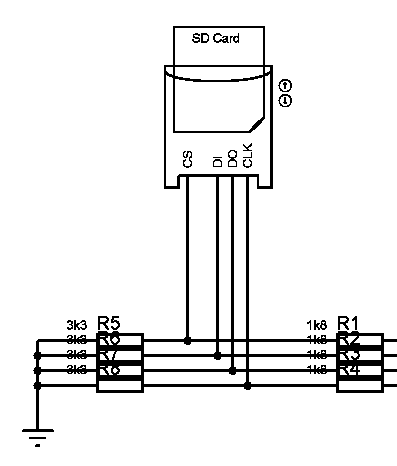
Теперь вообще про карту. Данная карта нам позволяет хранить данные, тип памяти у неё FLASH, который по сравнению с памятью типа EEPROM также является энергонезависимым, то есть при отключении питания данные никуда не пропадают, а остаются храниться. Также данная память имеет отличия, мы с ними познакомимся в процессе программирования. Одно из главных отличий то, что мы уже как в память EEPROM в данную память не можем записать один байт. Теоретически то конечно можем, но только запишутся туда либо только единички из нашего байта либо только нули в зависимости от типа FLASH – NOR или NAND. То есть прежде чем писать байт, нужно его стереть, а в силу организации данной памяти, стирать мы можем только блоками, вот и писать следовательно также только блоками. Но зато есть величайшее отличие от EEPROM – это цена. Она в разы дешевле, даже порой на порядки за одну хранящуюся единицу инфорамции (за мегабайт, за гигабайт). Поэтому у памяти FLASH как правило всегда гораздо больший объём информации.

Существуют 4 типа SD, но это мы изучим немного позднее.

Подключим данную карту пока в протеусе



Здесь всё просто. На самом деле не совсем так. Нужны ещё резисторы

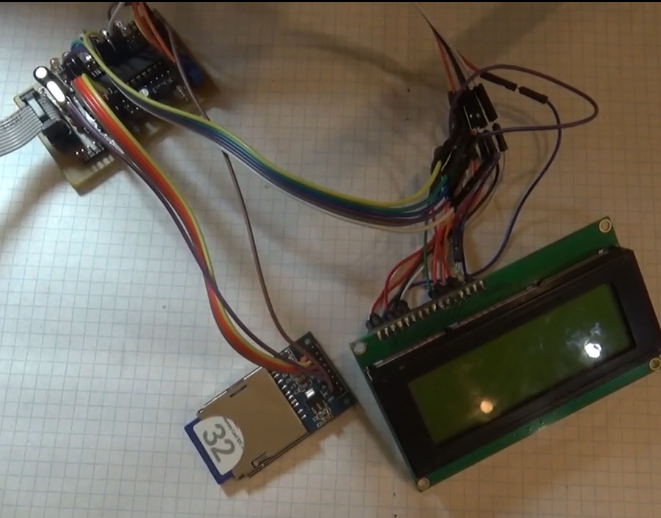


Данные резисторы нужны для того, чтобы обеспечить соответствующие уровни, так как карта питается от 3,3 вольт. Вообще по технической документации от 2,7 до 3,6 вольт.

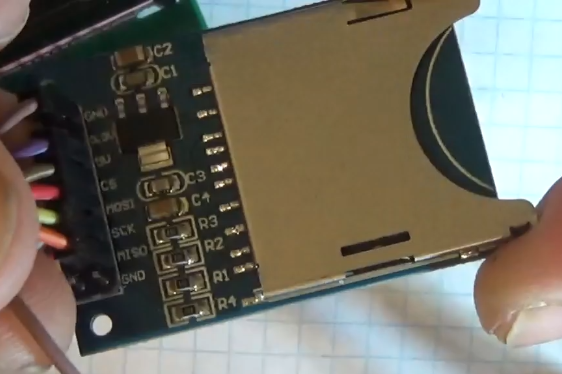
Также в протеусе не указано, а на самом деле мы будем питать нашу карту от отдлеьного питания, поставив микросхему, преобразующую 5 вольт в 3,3 вольт.

Вернее, мы не будем ничего ставить, а будем использовать готовый модуль, в котором уже всё установлено.

Вот так у нас всё выглядит в практической схеме



Вот так вот выглядит модуль с держателем



Найти такой модуль можно везде, стоит он копейки. Тот модуль, который конектится по SDIO, стоит дороже. Мы видим также, что на модуле уже установлена микросхема для понижения напряжения до 3,3 вольта. А подключаем питание мы только на контакт 5 вольт, а на 3,3 не подключаем ничего



Также на модуле установлены все делители для уровней, то есть данный модуль рассчитан именно на подключение к 5-вольтовым устройствам.

А флеш-карту для тестов я откопал на 32 мегабайта, именно мегабайта а не гигабайта



Данная флеш-карта была подарена вместе с каким-то фотоаппаратом и она нам лучше всего подойдёт для наших тестов, по крайней мере мы не будем думать, что тот или иной глюк у нас из-за слишком большого размера памяти на карте.

Код был весь взят также с прошлого занятия вместе с библиотекой дисплея, так как функцию, которую мы создали на прошлом уроке, мы будем очень активно использовать, только был конечно создан проект новый и назват соответственно **MYSD\_SPI**.

Удалим ненужные строки, в main() у нас останется только во это

int main(void)

{

  unsigned int i;

  port\_ini();

  LCD\_ini(); //инициализируем дисплей

  clearlcd();//очистим дисплей

  setpos(0,0);

  str\_lcd("String 1");

  setpos(2,1);

  str\_lcd("String 2");

  setpos(4,2);

  str\_lcd("String 3");

  setpos(6,3);

  str\_lcd("String 4");

  \_delay\_ms(2000);

  // for (i=0;i<=22;i++) {str80\_lcd(buffer2+i\*20);\_delay\_ms(1000);}

  while(1)

  {

  }

}

Так как мы посимвольно не будем выводить текст, то можно будет в переменной обойтись типом char

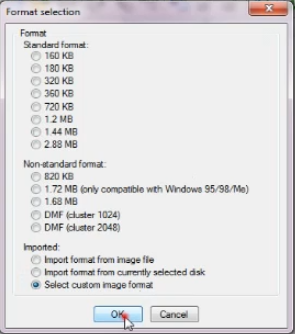
unsigned **char** i;

Теперь ещё один нюанс.

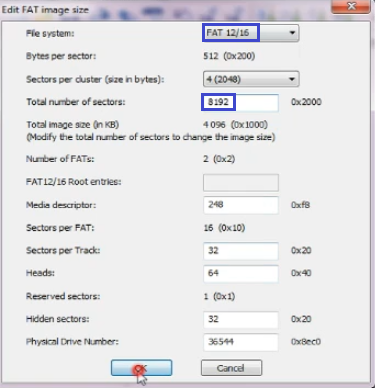
Чтобы нам работать с SD-картой в протеусе, нам мало добавить сам держатель с картой, необходимо также в его свойствах прикрепить файл образа флеш-карты.

Создать его не сложно. Одним из способов является создание с помощью программы WinImage.

Мы в ней стандартно создаём новый файл с помощью пункта меню File – > New. Выбираем в диалоге самый последний пункт и жмём "OK"

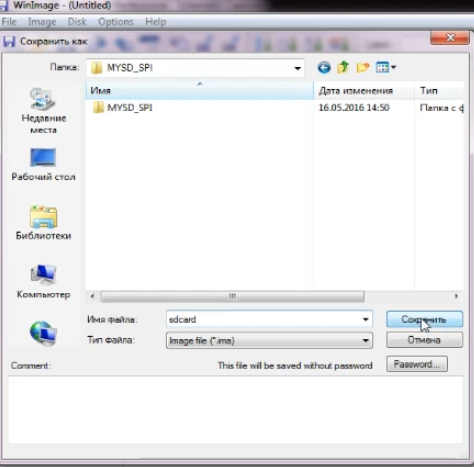


Для теста в протеусе нам вполне хватит размера 4 мегабайта, поэтому поменяем в следующем диалоге поле с номером секторов, а также выберем формат FAT12/16, потому что с 32-битной файловой системой немного другая специфика работы, и также нажмём "OK"

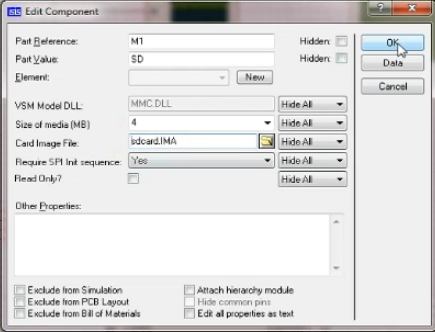


Вообще мы конечно можем оставить и FAT32, так как мы с файловой системой пока не работает, но в дальнейших частях занятия будет работа с файловой системой и мы будем именно работать с 12/16.

Затем мы сохраняем наш созданный файл с помощью пункта меню File -> Save As. И сохраняем мы его в ту папку, где у нас находится сохранённый проект протеуса. Назовём файл и нажмём "Сохранить"



Также затем нужно будет убедиться, что данный файл у нас получился не с аттрибутом "только для чтения" и после этого мы уже сможем его подключить  в протеусе. Надо будет вручную вписать имя файла, так как протеус требует какой-то свой формат и наш файл будет просто не виден



Путь нам никакой не нужен, так как файл у нас находится в папке с проектом. Жмём "ОК".

Инициализация шины нам не нужна, так как у нас SPI будет программный, с аппаратным флеш-карты работают корректно не все, то нам не надо будет использовать никаких регистров. Аппаратный конечно, лучше, но чтобы уяснить работу протокола досконально, надо ещё поработать и с программным, то есть подрыгать ножками портов. Вообще, глядя на схему, может показаться, что у нас всё аппаратно, так как я именно такие ножки выбрал, это потому, что я просто так выбрал, чтобы впоследствии когда-то может быть кто-то попытается всё-таки поработать с аппаратной реализацией шины.

Добавим макроподстановки для ножек порта

#include "main.h"

**#define MOSI 3**

**#define MISO 4**

**#define SCK 5**

**#define SS 2**

Добавим код для инициализации ножек в функцию инициализации портов

void port\_ini(void)

{

  PORTD=0x00;

  DDRD=0xFF;

**PORTB|=(1<<SS)|(1<<MISO)|(1<<MOSI);**

**DDRB|=(1<<SS)|(1<<MOSI)|(1<<SCK);**

}

Мы оставляем на вход ножку MISO, так как по умолчанию все биты в регистре равны нулю, и мы его просто не трогает. Также мы включаем сразу высокий уровень в MOSI и SS, а к MISO подтягиваем резистор.

Напишем функцию передачи байта по шине SPI

**void SPI\_SendByte (unsigned char byte)**

**{**

**}**

Добавим переменную для цикла и сам цикл

void SPI\_SendByte (unsigned char byte)

{

**unsigned char i;**

**for (i=0;i<8;i++) //движемся по битам байта**

**{**

**}**

}

Я думаю, понятно почем мы считаем до 8, так как битов мы передаём именно 8.

Ну и начнём их передавать потихоньку.

Проверим сначала самый левый бит, выделив его из всего байта маскированием, и, если он у нас равен 1, то выставим 1 и на шине MOSI, а если 0 – то не трогаем шину

for (i=0;i<8;i++) //движемся по битам байта

{

**if ((byte&0x80)==0x00)//проверяем левый бит**

**PORTB&=~(1<<MOSI); //если 0, то выставляем 0 на шине**

**else PORTB|=(1<<MOSI); //если 1, то выставляем 1**

Затем мы сдвинем наш байт влево на 1, чтобы старшим (левым) битом у нас стал следующий бит

else PORTB|=(1<<MOSI); //если 1, то выставляем 1

**byte<<=1; //сдвигаем влево, в сторону старшего для проверки следующего бита**

Дрыгнем ножкой SCK, чтобы сформировать импульс тактирования

    byte<<=1; //сдвигаем влево, в сторону старшего для проверки следующего бита

**PORTB|=(1<<SCK); //фронт на ножке SCK**

**asm("nop"); //1 такт подождём**

**PORTB&=~(1<<SCK); //спад на ножке SCK**

  }

}

# ****SPI. Карта SD. FAT****

В [**прошлой части**](https://narodstream.ru/avr-urok-33-spi-karta-sd-fat-chast-1/) 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++)**

**{**

**}**

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

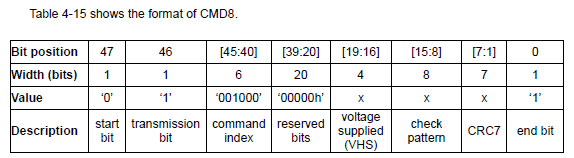
**}**

**return result;//вернем результат**

**}**

Теперь нам нужно написать функцию передачи команд в SD.

Откроем пример в даташите



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.

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

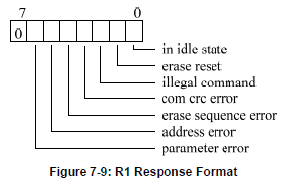
SPI\_SendByte(dt5); //контрольная сумма

**cnt=0;**

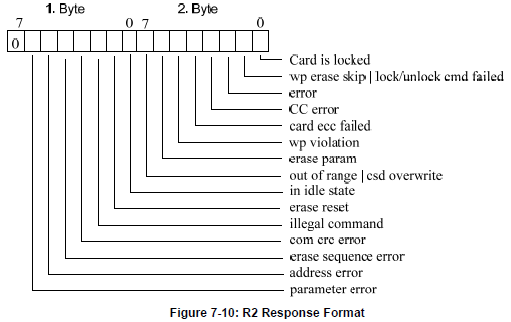
Причём команды, которые мы передаём в карту, могут не только с помощью них чем-то управлять, кое-что могут и возвращать.

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



Вот R2



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);

  r**eturn 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

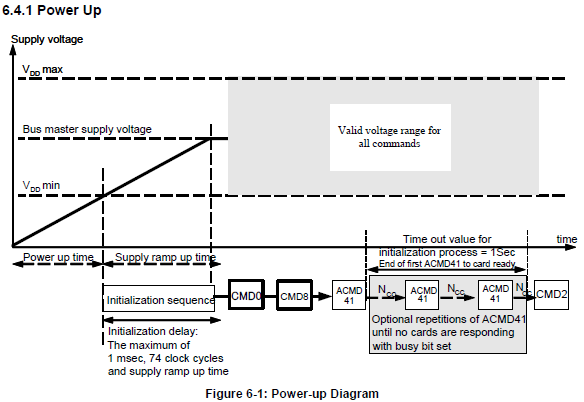
unsigned char SD\_Init(void)

{

**unsigned char i,temp;**

**long int cnt;**

Посмотрим рисунок из технической документации



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(0x40,0x00,0x00,0x00,0x00,0x95); //CMD0 Даташит стр 102 и 96**

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

Image14

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.

Image15

А это входные аргументы.

Проверим возвращённый результат

temp=SD\_cmd(0x40,0x00,0x00,0x00,0x00,0x95); //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 исползуя 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**

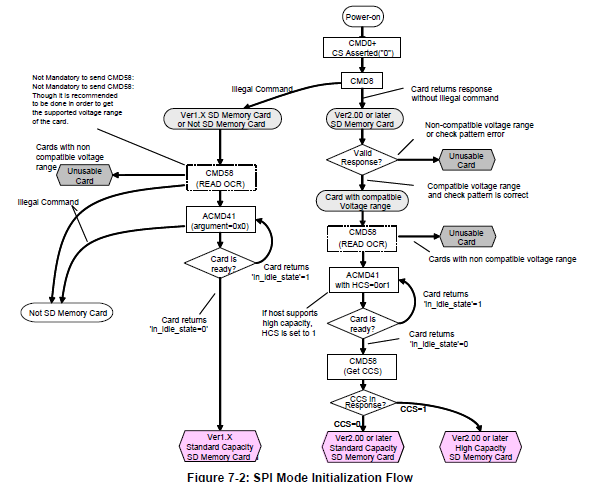
**if(cnt>=0xFFFF) return 2;**

**return 0;**

}

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

Посмотрим вот это дерево



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 будщем 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();**

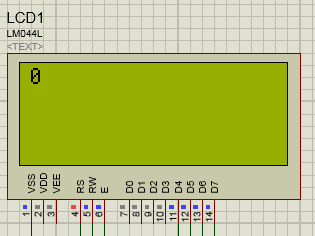
**sprintf(str,"%d",result);**

**clearlcd();//очистим дисплей**

**setpos(0,0);**

**str\_lcd(str);**

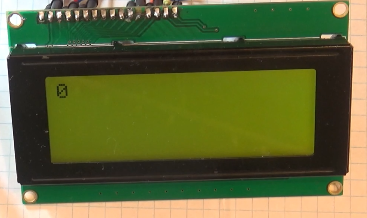
Соберём код и проверим пока результат в протеусе



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



Продолжим начатое дело по программированию и использованию флеш-карты SD. В [**предыдущей части**](https://narodstream.ru/avr-urok-33-spi-karta-sd-fat-chast-2/) нашего урока мы написали инициализацию нашей карты, теперь пришло время что-то с неё считать, а может даже и записать, пока не обращая внимание на структурирование данных в виде файловой системы.

Поэтому создадим функцию для записи блока с карты SD выше функции main(). Почему сразу записи, да потому, что читать нам ещё нечего, так как сначала надо что-то записать, чтобы потом прочитать и убедиться, что это именно то, что мы записали

**unsigned char SD\_Write\_Block (char\* bf, unsigned char dt1, unsigned char dt2, unsigned char dt3, unsigned char dt4)**

**{**

**}**

Вот сколько у нас входных параметров. Почему много, потому что во-первых мы передаём указатель на место в памяти, откуда мы будем писать данные в блок флеш-памяти карты. Во-вторых, чтобы инициировать передачу блока данных во флеш-карту, мы будем передавать определённую команду, в которой предусмотрено 4 основных параметра.

Создадим в функции две переменных

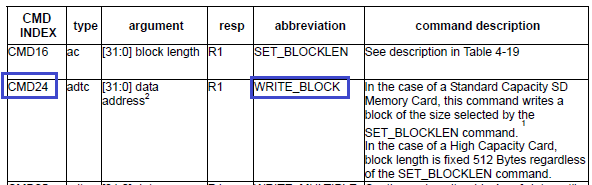
unsigned char SD\_Write\_Block (char\* bf, unsigned char dt1, unsigned char dt2, unsigned char dt3, unsigned char dt4)

{

**unsigned char result;**

**long int cnt;**

Теперь команда. Посмотрим техническую документацию и найдём команду для записи блока



Индекс команды у нас 24 или по-шестнадцатеричному – 0x18, первый бит, как мы знаем 0, второй всегда 1, поэтому будет 0x58.

Основным аргументом будет адрес данных. Поэтому напишем вызов функции передачи команды

long int cnt;

**result=SD\_cmd(0x58,dt1,dt2,dt3,dt4,0x95); //CMD24 даташит стр 51 и 97-98**

Так как все параметры у нас во входе функции, то мы их пока не увидим, увидим, когда будем нашу функцию записи вызывать.

Выйдем, если что-то не то, если всё то, то вычистим мусор из регистра сдвига карты

result=SD\_cmd(0x58,dt1,dt2,dt3,dt4,0x95); //CMD24 даташит стр 51 и 97-98

**if (result!=0x00) return 6; //Выйти, если результат не 0x00**

**SPI\_SendByte (0xFF);**

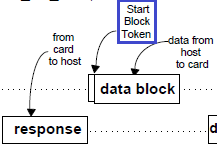
Вообще мы будем пробовать писать данные по адресу 0x400, так как если мы вдруг будем позже пробовать 32-битную файловую систему, то в первые байти мы писать не можем, так как там служебная информация файловой системы FAT32. В 0x300 мы писать не можем также, потому что блоки у нас по 512 байт и мы просто с такого адреса писать не можем, это не адрес блока.

Передадим начало буфера

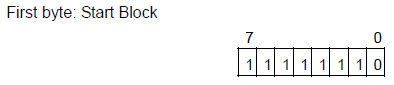
SPI\_SendByte (0xFF);

**SPI\_SendByte (0xFE); //Начало буфера**

Начало буфера – это своего рода метка, она входит в пакет передачи данных, состоящий из метки начала и собственно самих данных



Именно такая метка (0b11111110) должна быть для CMD 24, CMD17 и CMD18



Ну, и теперь непосредственно передаём данные из буфера и в конце контрольную сумму и любой байт

SPI\_SendByte (0xFE); //Начало буфера

**for (cnt=0;cnt<512;cnt++) SPI\_SendByte(bf[cnt]); //Данные**

**SPI\_SendByte (0xFF); //Котрольная сумма**

**SPI\_SendByte (0xFF);**

Контрольная сумма любая, поэтому по этому поводу не заморачиваемся.

Теперь примем результат команды из шины и произведём первичную проверку

SPI\_SendByte (0xFF);

**result=SPI\_ReceiveByte();**

**if ((result&0x05)!=0x05) return 6; //Выйти, если результат не 0x05 (Даташит стр 111)**

Аналогично, как и в предыдущей функции, создадим цикл и дождёмся свободного состояния карты, и, если всё нормально, то возвращаем 0

  if ((result&0x05)!=0x05) return 6; //Выйти, если результат не 0x05 (Даташит стр 111)

**cnt=0;**

**do { //Ждем окончания состояния BUSY**

**result=SPI\_ReceiveByte();**

**cnt++;**

**} while ( (result!=0xFF)&&(cnt<0xFFFF) );**

**if (cnt>=0xFFFF) return 6;**

**return 0;**

}

Вызовем нашу функцию в main() и отобразим результат на дисплее в следующей строке

str\_lcd(str);

**result=SD\_Write\_Block(buffer,0x00,0x00,0x04,0x00);//Запишем блок из буфера**

**sprintf(str,"%d",result);**

**setpos(0,1);**

**str\_lcd(str);**

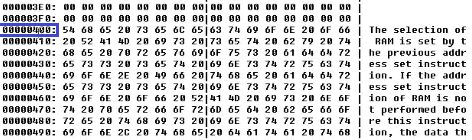
В первом пармаетре у нас буфер, в котором у нас есть строчка, с которой мы игрались в прошлом занятии.

Далше идёт у нас адрес, просто разбитый по 8 бит. А полностью получается 0x00000400, то есть именно тот адрес в который мы и будем пытаться писать блок.

Проверим сначала в протеусе



Хоть у нас пока нет функции чтения с карты, но проверить мы можем, открыв наш файл образа карты памяти



Мы видим, что вся наша информация записана в правильное место.

Теперь поиграем с чтением. Создадим функцию опять же над функцией main() и сразу напишем в неё локальные переменные

unsigned char SD\_Read\_Block (char\* bf, unsigned char dt1, unsigned char dt2, unsigned char dt3, unsigned char dt4)

{

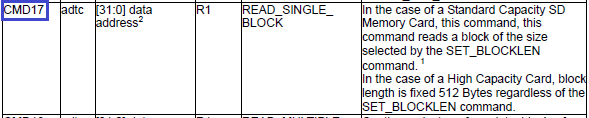
**unsigned char result;**

**long int cnt;**

}

Дальше всё скопируем с предыдущей функции записи и будем потихоньку вносить изменения

Команда будет у нас уже CMD17



Соответственно, CMD17 у нас превращается в 0x51, остальные параметры аналогичные

long int cnt;

**result=SD\_cmd (0x51,dt1,dt2,dt3,dt4,0x95); //CMD17 даташит стр 50 и 96**

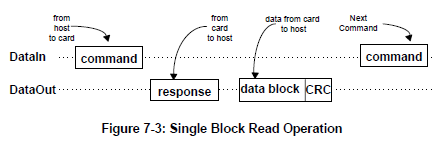
**if (result!=0x00) return 5; //Выйти, если результат не 0x00**

Затем также передача 0xFF с целью выждать время и заодно прочистить регистр SPI у карты

if (result!=0x00) return 5; //Выйти, если результат не 0x00

**SPI\_SendByte (0xFF);**

Посмотрим, как читаются данные с карты



Мы видим, что никаких меток у нас нет, поэтому сразу принимаем ответ и проводим первичную его проверк

SPI\_SendByte (0xFF);

**cnt=0;**

**do{ //Ждем начала блока**

**result=SPI\_ReceiveByte();**

**cnt++;**

**} while ( (result!=0xFE)&&(cnt<0xFFFF) );**

**if (cnt>=0xFFFF) return 5;**

А вот теперь уже читаем данные

if (cnt>=0xFFFF) return 5;

**for (cnt=0;cnt<512;cnt++) bf[cnt]=SPI\_ReceiveByte(); //получаем байты блока из шины в буфер**

Затем мы обязаны принять контрольную сумму, поэтому примем её символически, никак не проверяя результат, даже не видя его, и возвращаем 0

  for (cnt=0;cnt<512;cnt++) bf[cnt]=SPI\_ReceiveByte(); //получаем байты блока из шины в буфер

**SPI\_ReceiveByte(); //Получаем контрольную сумму**

**SPI\_ReceiveByte();**

**return 0;**

}

Теперь вызовем нашу функцию в main() и отобразим результат в третьей строке дисплея

str\_lcd(str);

**result=SD\_Read\_Block(buffer,0x00,0x00,0x04,0x00); //Считаем блок в буфер**

**sprintf(str,"%d",result);**

**setpos(0,2);**

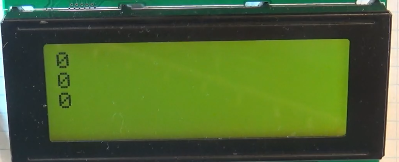
**str\_lcd(str);**

Попробуем собрать код и запустить его в протеусе



Всё у нас считалось.

Теперь проверим в живой схеме, прошив перед этим контроллер



Только мы не видим, что именно у нас считалось. Если мы отобразим наш буфер, в который мы считали, то толку от этого не будет, так как там уже есть этот текст. Поэтому мы должны назначить буфер для чтения, так как наш буфер с текстом нам уже не подойдёт, ибо в нём уже есть текст и мы просто не проверим работу функции. Поэтому создадим ещё один буфер, а предыдущий закомментируем, так как у нас не хватит на два буфера оперативной памяти и компилятор даст ошибку и не будет собирать код

//char buffer[512] ="The…"; //Буфер данных для записи/чтения

**char buffer2[512] ={}; //Буфер данных для чтения**

И также закомментируем код вызова и отображения результата функции записи в main(), а также исправим имя буфера в вызове функции чтения, затем добавим задержку, раскомментируем функцию вывода буфера на экран дисплея, исправив там также имя буфера

// result=SD\_Write\_Block(buffer,0x00,0x00,0x04,0x00);//Запишем блок из буфера

// sprintf(str,"%d",result);

// setpos(0,1);

// str\_lcd(str);

result=SD\_Read\_Block(**buffer2**,0x00,0x00,0x04,0x00); //Считаем блок в буфер

sprintf(str,"%d",result);

setpos(0,2);

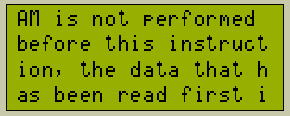
str\_lcd(str);

**\_delay\_ms(1000);**

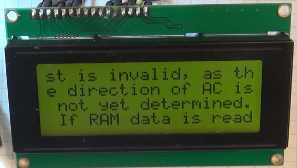
for (i=0;i<=22;i++) {str80\_lcd(**buffer2**+i\*20);\_delay\_ms(1000);}

while(1)

Сначала соберём код и проверим в протеусе



Затем прошьём контроллер и посмотрим результат вживую

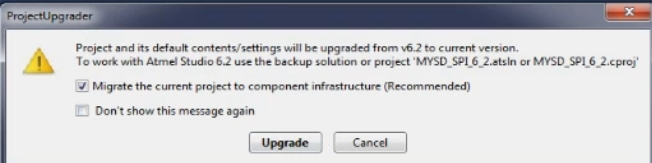


Только рутинную работу по написанию библиотеки для именно дисково-секторных процедур мы на себя брать не будем, так как для этого есть универсальная бесплатная готовая библиотека Petit FAT File System Module, которая размещена на сайте <http://elm-chan.org/> с лицензией Creative Commons (указание автора).

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

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.

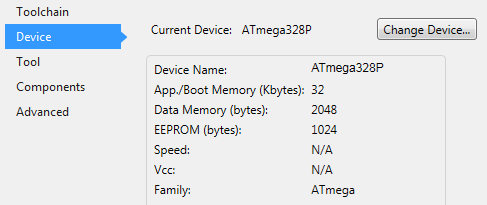
Проект у нас из прошлой части. Чтобы его преобразовать в проект для седьмой студии, его достаточно с помощью неё открыть и согласиться с преобразованием



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

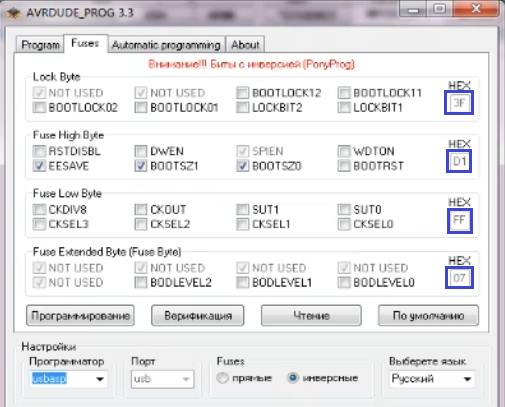


Больше ничего не меняем, в том числе и в коде. Возможно, если бы был у нас аппаратный SPI, то изменения бы какие-то были, врать не буду. Если был бы USART, то были бы точно.

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



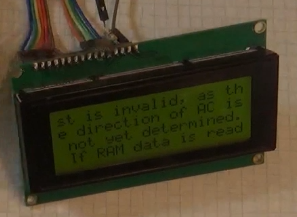
Затем переходим во фьюзы и выстанавливаем их вот так при условии, что на нашей плате мы ничего не трогали кроме замены контроллера, что тот же резонатор у нас отсался



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



Теперь вернёмся в проект.

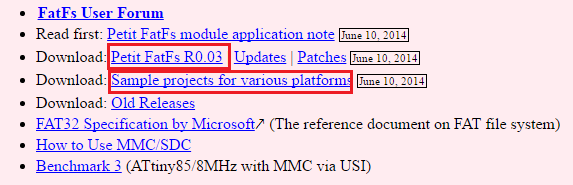
Скачаем сначала файлы с сайта. Заходим на вышеуказанный сайт, жмём там ссылочку Softwares, и в таблице выбираем Petit FatFs Module



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

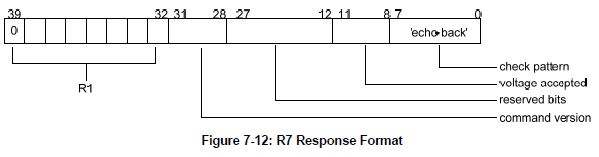
  } else { // SDv1 or MMCv3

  }

}

В качестве аргумента у нас 0x1AA.

Данная команда возвращает результат типа R7



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);

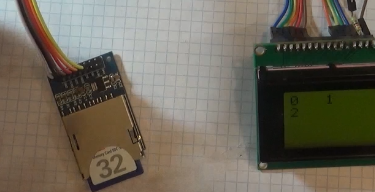
  str\_lcd(str);

  release\_spi();

  return ty ? 0 : STA\_NOINIT;

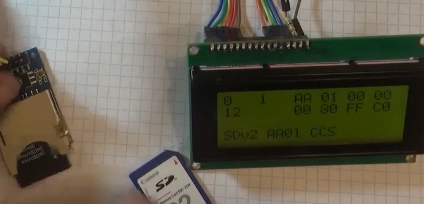
}

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



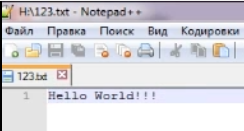
Определился ноль, означающий, что карта у нас самого простого типа, так как она у нас всего на 32 мегабайта.

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.

Чтобы считать какие-то данные из открытого файла, существует функция pf\_lseek. Поэтому также вызовем её в main() и отобразим результат

str\_lcd(str);

**res=pf\_lseek(0); //Установим курсор чтения на 0 в 123.txt**

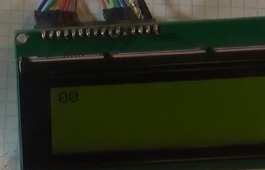
***sprintf*(str,"%d",res);**

**setpos(1,0);**

**str\_lcd(str);**

То есть читать мы будем с самого начала – с нулевого байта.

Соберём проект, прошьём контроллер и посмотрим результат



Мы видим два нуля, что свидетельствует об удачном выполнении функций

Далее уже вызовем непосредственно функцию чтения и также вернём результат

str\_lcd(str);

**res=pf\_read(buffer2,128,&s1);**

***sprintf*(str,"%d",res);**

**setpos(2,0);**

**str\_lcd(str);**

Здесь у функции появились уже какие-то параметры. первый параметр – указатель на наш строковый массив, который у нас инициализирован нулями. Второй – количество байтов для считывания. Функция данная "умная" и она, не смотря на то, что байтов у нас в файле меньше, чем мы пытаемся считать, считает до последнего байта и лишнее считывать пытаться не будет. Третий параметр – это формальный указатель, который ничего нам не даёт, но без него нельзя.

Соберём опять код, прошьём наш контроллер и посмотрим результат на дисплее

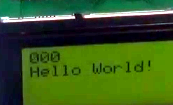


Отлично! Значит всё у нас прочиталось. Теперь осталось нам попробовать вывести на дисплей текст из файла

str\_lcd(str);

**setpos(0,1);**

**str\_lcd(buffer2);**



Всё содержимое нашего файла считалось. Также мы видим, что функция ничего лишнего не считала.

Теперь, прежде чем нам что-то записать на карту SD, размонтируем файловую систему

str\_lcd(buffer2);

**res=pf\_mount(0x00);//Демонтируем файловую систему карты**

***sprintf*(str,"%d",res);**

**setpos(3,0);**

**str\_lcd(str);**

Я думаю, нет смысла проверять, что система отмонтировалась. Вставим задержку, очистим дисплей, вызовем заново функцию монтирования и опять отобразим результат, можно с первой позиции, мы же очистили дисплей. И заново откроем тот же файл, вставив перед этим задержку.

str\_lcd(str);

***\_delay\_ms*(2000);**

**clearlcd();//очистим дисплей**

**res=pf\_mount(&fs);**

***sprintf*(str,"%d",res);**

**setpos(4,0);**

**str\_lcd(str);**

***\_delay\_ms*(2000);**

**clearlcd();//очистим дисплей**

**res=pf\_open("/123.txt"); //Попытка открыть файл 123.txt**

***sprintf*(str,"%d",res);**

**setpos(5,0);**

**str\_lcd(str);**

Пока проверять ничего не будем, потом всё проверится.

Указатель мы никуда устанавливать не будем, он автоматически встанет на нулевую позицию. Вызовем теперь функцию записи в файл

str\_lcd(str);

***sprintf*(buffer2,"World Hello!");**

**res=pf\_write(buffer2,*strlen*(buffer2),&s1);//Запишем текст из buffer2 в файл 123.txt**

***sprintf*(str,"%d",res);**

**setpos(6,0);**

**str\_lcd(str);**

Что мы тут заметили нового? Мы перевернули слова наоборот в тексте, а также использовали новую функцию для измерения длины строки. Измеряются байты до первого вхождения нулевого байта.

Только есть одна загвоздка. Применив данную функцию, мы ничего не запишем в файл, чтобы всё записалось, нужно финализировать запись. То есть мы можем записывать частями и каждый раз не финализировать, так будет процесс идти быстрее. А затем перед размонтированием файловой системы мы финализируем запись. Поэтому вызовем ещё функцию

str\_lcd(str);

**res=pf\_write(0,0,&s1);//Финализируем файл 123.txt**

***sprintf*(str,"%d",res);**

**setpos(7,0);**

**str\_lcd(str);**

Вот так. Осталось нам только отмонтировать файловую систему

str\_lcd(str);

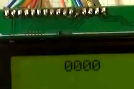
**res=pf\_mount(0x00);//Демонтируем файловую систему карты**

***sprintf*(str,"%d",res);**

**setpos(8,0);**

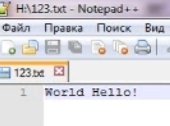
**str\_lcd(str);**

Соберём код, прошьём контроллер и проверим результат



Все результаты у нас хорошие.

Вынем карту из картоприёмника и поместим в картовод, чтобы считать файл на ПК. Считаем файл:



Теперь давайте отобразим новое содержание файла на дисплее, дописав код для чтения

**read file**

str\_lcd(str);

**clearlcd();//очистим дисплей**

**res=pf\_mount(&fs);**

***sprintf*(str,"%d",res);**

**setpos(9,0);**

**str\_lcd(str);**

***\_delay\_ms*(2000);**

**clearlcd();//очистим дисплей**

**res=pf\_open("/123.txt"); //Попытка открыть файл 123.txt**

***sprintf*(str,"%d",res);**

**setpos(10,0);**

**str\_lcd(str);**

**res=pf\_lseek(0); //Установим курсор чтения на 0 в 123.txt**

***sprintf*(str,"%d",res);**

**setpos(11,0);**

**str\_lcd(str);**

**res=pf\_read(buffer2,128,&s1);**

***sprintf*(str,"%d",res);**

**setpos(12,0);**

**str\_lcd(str);**

**setpos(0,2);**

**str\_lcd(buffer2);**

**res=pf\_mount(0x00);//Демонтируем файловую систему карты**

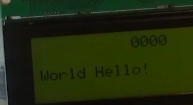
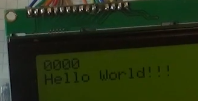
***sprintf*(str,"%d",res);**

**setpos(13,0);**

**str\_lcd(str);**

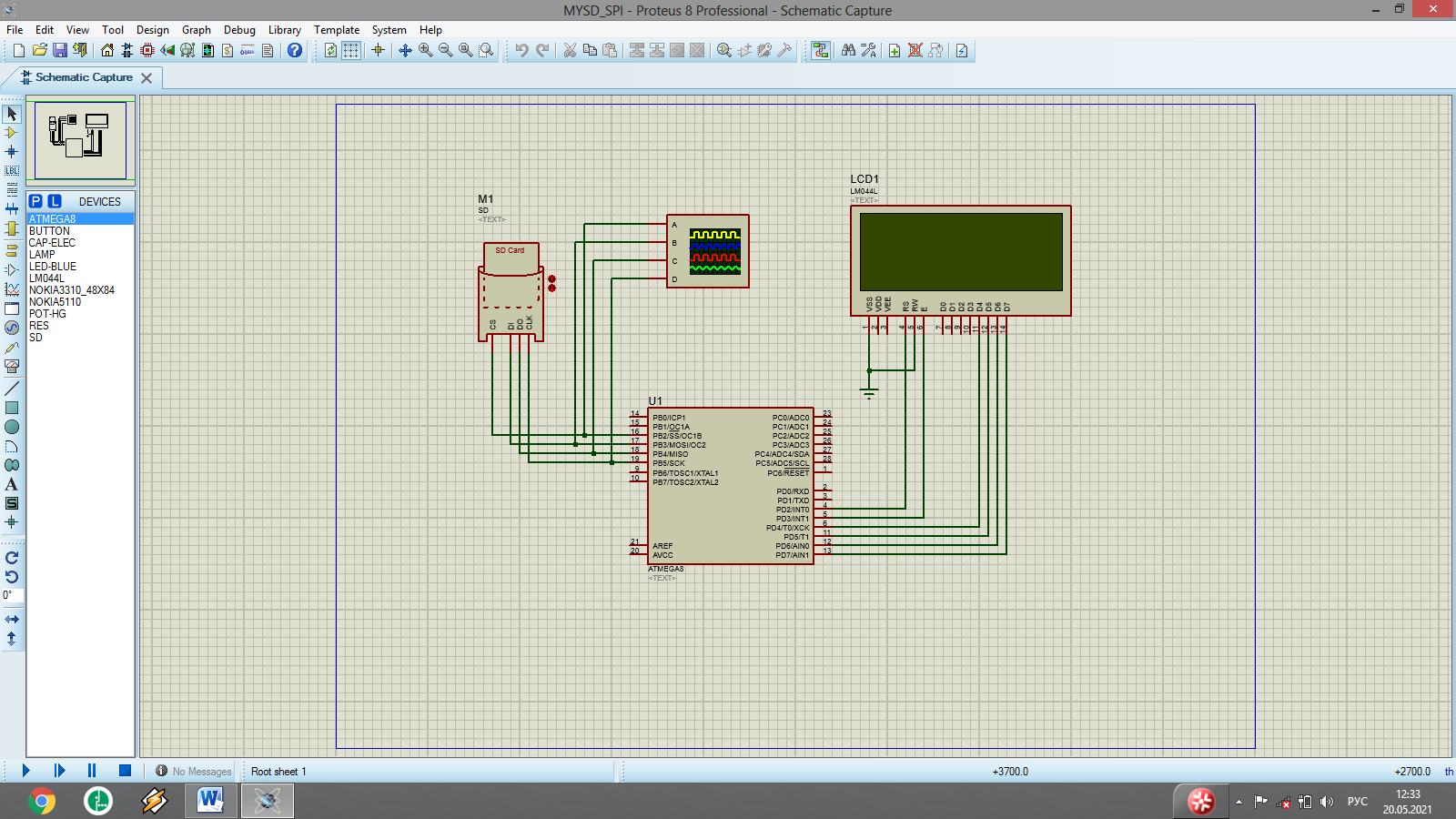
while(1)

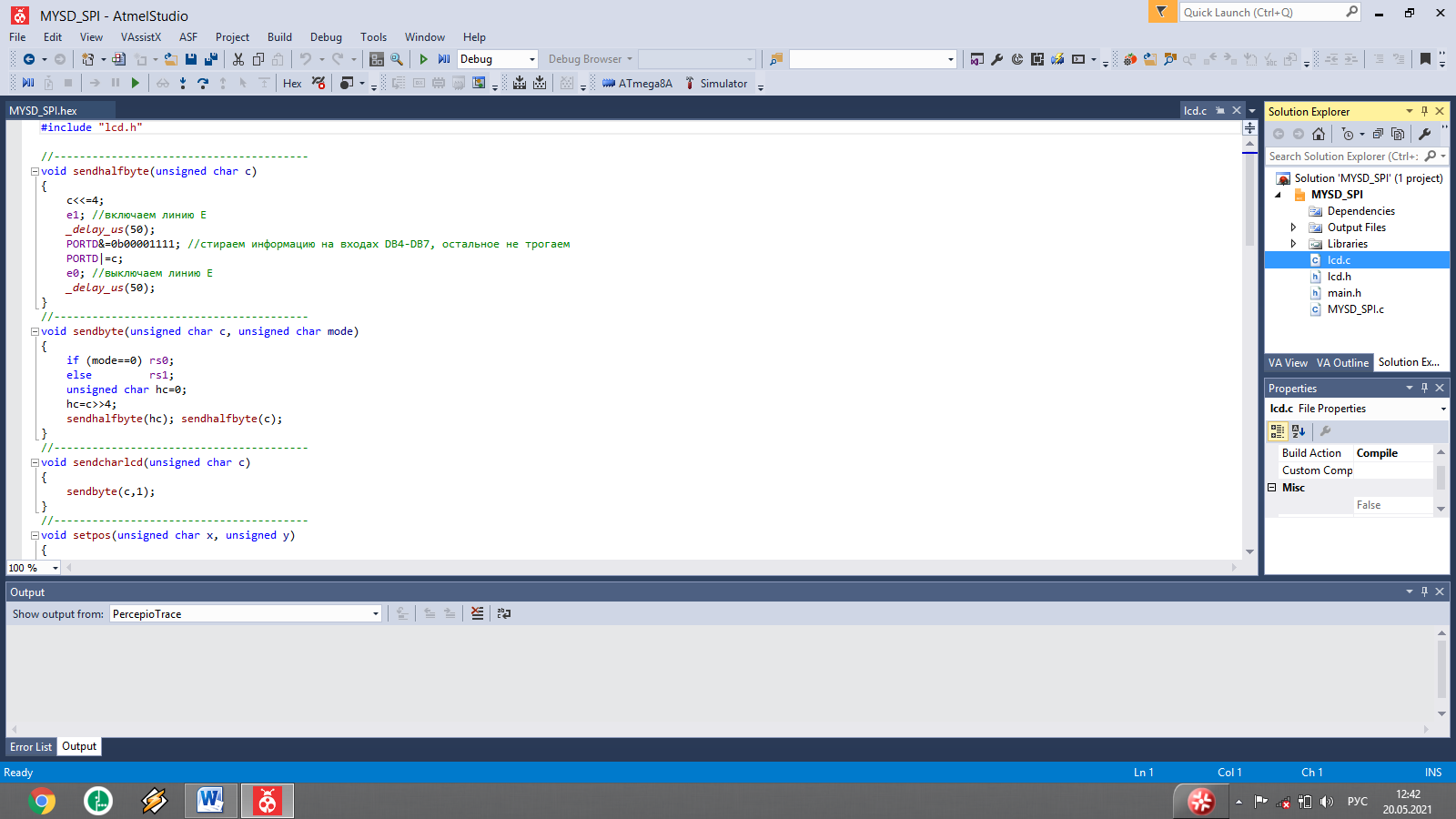
Проверим результат, воткнув перед этим SD-карту, но не ту же, так как на ней уже файл с новым содержимым, а другую



Всё отлично считывается и пишется.

ПРОГРАММНЫЙ КОД РАСПОЛОЖЕН ВНИЗУ





#include "lcd.h"

//----------------------------------------

void sendhalfbyte(unsigned char c)

{

c<<=4;

e1; //включаем линию Е

*\_delay\_us*(50);

PORTD&=0b00001111; //стираем информацию на входах DB4-DB7, остальное не трогаем

PORTD|=c;

e0; //выключаем линию Е

*\_delay\_us*(50);

}

//----------------------------------------

void sendbyte(unsigned char c, unsigned char mode)

{

if (mode==0) rs0;

else rs1;

unsigned char hc=0;

hc=c>>4;

sendhalfbyte(hc); sendhalfbyte(c);

}

//----------------------------------------

void sendcharlcd(unsigned char c)

{

sendbyte(c,1);

}

//----------------------------------------

void setpos(unsigned char x, unsigned y)

{

switch(y)

{

case 0:

sendbyte(x|0x80,0);

break;

case 1:

sendbyte((0x40+x)|0x80,0);

break;

case 2:

sendbyte((0x14+x)|0x80,0);

break;

case 3:

sendbyte((0x54+x)|0x80,0);

break;

}

}

//----------------------------------------

void LCD\_ini(void)

{

*\_delay\_ms*(15); //Ждем 15 мс (стр 45)

sendhalfbyte(0b00000011);

*\_delay\_ms*(4);

sendhalfbyte(0b00000011);

*\_delay\_us*(100);

sendhalfbyte(0b00000011);

*\_delay\_ms*(1);

sendhalfbyte(0b00000010);

*\_delay\_ms*(1);

sendbyte(0b00101000, 0); //4бит-режим (DL=0) и 2 линии (N=1)

*\_delay\_ms*(1);

sendbyte(0b00001100, 0); //включаем изображение на дисплее (D=1), курсоры никакие не включаем (C=0, B=0)

*\_delay\_ms*(1);

sendbyte(0b00000110, 0); //курсор (хоть он у нас и невидимый) будет двигаться влево

*\_delay\_ms*(1);

}

//----------------------------------------

void clearlcd(void)

{

sendbyte(0b00000001, 0);

*\_delay\_us*(1500);

}

//----------------------------------------

void str\_lcd (char str1[])

{

wchar\_t n;

for(n=0;str1[n]!='\0';n++)

sendcharlcd(str1[n]);

}

//----------------------------------------

void str80\_lcd (char str1[])

{

unsigned char n;

sendbyte(0x80,0);//1 строка

for(n=0;n<=19;n++)

sendcharlcd(str1[n]);

for(n=40;n<=59;n++)//на 3ю строку перейдём автоматически в силу организации DDRAM дисплея

sendcharlcd(str1[n]);

sendbyte(0xC0,0);//2 строка

for(n=20;n<=39;n++)

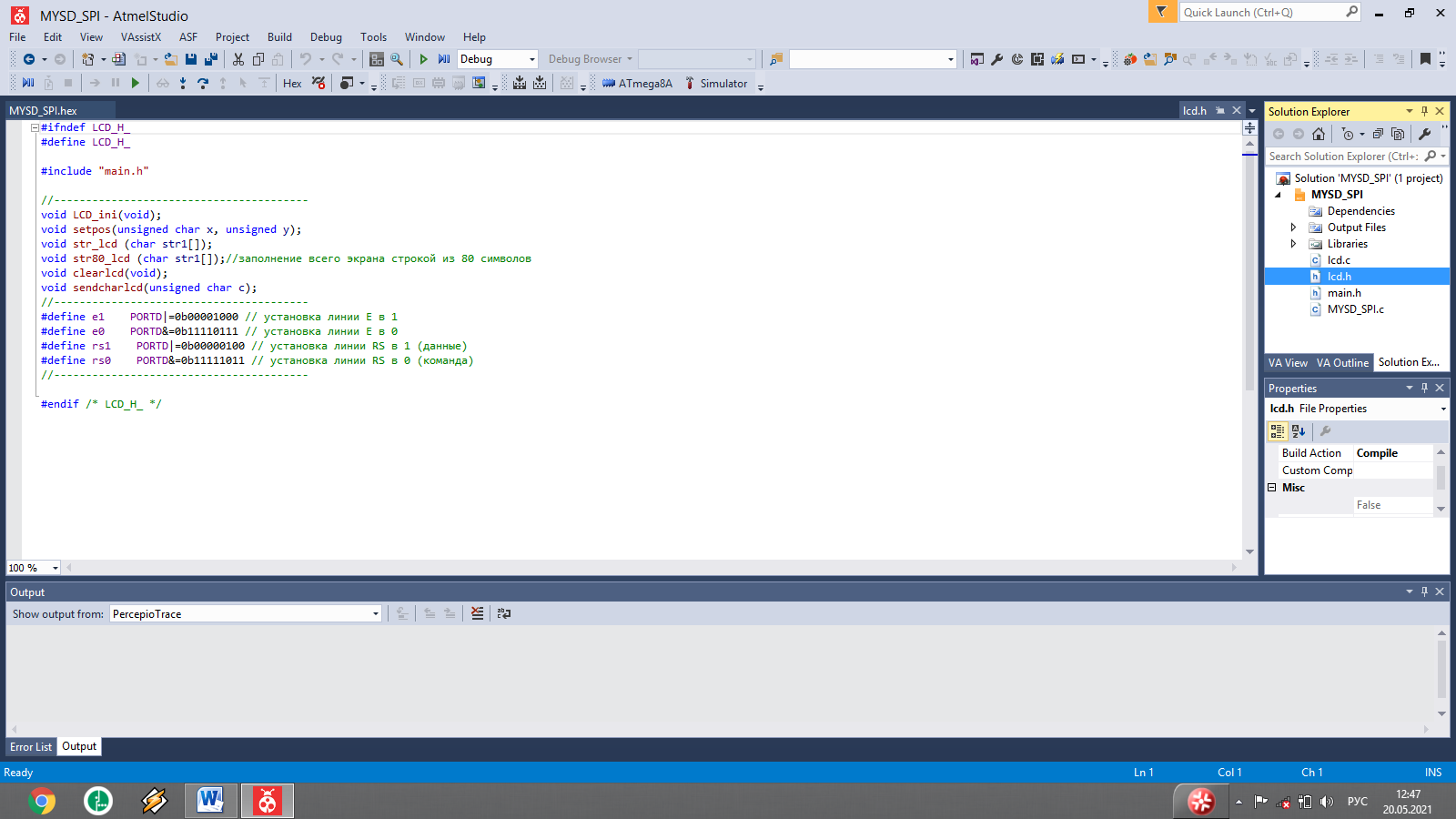
sendcharlcd(str1[n]);

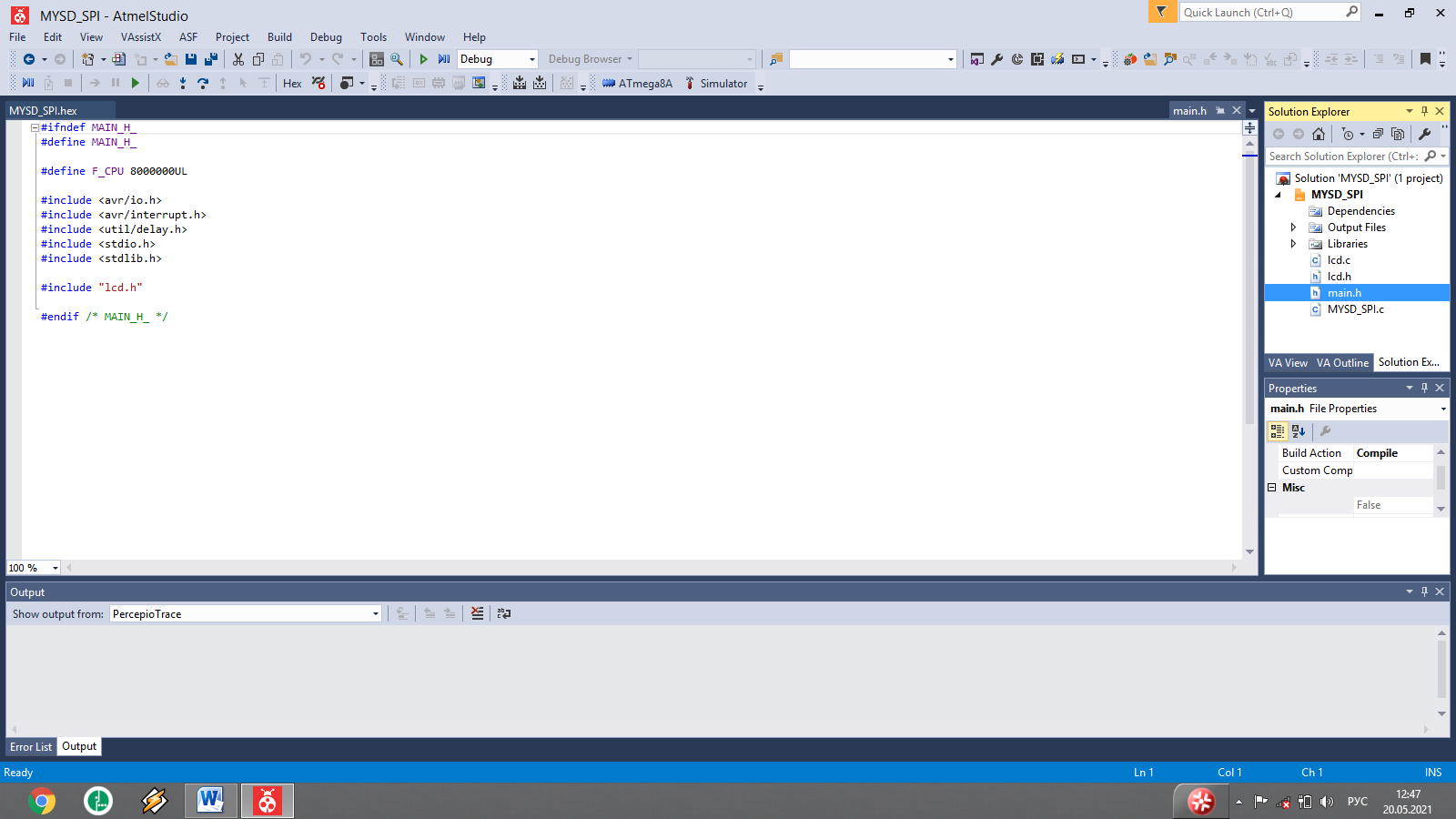
for(n=60;n<=79;n++)//на 4ю строку перейдём автоматически в силу организации DDRAM дисплея

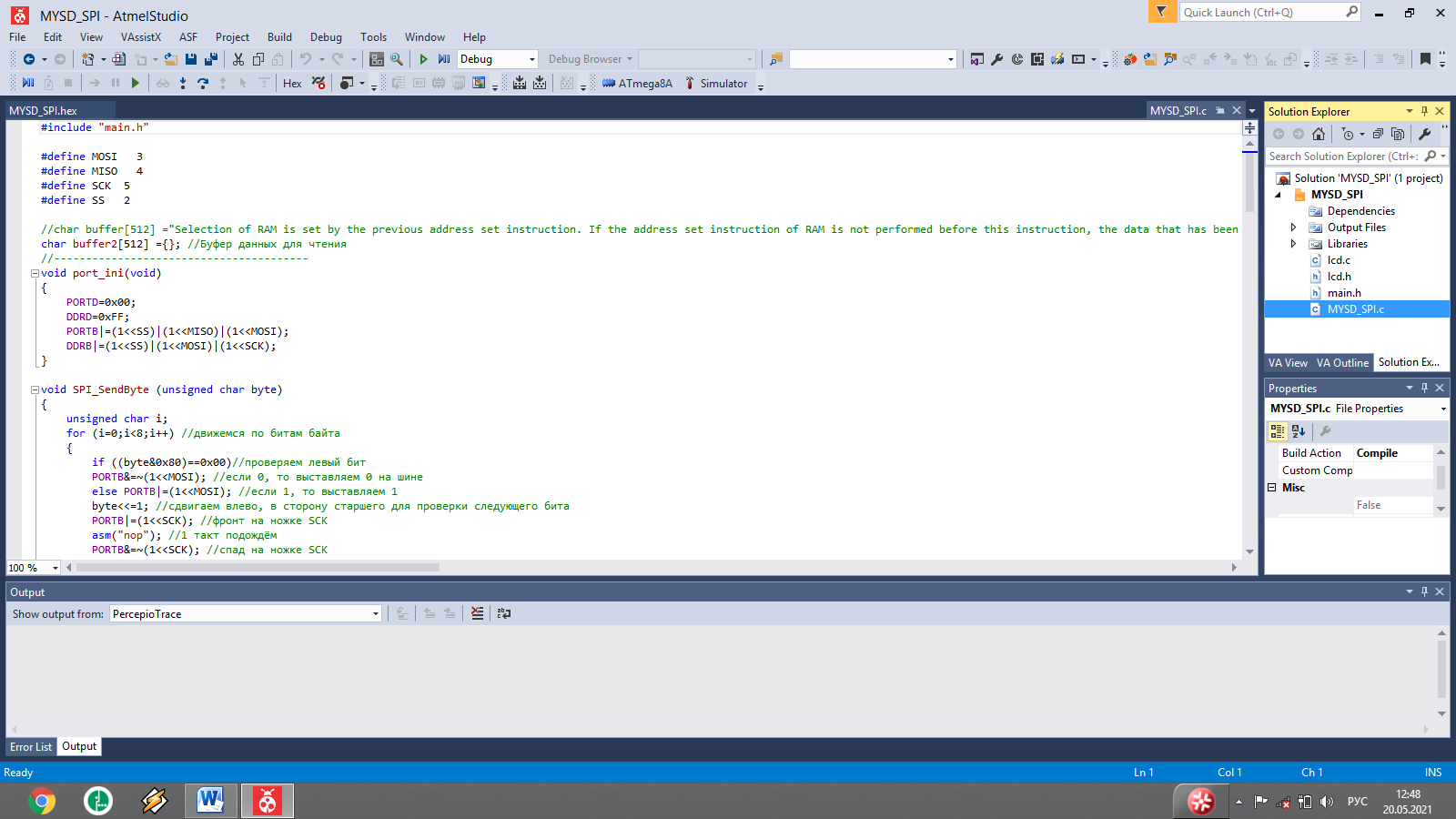
sendcharlcd(str1[n]);

}

//----------------------------------------







#include "main.h"

#define MOSI 3

#define MISO 4

#define SCK 5

#define SS 2

//char buffer[512] ="Selection of RAM is set by the previous address set instruction. If the address set instruction of RAM is not performed before this instruction, the data that has been read first is invalid, as the direction of AC is not yet determined. If RAM data is read several times without RAM address instructions set before, read operation, the correct RAM data can be obtained from the second. But the first data would be incorrect, as there is no time margin to transfer RAM data. In case of DDRAM read operation The..."; //Буфер данных для записи/чтения

char buffer2[512] ={}; //Буфер данных для чтения

//----------------------------------------

void port\_ini(void)

{

PORTD=0x00;

DDRD=0xFF;

PORTB|=(1<<SS)|(1<<MISO)|(1<<MOSI);

DDRB|=(1<<SS)|(1<<MOSI)|(1<<SCK);

}

void SPI\_SendByte (unsigned char byte)

{

unsigned char i;

for (i=0;i<8;i++) //движемся по битам байта

{

if ((byte&0x80)==0x00)//проверяем левый бит

PORTB&=~(1<<MOSI); //если 0, то выставляем 0 на шине

else PORTB|=(1<<MOSI); //если 1, то выставляем 1

byte<<=1; //сдвигаем влево, в сторону старшего для проверки следующего бита

PORTB|=(1<<SCK); //фронт на ножке SCK

asm("nop"); //1 такт подождём

PORTB&=~(1<<SCK); //спад на ножке SCK

}

}

unsigned char SPI\_ReceiveByte (void)

{

unsigned char i, result=0;

for(i=0;i<8;i++)

{

PORTB|=(1<<SCK); //фронт на ножке SCK

result<<=1; //сдвигаем влево, чтобы записать новый бит

if ((PINB&(1<<MISO))!=0x00) //запишем новый бит (в младший разряд)

result=result|0x01; //Считать бит данных

PORTB&=~(1<<SCK); //спад на ножке SCK

asm("nop"); //1 такт подождём

}

return result;

}

unsigned char SD\_cmd(char dt0, char dt1, char dt2, char dt3, char dt4, char dt5) //передача команды (пример даташит стр 40)

{

unsigned char result;

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

cnt=0;

do { //Ждем ответ в формате R1 (даташит стр 109)

result=SPI\_ReceiveByte();

cnt++;

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

return result;

}

unsigned char SD\_Init(void)

{

unsigned char i, temp;

long int cnt;

for (i=0;i<10;i++) //80 импульсов (не менее 74) Даташит стр 94

SPI\_SendByte(0xFF);

PORTB&=~(1<<SS); //опускаем SS

temp=SD\_cmd (0x40,0x00,0x00,0x00,0x00,0x95); // CMD0 Даташит стр 102 и 96

if (temp!=0x01) return 3; //Выйти, если ответ не 0х01 (результат

SPI\_SendByte (0xFF);

cnt=0;

do{

temp=SD\_cmd (0x41,0x00,0x00,0x00,0x00,0x95); //CMD1 (аналогично CMD0, только индекс меняется)

SPI\_SendByte (0xFF);

cnt++;

} while ( (temp!=0x00)&&(cnt<0xFFFF) ); //Ждем ответа R1

if (cnt>=0xFFFF) return 4;

return 0;

}

unsigned char SD\_Write\_Block (char\* bf, unsigned char dt1, unsigned char dt2, unsigned char dt3, unsigned char dt4)

{

unsigned char result;

long int cnt;

result=SD\_cmd(0x58,dt1,dt2,dt3,dt4,0x95); //CMD24 даташит стр 51 и 97-98

if (result!=0x00) return 6; //Выйти, если результат не 0x00

SPI\_SendByte (0xFF);

SPI\_SendByte (0xFE); //Начало буфера

for (cnt=0;cnt<512;cnt++) SPI\_SendByte(bf[cnt]); //Данные

SPI\_SendByte (0xFF); //Котрольная сумма

SPI\_SendByte (0xFF);

result=SPI\_ReceiveByte();

if ((result&0x05)!=0x05) return 6; //Выйти, если результат не 0x05 (Даташит стр 111)

cnt=0;

do { //Ждем окончания состояния BUSY

result=SPI\_ReceiveByte();

cnt++;

}while ( (result!=0xFF)&&(cnt<0xFFFF) );

if (cnt>=0xFFFF) return 6;

return 0;

}

unsigned char SD\_Read\_Block (char\* bf, unsigned char dt1, unsigned char dt2, unsigned char dt3, unsigned char dt4)

{

unsigned char result;

long int cnt;

result=SD\_cmd (0x51,dt1,dt2,dt3,dt4,0x95); //CMD17 даташит стр 50 и 96

if (result!=0x00) return 5; //Выйти, если результат не 0x00

SPI\_SendByte (0xFF);

cnt=0;

do{ //Ждем начала блока

result=SPI\_ReceiveByte();

cnt++;

} while ( (result!=0xFE)&&(cnt<0xFFFF) );

if (cnt>=0xFFFF) return 5;

for (cnt=0;cnt<512;cnt++) bf[cnt]=SPI\_ReceiveByte(); //получаем байты блока из шины в буфер

SPI\_ReceiveByte(); //Получаем контрольную сумму

SPI\_ReceiveByte();

return 0;

}

int main(void)

{

unsigned char i;

char str[10];

unsigned char result;

port\_ini();

LCD\_ini(); //инициализируем дисплей

clearlcd();//очистим дисплей

setpos(0,0);

str\_lcd("String 1");

setpos(2,1);

str\_lcd("String 2");

setpos(4,2);

str\_lcd("String 3");

setpos(6,3);

str\_lcd("String 4");

*\_delay\_ms*(1000);

asm("nop");

result=SD\_Init();

*sprintf*(str,"%d",result);

clearlcd();//очистим дисплей

setpos(0,0);

str\_lcd(str);

// result=SD\_Write\_Block(buffer,0x00,0x00,0x04,0x00);//Запишем блок из буфера

// sprintf(str,"%d",result);

// setpos(0,1);

// str\_lcd(str);

result=SD\_Read\_Block(buffer2,0x00,0x00,0x04,0x00); //Считаем блок в буфер

*sprintf*(str,"%d",result);

setpos(0,2);

str\_lcd(str);

*\_delay\_ms*(1000);

for (i=0;i<=22;i++) {str80\_lcd(buffer2+i\*20);*\_delay\_ms*(1000);}

while(1)

{

}

}