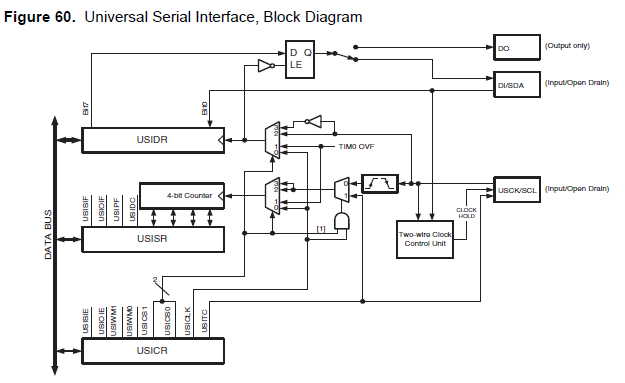
**Связь ATtiny2313 и Atmega8 по SPI**

.

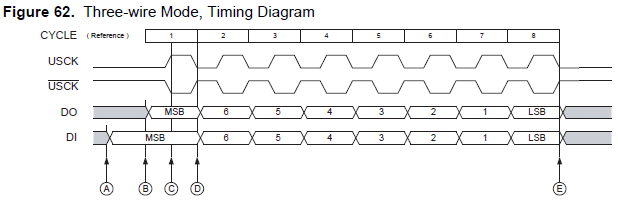
Посмотрим схему модуля USI



Вообщем, очевидная разница хотя бы в том, что как аппаратного тактирования в модуле нет, так как там нет даже генератора, есть только модуль, управляющий тактированием.

В качестве генератора тактовых импульсов можно использовать нулевой таймер, либо применить подключенный по определённой ножке внешний осцилятор, либо, как мы с вами поступим, мы будем весь этот процесс проводить в цикле.

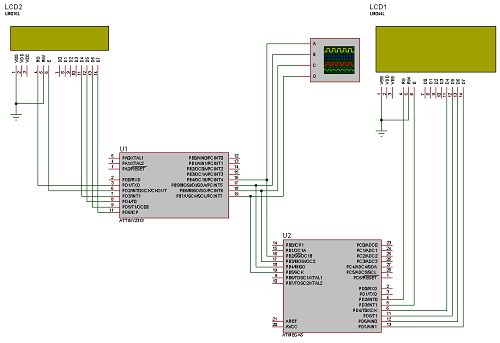
Также можно здесь настроить режим SPI с помощью определённого бита



Вообщем-то здесь мы видим, что процесс не сильно отличается от того же процесса на Atmega, единстенное нет ещё одного бита. Но это всё равно не отнимает у нас права пользоваться данной шиной для приёма и передачи, просто будут только 2 режима вместо 4.

А все регистры мы будем смотреть уже по мере написания кода.

Теперь схема в протеусе (нажмите на картинку для увеличения изображения)

[](https://narodstream.ru/wp-content/uploads/2017/01/Image02-8.png)

Здесь стандартная схема на ATtiny и стандартная схема на ATmega с подключенными дисплеями. Единственное различие, что все информационные входы, отвечающие за интерфейс SPI между собою соединены следующим образом

ATTiny2313       ATmega8

MOSI                 MOSI

MISO                 MISO

USCK                SCK

PB4                   SS

Ну и ещё подключен виртуальный осциллограф, чтобы смотреть, как передаются биты.

Сначала создадим проект для ATtiny. Имя мы ему присвоим **ATTINY\_SPI**.

Затем также создадим проект для Atmega8 и назовём его **ATMEGA\_SPI**. Код для данного проекта использован будет из проекта [**урока 12**](https://narodstream.ru/avr-urok-12-lcd-indikator-16x2-chast-1/). А библиотека для дисплея была взята с урока по часам, так как она там уже более дополненная. А код функции setpos в lcd.c был взят из проекта по подключению дисплея через переходник, так как дисплей мы подключили 4-строчный, чтобы у нас работали все 4 строки. Ну и допишем код для теста нижних строк в main()

setpos(2,1);

str\_lcd("String 2");

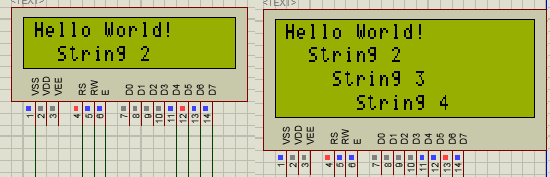
**setpos(4,2);**

**str\_lcd("String 3");**

**setpos(6,3);**

**str\_lcd("String 4");**

Соберём коды обоих проектов и посмотрим в протеусе, у нас должны работать оба дисплея



Пока начнём писать код в проекте **ATTINY\_SPI**.

Первым делом добавим переменную в функции main() для того, чтобы где-то хранить наш байт

int main(void)

{

**unsigned char n=0;//переменная для случайного числа**

Начнём писать теперь функцию для инициализации шины SPI. Мы её не будем называть USI, а будем использовать стандартное привычное название

**//—————————————-**

**void SPI\_init(void)**

**{**

**DDRB|=((1<<PORTB4)|(1<<PORTB6)|(1<<PORTB7));//Ножки USI на выход**

**DDRB&=~(1<<PORTB5);//Ножка DI на выход**

**PORTB&=~((1<<PORTB4)|(1<<PORTB6)|(1<<PORTB7));//Ножки USI в низкий уровень**

**}**

**//—————————————-**

Из кода функции и комментариев мы видим, что для инициализации шины мы не используем вообще никакие регистры, кроме регистров портов общего назначения, настроив в них определённые направления и значения.

Вызовем данную функцию в main()

LCD\_ini(); //Инициализируем дисплей

**SPI\_init(); //Инициализируем шину**

Также для красоты между выводом строк, а также в бесконечный цикл вставим небольшие задержки

str\_lcd("Hello World!");

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

setpos(2,1);

str\_lcd("String 2");

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

while(1)

{

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

Передавать мы будем случайное число, поэтому сгенерируем его в бесконечном цикле, используя функцию **rand()**, предварительно очистив дисплей и установив указатель на начало координат

while(1)

{

**clearlcd();**

**setpos(0,0);**

**n= (unsigned char) rand() % 256;**

  \_delay\_ms(1000);

То есть у нас будет генерироваться случайное число от 0 до 255, что с успехом влезет в передаваемый байт.

Создадим функцию для передачи байта в SPI над функцией main(), чтобы не писать прототип

**//—————————————-**

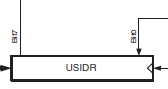
**void SPI\_SendByte(char byte)**

**{**

**}**

**//—————————————-**

Теперь идём в даташит и смотрим, куда же мы должны положить байт для передачи по SPI или USI. Оказывается для этого есть вот такой регистр



Ну и давайте туда и положим байт

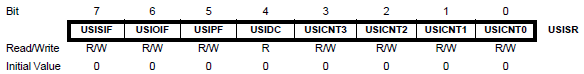
void SPI\_SendByte(char byte)

{

**USIDR = byte; //данные в регистр**

Теперь нам нужно как то заставить контроллер начать передавать данные

Для этого есть вот такой статусный регистр



А в данном регистре есть байт **USIOIF**, который является флагом начала передачи, не взирая на то, что регистр статусный

Поэтому напишем следующий код в нашу функцию

USIDR = byte; //данные в регистр

**USISR |= (1<<USIOIF);//установим флаг начала передачи**

Только данное действие не начнёт само собой передавать данные, так как нет тактирования. Здесь будет условный цикл

**while(!(USISR & (1<<USIOIF)))**

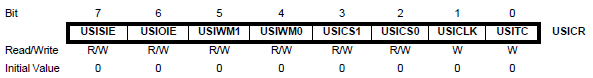
**{**

**}**

В данном цикле мы будем отслеживать сброс в ноль того же бита в том же регистре.

Теперь самое интересное – тело данного цикла. Тут не всё так просто.

Нужно будет немного ознакомиться со следующим регистром

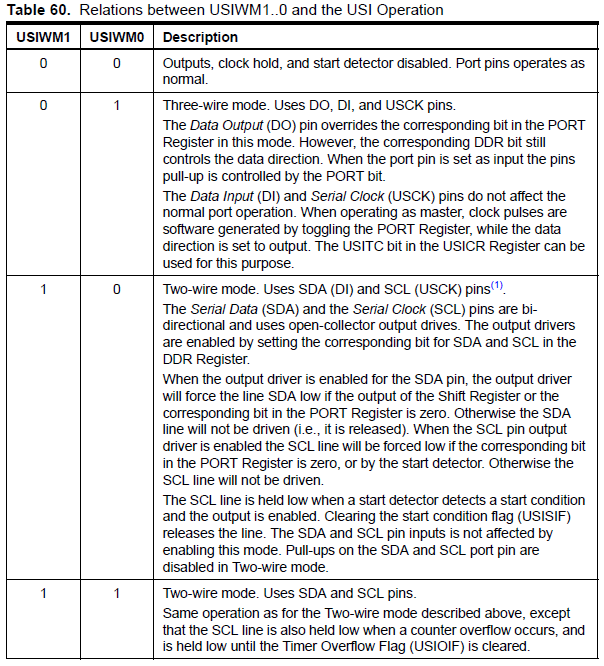


Это уже полноправный регистр управления. В нём существуют следующие биты

**USISIE (Start Condition Interrupt Enable)** – бит разрешения прерываний.

**USIOIE (Counter Overflow Interrupt Enable)** – флаг для прерываний по переполнению счётчика.

**USIWM1..0: (Wire Mode)** – биты режима. За счёт того, что этих бита два, то мы можем включить один из четырёх режимов:

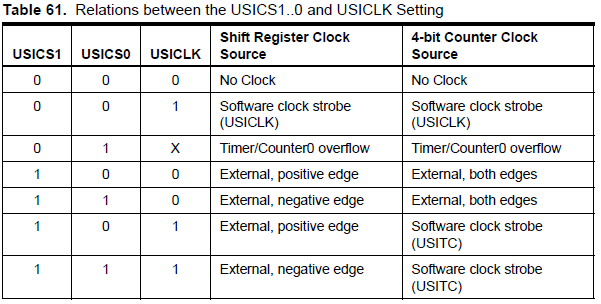


То есть так как шина USI она универсальная, может выступать не только как SPI, а и как I2C, но нас будет интересовать именно двухпроводной SPI, поэтому мы выберем режим второй, то есть включать будем бит **USIWM0**.

**USICS1..0 (Clock Source Select)** – биты установки варианта тактирования шины. Работают совместно со следующим битом.

**USICLK (Clock Strobe)** – бит стробирования регистра сдвига.

Вот таблица вариантов трёх вышеуказанных битов



Нас будет интересовать вариант шестой – программное тактирование и программное управление регистром сдвига.

**USITC (Toggle Clock Port Pin)** – бит переключения порта. В случае если контроллер является ведущим устройством, то включается в 1.

Таким образом, в нашей функции мы установим следующие биты управляющего регистра

while(!(USISR & (1<<USIOIF)))

{

**USICR |= ((1<<USIWM0)|(1<<USICS1)|(1<<USICLK)|(1<<USITC));//постепенно передаем байт**

Так как у нас всё происходит в цикле, то для обеспечения не слишком большой частоты нам нужно будет установить задержку

  USICR |= ((1<<USIWM0)|(1<<USICS1)|(1<<USICLK)|(1<<USITC));//постепенно передаем байт

**\_delay\_us(10);**

}

Может быть ATmega примет и с большей скоростью и задержка нам будет не нужна, но, во-первых, мы тогда очень сильно загрузим контроллер, а также нам тяжело будет с такой скоростью отследить процесс в протеусе на нашем виртуально осциллографе.

Вернёмся в бесконечный цикл и вызовем нашу новую функцию для того, чтобы передать байт ведомому устройству

n= (unsigned char) rand() % 256;

**SPI\_SendByte(n);**

 Затем нам нужно будет отобразить наш отправляемый байт на дисплее, для этого преобразуем его сначала в строку. Для этого кроме функции sprinf существует функция **itoa**.

SPI\_SendByte(n);

**itoa(n,str,10);**

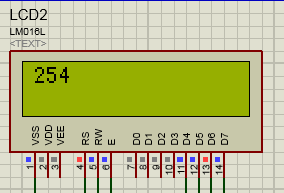
Первым входным аргументом в данной функции является преобразовываемая целочисленная величина, вторым – указатель на строку, а третьим – система исчисления. Так как мы будем выводить показания в десятичном виде, то у нас будет 10.

Затем мы этот байт непосредственно отобразим на дисплее

itoa(n,str,10);

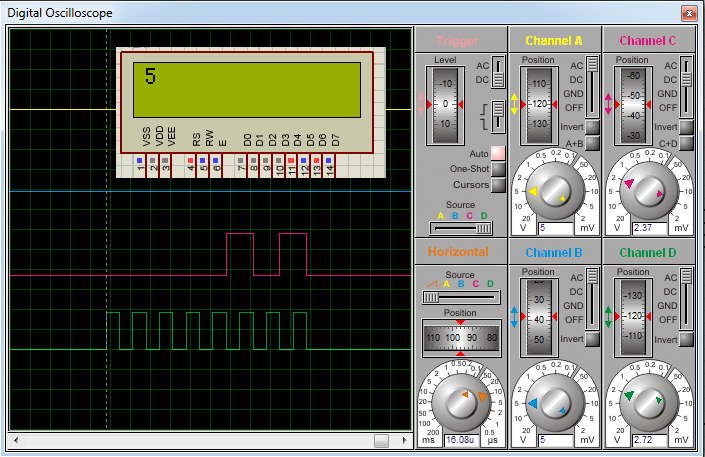
**str\_lcd(str);**

Соберём код и запустим его в протеусе. Посмотрим сначала отображение на дисплее



Байты нормально отображаются.

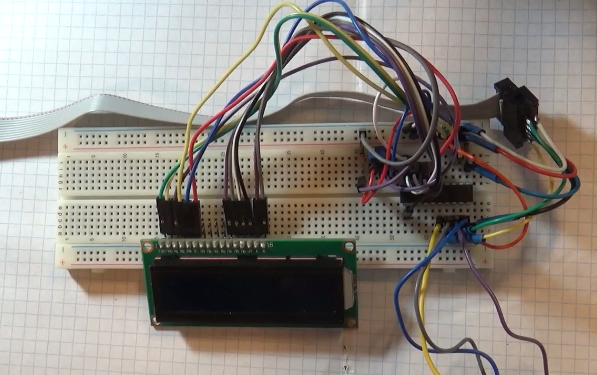
Теперь смотрим на осциллограмму и видим, что там также всё нормально



Посмотрим как именно передаётся байт. Зелёная осциллограмма показывает тактовые импульсы, а розовая – непосредственно шину передачи байта. Я дождался, когда будет байт попроще. У нас значение 5, если перевести в двоичный формат, то это 00000101. Так оно и есть на осциллограмме. Когда идут первые пять тактовых импульса, то шина передачи у нас находится в нуле, как только пятый импульс у нас заканчивается, то по его спадающему фронту у нас устанавливается шина передачи в единицу, чтобы во время возрастающего фронта шестого импулься она в этой единице находилась, так оно и есть. Затем по спаду шестого импульса шина передачи сбрасывается в ноль, а по спаду седьмого импульса, она поднимается.

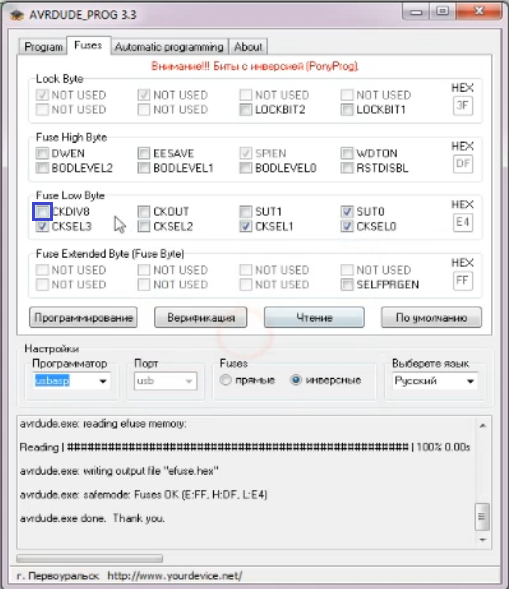
Вообщем, я думаю теперь нам стал более понятен протокол передачи данных по шине SPI.

Теперь посмотрим все соединения на макетной плате, касающиеся контроллера ATtiny2313



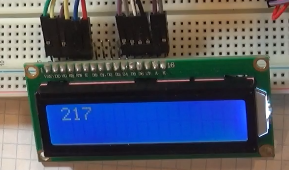
Всё у нас практически как в прошлом занятии, только мы ещё можем наблюдать, что у нас от платы куда-то отходят четыре провода. Это не что иное, как провода шины SPI для соединеия с другим контроллером, и также общий провод. Общие провода должны быть соединены. Хотя, возможно, будет работать и без этого, так как у нас шина USB подключена к программаторам от одного ПК. А вообще будет не так-то это всё просто, так как avrdude не будет знать, какой именно контроллер ей прошивать.

Попробуем прошить наш контроллер. Но прежде чем это делать, мы выберем в avrdude проект, считаем конфигурационные биты. Но главное не в этом. Ещё немаловажная информация! Перейдём на вкладку с фьюзами, считаем их и снимем вот этот вот бит, иначе у нас будет слишком маленькая скрость



Всё-таки он для 8 МГц не нужен, мы зря его установили на прошлом занятии.

Прошьём контроллер, и посмотрим



У нас всё нормально работает, только работу шины SPI мы сейчас не оценим, так как нам это отследить негде. Мы ещё не подключали другой контроллер, а настоящего осциллографа у меня, к сожалению нет, также нет даже логического анализатора. Возможно, в недалёком будущем всё это будет. Но пока вот так. Мы только видим, что переданные байты отображаются у нас на дисплее. А другим контроллером, а также дружбой между первым и вторым контроллером мы займёмся уже в [следующей части](https://narodstream.ru/avr-urok-31-svyaz-attiny2313-i-atmega8-po-spi-chast-2/) нашего такого вот интересного занятия.

 Теперь начнём работать с контроллером ATmega8, чтобы он сумел данные, посланные ему, принять. Данные мы уже принимали, но с той разницей, что контроллер наш был ведомым, теперь он уже будет ведущим, что конечно внесёт определённые коррективы в написание исходного кода.

Откроем проект и добавим функцию инициализации выше функции main(), в имя которой мы ещё добавим определённый суффикс, чтобы было видно что мы подключаем устройство как ведомое (Slave)

**//—————————————-**

**void SPI\_init\_SL(void)**

**{**

**}**

**//—————————————-**

Сначала в теле данной функции мы инициализируем задействованные ножки портов

void SPI\_init\_SL(void)

{

**DDRB |= (1<<PORTB4);//ножки SPI на выход**

**DDRB &= ~((1<<PORTB2)|(1<<PORTB3)|(1<<PORTB5));//ножки SPI на вход**

Как мы видим, что теперь большинство ножек у нас настраивается на вход, что и логично, у нас же ведомый контроллер, поэтому на выход мы включаем только ножку MISO.

Теперь включим шину, настроив управляющий регистр

  DDRB &= ~((1<<PORTB2)|(1<<PORTB3)|(1<<PORTB5));//ножки SPI на вход

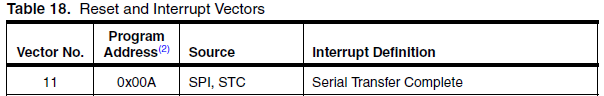
**SPCR = ((1<<SPE)|(1<<SPIE));//включим шину, включим прерывания**

}

Мы видим, что у нас почти ничего не изменилось по сравнению с прошлыми проектами. Единственное отличие в том, что мы не включаем в единицу, бит устанавливающий устройство ведущим.

Теперь вопрос, где нам вообще ловить этот пришедший байт. Можно конечно где-то его долго ждать, но мы так поступать не будем, для того мы и включили прерывания.

По прерываниям от шины **SPI** мы используем вот такой вот вектор



Добавим обработчик прерывания над функцией main()

**//—————————————-**

**ISR(SPI\_STC\_vect)//прерывание SPI прием байта**

**{**

**}**

**//—————————————-**

Добавим переменную и занесём в неё байт из регистра

ISR(SPI\_STC\_vect)//прерывание SPI прием байта

{

**unsigned char n;**

**n = SPDR;**

То есть данное прерывание у нас произойдёт тогда, когда ATmega8 закончит полностью приём байта, а вернее не приём а обмен байтами, у нас же всё идёт по кругу.

И затем аналогично тому, как мы делали на ATTiny, мы данный принятый байт преобразуем в строку и отобразим на дисплее, не забывая его перед этим очистить

  n = SPDR;

**clearlcd();//очистим дисплей**

**setpos(0,0);**

**itoa(n,str,10);**

**str\_lcd(str);**

**SPDR = n;**

}

Объявление переменной str мы из функции main() перенесём вверх и сделаем её глобальной, немного добавив количество элементов

#include "main.h"

//—————————————-

**char str[10];**

Также функцию инициализации шины SPI мы должны вызвать в main(), а также включить глобальные прерывания

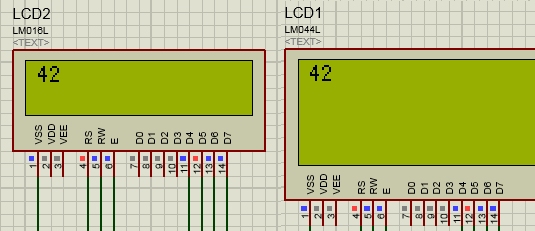
LCD\_ini(); //»нициализируем дисплей

**SPI\_init\_SL();//инициализируем SPI**

**sei();**

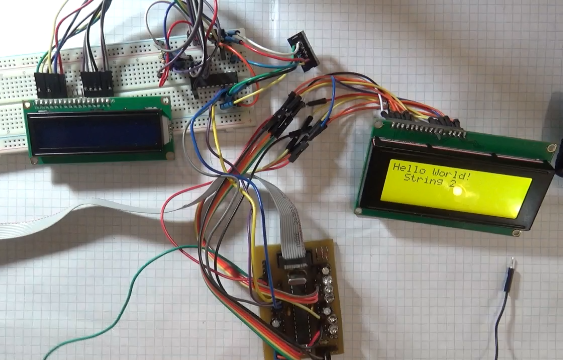
Ну, в принципе можно проверить наш код на деле.

Соберём наш код и запустим его в протеусе



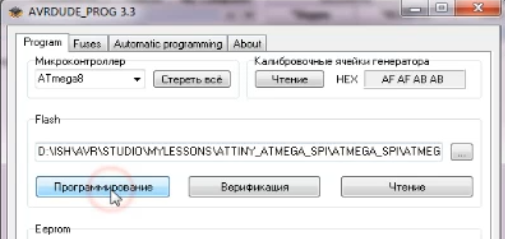
Всё у нас хорошо передаётся и принимается.

Прежде чем нам полюбоваться, как это всё происходит на практике, давайте посмотрим нашу конструкцию



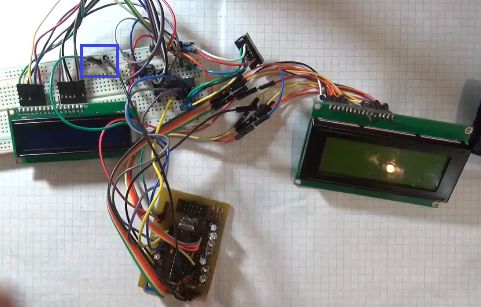
Всё у нас соединено, единственное то, что шину питания и общий провод я пока не подсоединял от одного контроллера к другому. Для начала их нужно будет прошить. Делается это потому, чтобы код прошивки не залился один и тот же сразу в два контроллера, так как у нас соединение происходит по SPI, а прошивка у нас именно по такому интерфейсу и происходит.

Сначала прошьём контроллер ATmega8

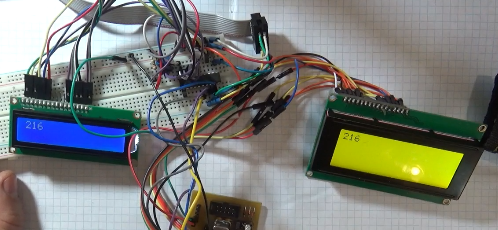


И, если мы сейчас посмотрим, на дисплей, то он нам будет только отображать нашу инициализацию, так как код отображения принятых байтов у нас расположен в обработчике прерывания, а нам никто ничего не посылает и прерывания, соответственно, никакие не будет происходить.

Так как другой контроллер у нас уже прошит, то мы не будем в него подключать программатор, а просто подведём от отладочной платы с ATmega к нему плюс и общий провод



Воткнём программатор в разъём любого контроллера, чтобы у нас появилось питание для нашей схемы, и посмотрим результат нашей работы



Казалось бы, на этом можно и завершить, но мы ещё не пробовали принимать байты на ведущем контроллере, а также передавать с ведомого.

Давайте и этим заодно займёмся, благо контроллеры наши это делать позволяют. То есть обмен у нас пойдёт полнодуплесный и сдвиговые регистры наши будут работать на полную.

Сначала давайте, сделаем всё для контроллера ATtiny, раз уж у нас все равно программатор воткнут в его гнездо.

Первым делом по эстетическим соображением назовём и функцию передачи байтов по-другому, так как она будет работать в обе стороны, а также добавим ей возвращаемый тип

**char** SPI\_**Change**Byte(char byte)

И в конце тела данной функции вернём байт

**return USIDR;**

}

Добавим ещё одну переменную в main() для принимаемого байта

unsigned char n=0**,m=0**;//переменная для случайного числа

Также изменим имя и в вызове функции обмена в бесконечном цикле и добавим возврат значения

**m** = SPI\_**Change**Byte(n);

То есть здесь мы передадим n, а примем m.

Ну, и приблизительно таким же образом мы отобразим на дисплее ещё и принятый байт, а чтобы циферки между собой не соединились, мы спозиционируем перед этим курсор дисплея в другое место

str\_lcd(str);

**itoa(m,str,10);**

**setpos(0,1);**

**str\_lcd(str);**

Теперь перейдём к проекту с контроллером ATmega8 и подумаем, как же нам оттуда отправить байт.

Оказывается это сделать ещё проще. В обработчике прерывания в момент, когда принялся байт, то у нас регистр **SPDR** очищается. Поэтому давайте мы в него обратно запишем значение нашей переменной n

  str\_lcd(str);

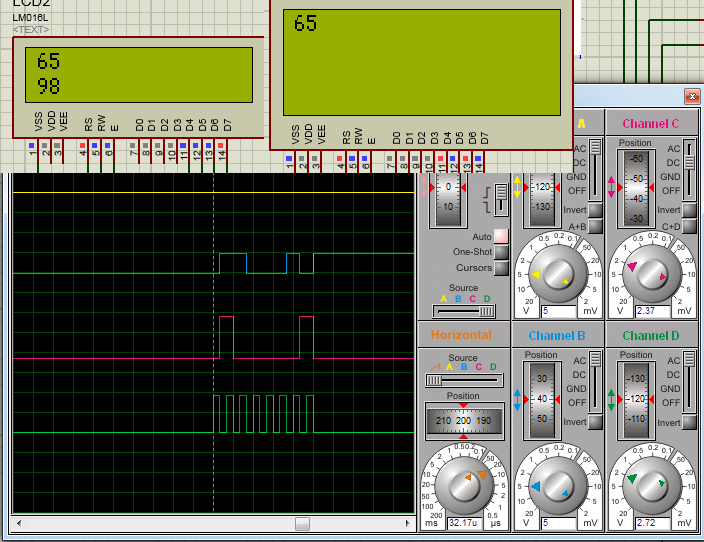
**SPDR = n;**

}

И записанный в SPDR байт в следующем цикле обмена байтами в регистре уйдёт обратно в контроллер ATtiny2313.

Соберём сначала наши оба проекта.

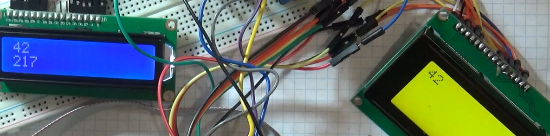
Посмотрим результат в протеусе



Мы видим, что всё у нас отлично передаётся, также на другой ножке на осциллограмме появилась передача байтов.

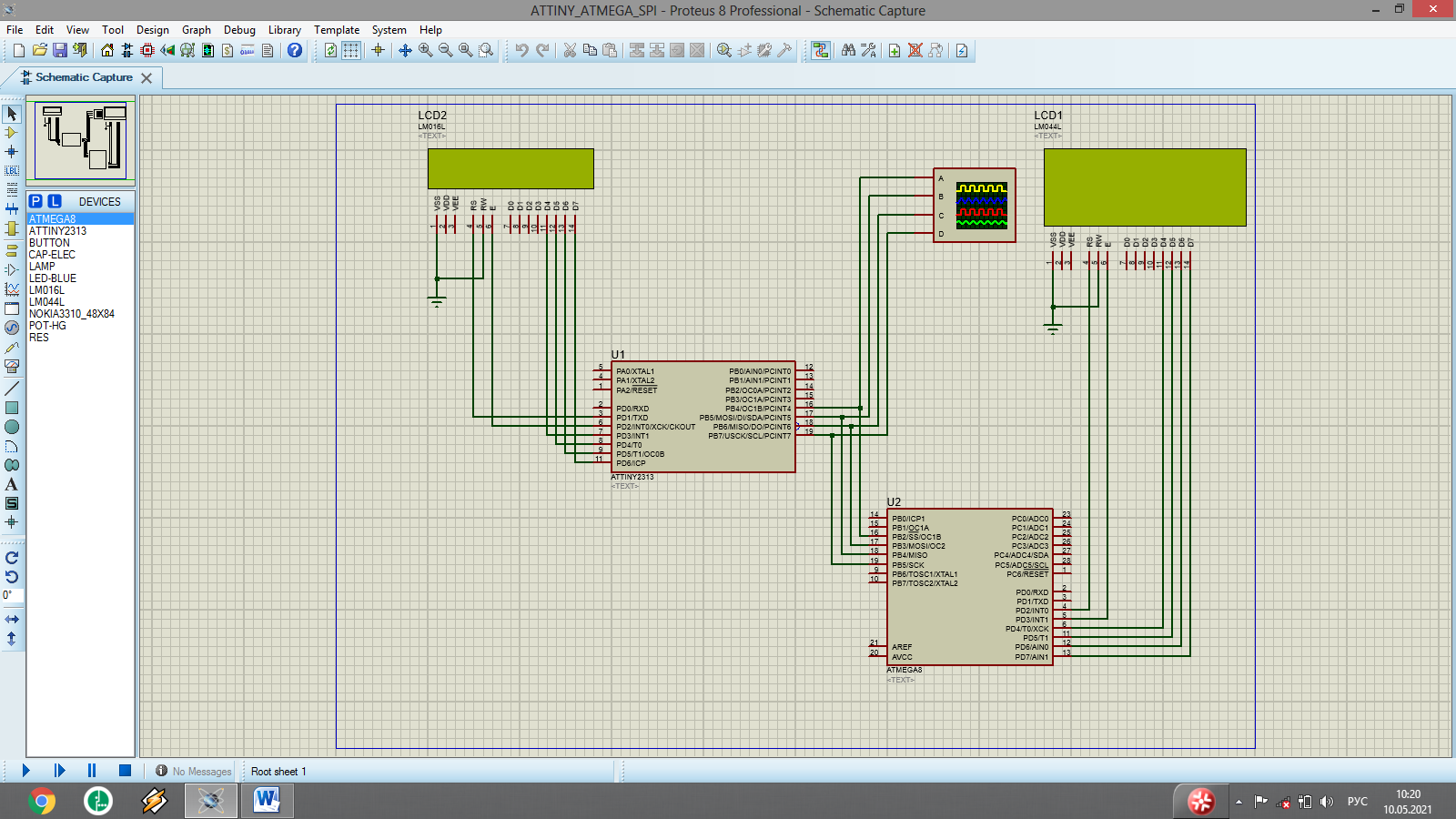
Теперь нам нужно проверить всё на практической схеме.

Опять разъединим межконтроллерное питание и прошьём сначала контроллер ATtiny2313, затем, не соединяя питание, переключим разъём программатора в отладочную плату с контроллером ATmega8 и прошьём его. Затем отключим общее питание, соединим провод питания и общий провод между обеими схемами, подадим общее питание и посмотрим результат

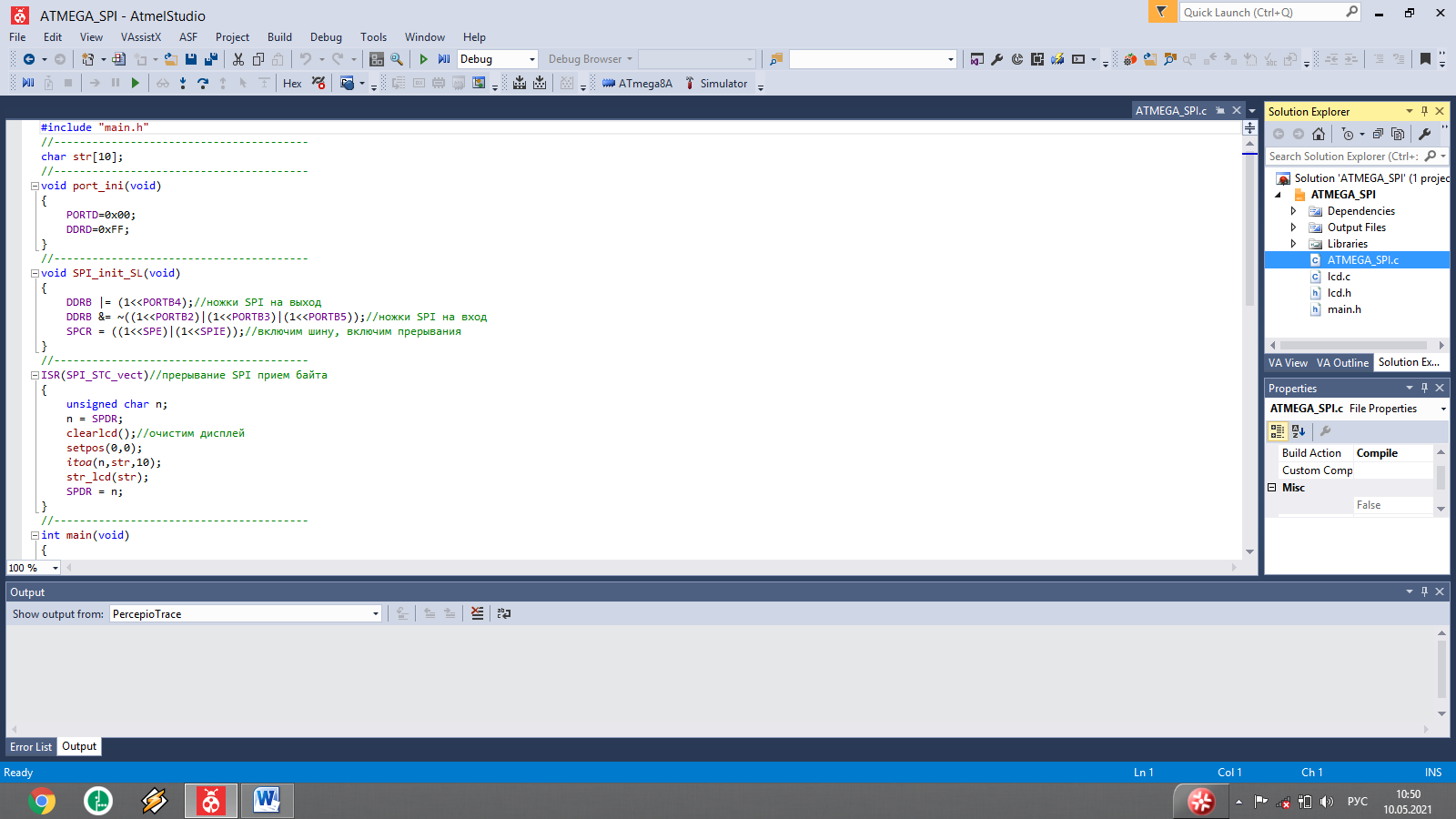


КОД ДЛЯ ПРОТЕУСА И АТМЕЛ СТУДИО

ПРЕДСТАВЛЕН НИЖЕ



КОД ДЛЯ ATMEGA 8



#include "main.h"

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

char str[10];

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

void port\_ini(void)

{

PORTD=0x00;

DDRD=0xFF;

}

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

void SPI\_init\_SL(void)

{

DDRB |= (1<<PORTB4);//ножки SPI на выход

DDRB &= ~((1<<PORTB2)|(1<<PORTB3)|(1<<PORTB5));//ножки SPI на вход

SPCR = ((1<<SPE)|(1<<SPIE));//включим шину, включим прерывания

}

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

ISR(SPI\_STC\_vect)//прерывание SPI прием байта

{

unsigned char n;

n = SPDR;

clearlcd();//очистим дисплей

setpos(0,0);

*itoa*(n,str,10);

str\_lcd(str);

SPDR = n;

}

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

int main(void)

{

port\_ini(); //»нициализируем порты

LCD\_ini(); //»нициализируем дисплей

SPI\_init\_SL();//инициализируем SPI

sei();

clearlcd();//очистим дисплей

setpos(0,0);

str\_lcd("Hello World!");

setpos(2,1);

str\_lcd("String 2");

setpos(4,2);

str\_lcd("String 3");

setpos(6,3);

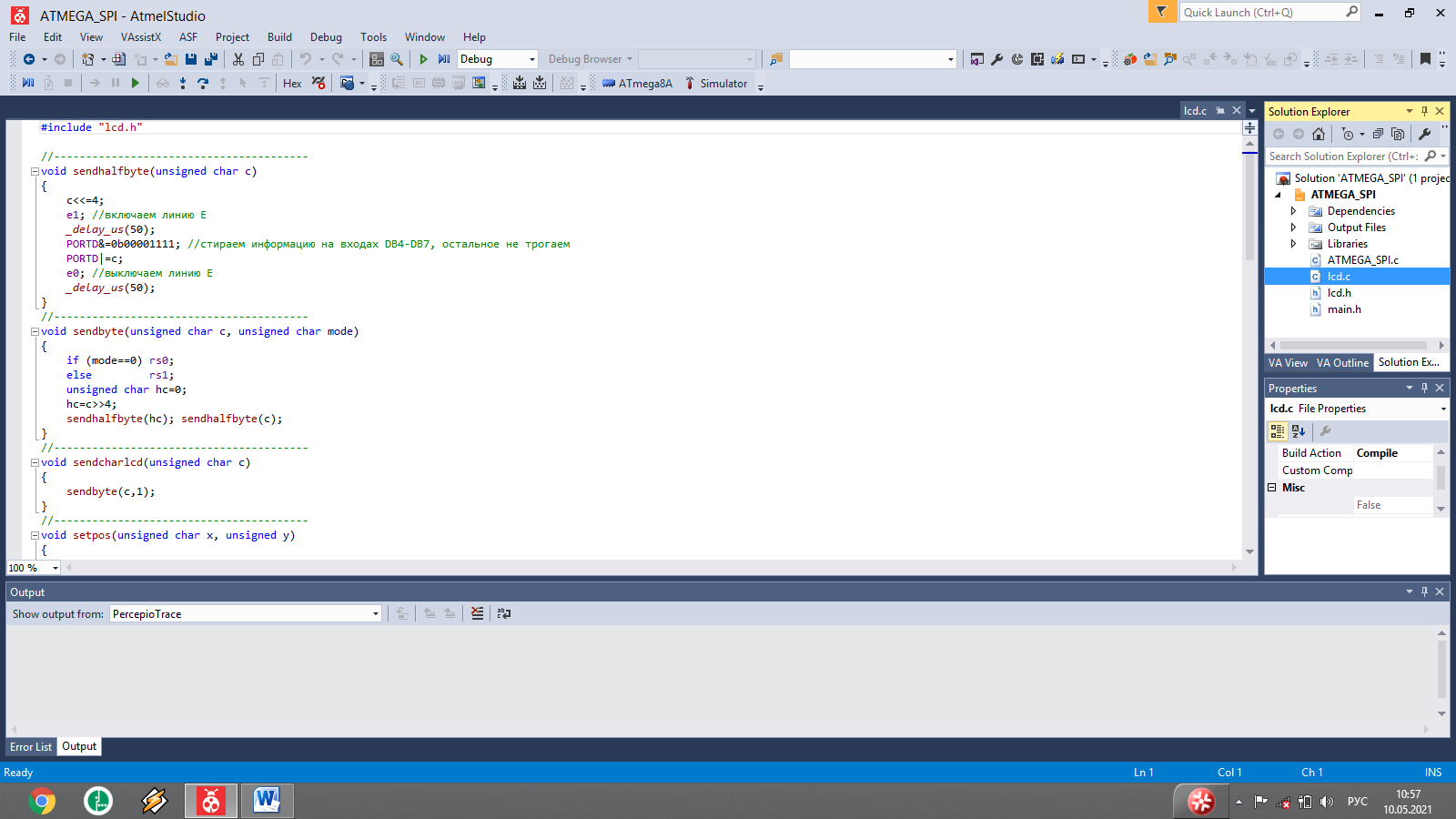
str\_lcd("String 4");

while(1)

{

}

}



#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[])

{

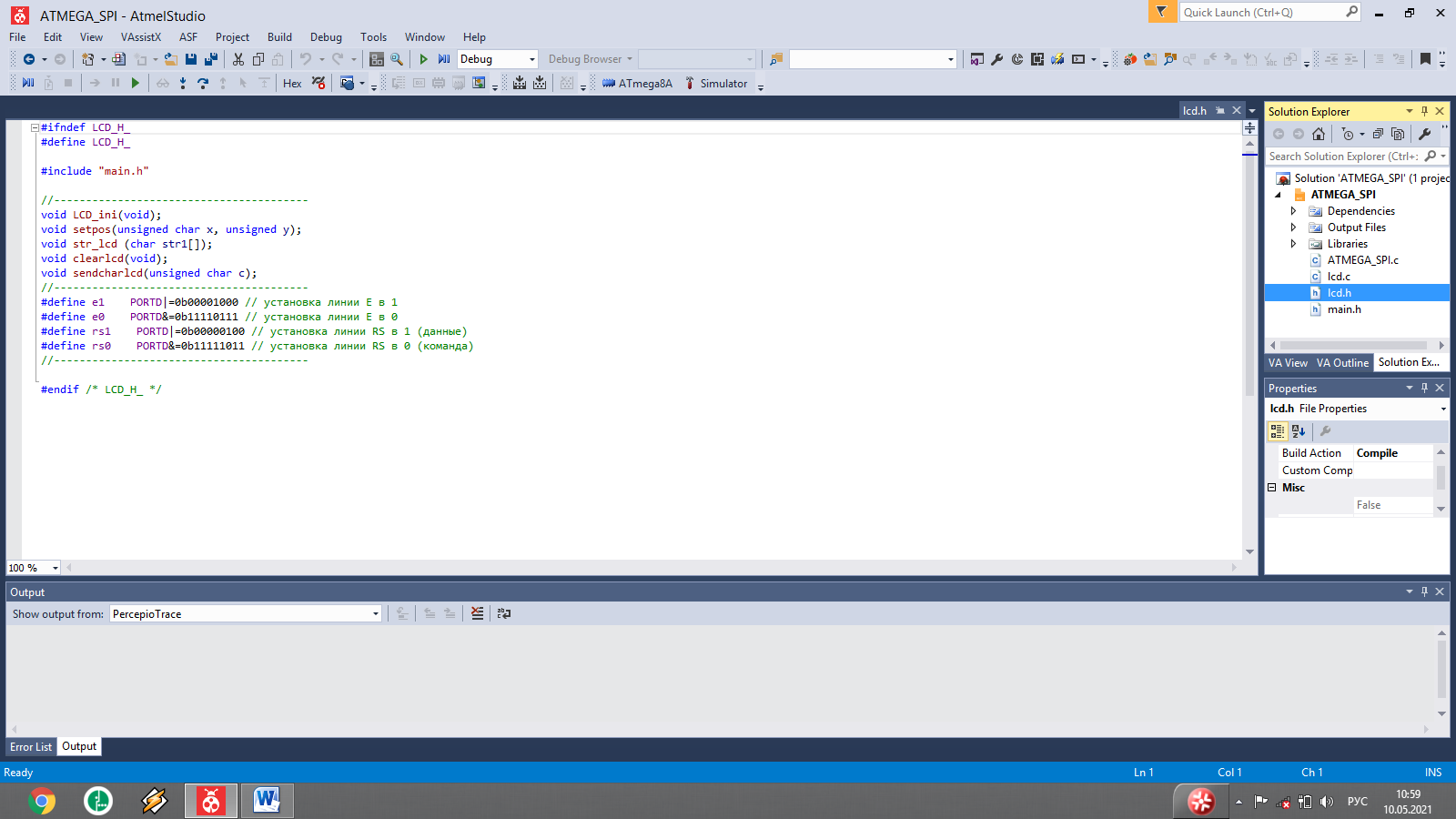
wchar\_t n;

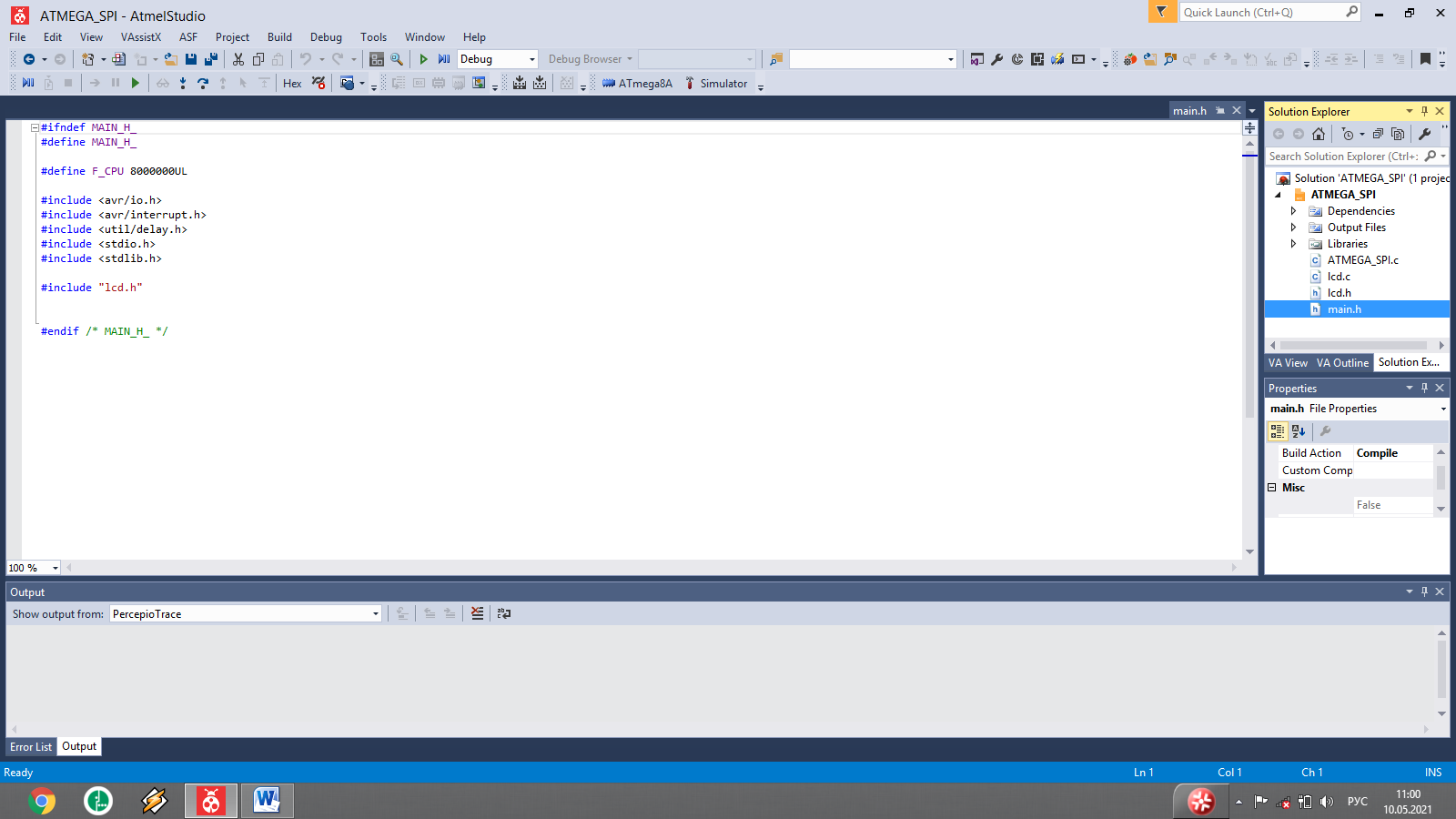
for(n=0;str1[n]!='\0';n++)

sendcharlcd(str1[n]);

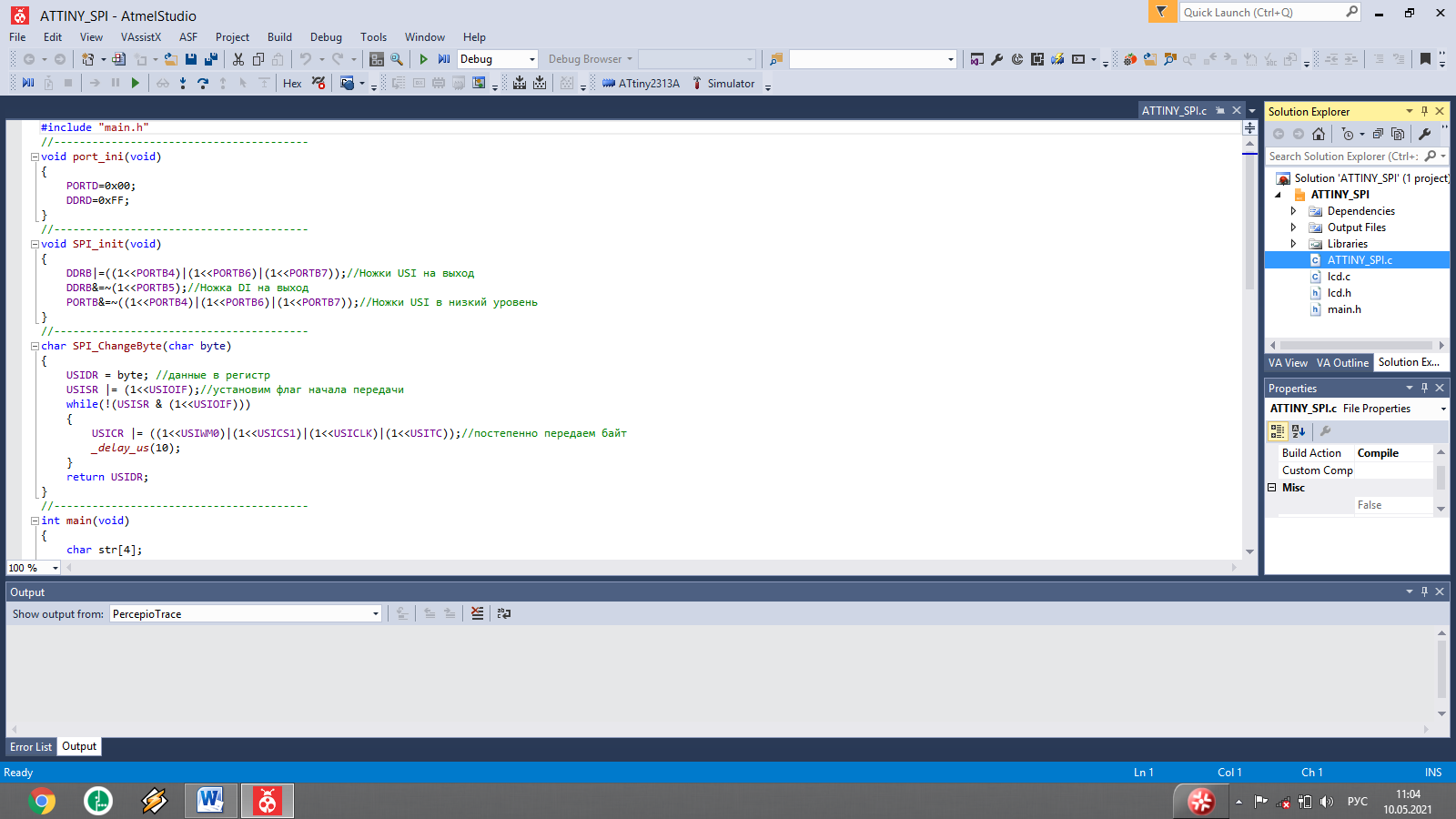
}

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





КОД ДЛЯ ATTINY



#include "main.h"

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

void port\_ini(void)

{

PORTD=0x00;

DDRD=0xFF;

}

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

void SPI\_init(void)

{

DDRB|=((1<<PORTB4)|(1<<PORTB6)|(1<<PORTB7));//Ножки USI на выход

DDRB&=~(1<<PORTB5);//Ножка DI на выход

PORTB&=~((1<<PORTB4)|(1<<PORTB6)|(1<<PORTB7));//Ножки USI в низкий уровень

}

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

char SPI\_ChangeByte(char byte)

{

USIDR = byte; //данные в регистр

USISR |= (1<<USIOIF);//установим флаг начала передачи

while(!(USISR & (1<<USIOIF)))

{

USICR |= ((1<<USIWM0)|(1<<USICS1)|(1<<USICLK)|(1<<USITC));//постепенно передаем байт

*\_delay\_us*(10);

}

return USIDR;

}

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

int main(void)

{

char str[4];

unsigned char n=0,m=0;//переменная для случайного числа

port\_ini(); //Инициализируем порты

LCD\_ini(); //Инициализируем дисплей

SPI\_init(); //Инициализируем шину

clearlcd();

setpos(0,0);

str\_lcd("Hello World!");

*\_delay\_ms*(1000);

setpos(2,1);

str\_lcd("String 2");

*\_delay\_ms*(1000);

while(1)

{

clearlcd();

setpos(0,0);

n= (unsigned char) *rand*() % 256;

//n = 0b10101010;

m = SPI\_ChangeByte(n);

*itoa*(n,str,10);

str\_lcd(str);

*itoa*(m,str,10);

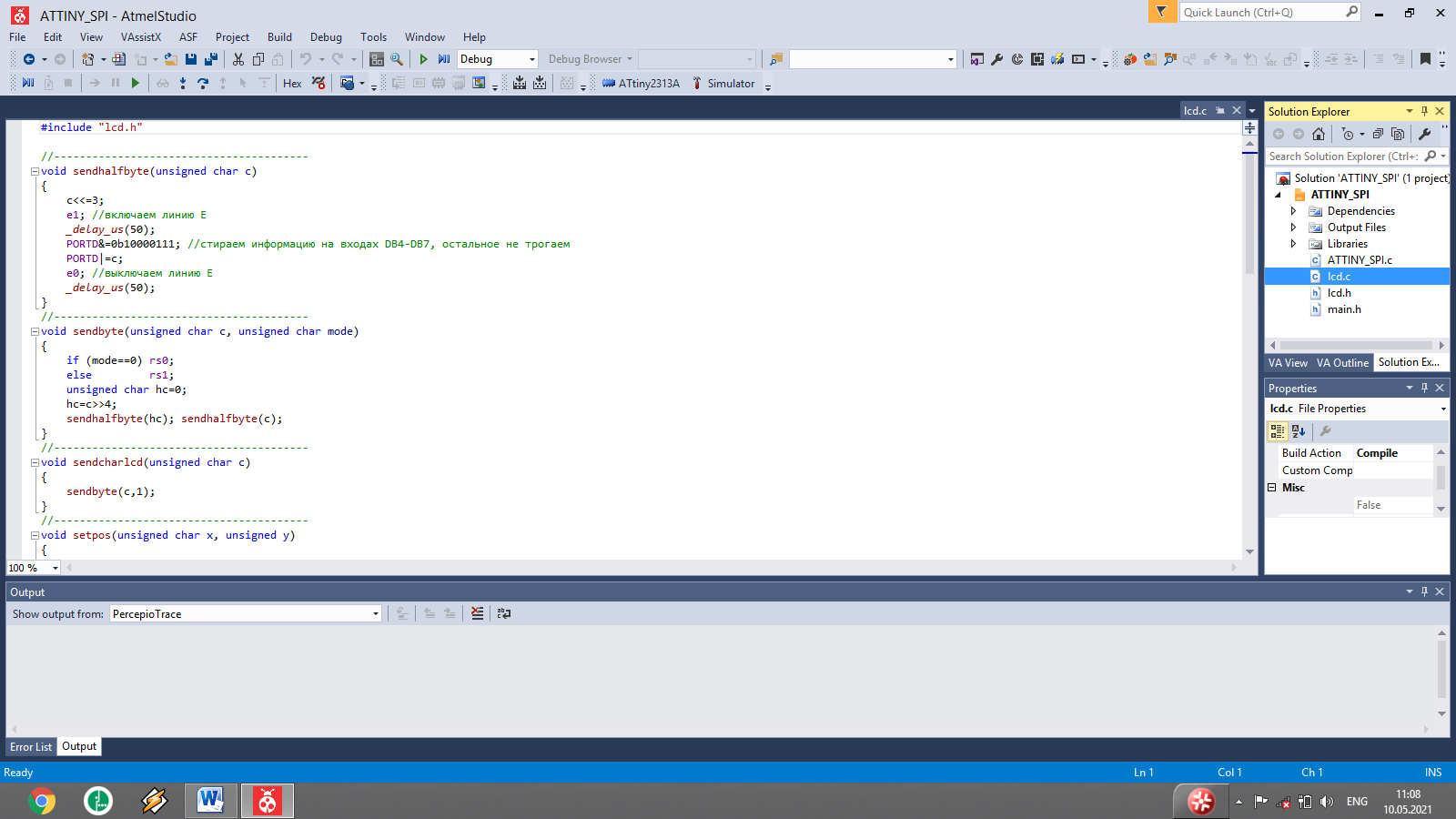
setpos(0,1);

str\_lcd(str);

*\_delay\_ms*(1000);

}

}



#include "lcd.h"

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

void sendhalfbyte(unsigned char c)

{

c<<=3;

e1; //включаем линию Е

*\_delay\_us*(50);

PORTD&=0b10000111; //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)

{

char adress;

adress=(0x40\*y+x)|0b10000000;

sendbyte(adress, 0);

}

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

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]);

}

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

