**SPI. Подключаем сдвиговый регистр 74HC595**

Сегодня мы продолжим работать с шиной **SPI**. Вернее мы продолжим её — эту шину изучать, только занятие теперь будет практическое. Мы возьмём определённое устройсто и попробуем данным устройством поуправлять.

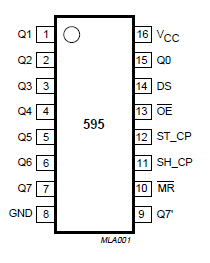
А в качестве устройства мы возьмём простейшую микросхему **74HC595**, которая управляется именно по интерфейсу SPI и представляет собой сдвиговый регистр. Судя по технической документации на данный регистр, разработчиком его является компания Philips.

Обратим внимание на некоторые технические характеристики данного регистра сдвига.

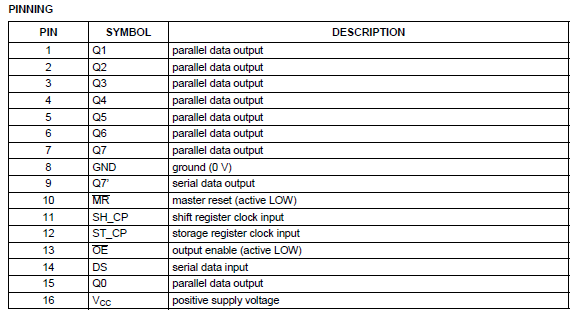
Граничная частота тактирования не должна превышать 100 МГц, так что с нашими 8 или 16 МГц тактировой частоты контроллера мы вряд ли данную частоту превысим.

Напряжение питания микросхемы — от 2 до 6 В.

Данная микросхема существует в двух типах корпусов — обычный DIP, а также DHVQFN16, Мы будем использовать первый вариант, поэтому и посмотрим распиновку по данному варианту



У микросхемы 16 ножек. Посмотрим их назначение в таблице



Немного расшифруем данные ножки

**Q0-Q7** — восемь параллельных выходов общего назначения. Данные выходы нужны для того, чтобы мы могли как-то воспользоваться пришедшими данными по SPI — подключить линейку светодиодов, либо сегменты какого-то индикатора, либо дешифратор и т.д.

**VCC** — напряжение питания.

**GND** — общий провод.

**Q7'** — последовательный выход данных. Тот же самый **MISO**.

**DS** — последовательный вход данных или **MOSI**.

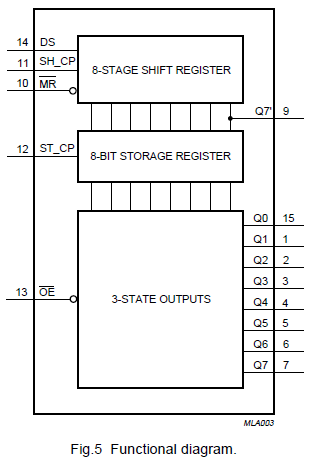
**MR** — это master reset. Будет изучен в процессе использования.

**SH\_CP** — в нашем случае это будет chip select.

**ST\_CP** — это ножка управления регистром хранения, в нашем случае это будет ножка синхронизации, на которую мы будем подавать тактовые импульсы.

**OE** — задействования выхода. При отрицательном значении последовательный выход включен, при положительном — выключен.

Также ещё нам поможет до конца понять назначение ножек поможет вот такая вот блок-схемка



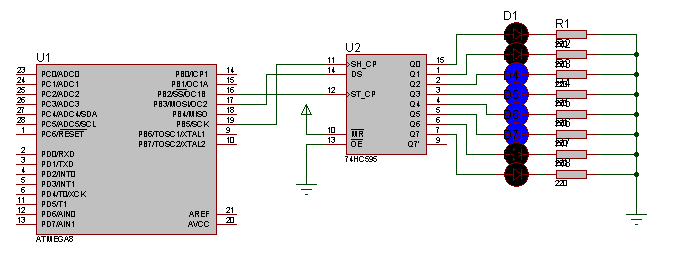
Самое интересное в данной схеме — то, что мы видим здесь не один регистр, а целых два. Один из них — это обычный сдвиговый регистр, в который приходят и из которого уходят наши данные по SPI. Но также есть ещё один регистр, в который эти данные попадают не автоматически, а при определённых условиях. Уже к этому регистру, называемому регистром хранения и привязан жестко параллельный 8-разрядный выход.

Поэтому перед нами теперь встаёт задача. Как же перенести наши данные из верхнего регистра в нижний. А, оказывается, для этого нам необходим квадратный импульс на ножке **12 — ST\_CP**. И отрицательный фронт данного импульса и "попросит" данные спуститься сверху вниз.

Следующая задача — подключить данный сдвиговый регистр **74HC595** к контроллеру ATmega8.

Так как мы не будем пользоваться последовательным выходом из нашей микросхемы, то вывод **MISO** контроллера мы задейстовать не будем.

А к ножкам параллельного выхода мы подключим светодиоды через токоограничивающие резисторы. Получится у нас вот такая схема



Вот так вот всё и подключается. Выход мы включили, хотя он нам не нужен. Позже понадобится. Включается как мы помним он ножкой OE низким уровнем. Выход MR нам не нужен. Он для перезагрузки, поэтому подтянем его к питанию. Ну и соответственно SPI. Здесь также всё несложно. Подключены ножки MISO, SCK и SS. Последнюю можно было подключить с любой ножки МК, так как она аппаратно не управляется.

Теперь нам необходимо как-то написать код и заставить всё это работать.

Создадим проект с именем MYSPI595, создадим и подключим заголовочный файл main.h со следующим содержимым

**#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>**

**#endif /\* MAIN\_H\_ \*/**

Можно также воспользоваться примером, находящимся в технической документации на наш контрооллер в разделе по шине SPI.

Ну что ж, начнём придумывать.

Соответственно, как обычно, нам сначала необходимо будет настроить ножки портов. Все ножки SPI будут у нас настроены на выход, так как MISO мы не используем, а также мы на них установим низкий логический уровень

**#include "main.h"**

**int main(void)**

**{**

**unsigned int i=0;**

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

**PORTB &= ~((1<<PORTB2)|(1<<PORTB3)|(1<<PORTB5)); //низкий уровень**

**while(1)**

**{**

**}**

**}**

Теперь настроим саму шину SPI

PORTB &= ~((1<<PORTB2)|(1<<PORTB3)|(1<<PORTB5)); //низкий уровень

**SPCR = ((1<<SPE)|(1<<MSTR));//Включим шину, объявим ведущим**

**SPDR = 0b00000000;**

Делитель и умножитель мы не включаем, так как частота на микросхеме поддерживается большая. Также мы включили SPI и настроили микроконтроллер ведущим устройством.

Затем занесли в регистр данных нули.

Ну теперь попробуем эти данные передать микросхеме

SPDR = 0b00000000;

**while(!(SPSR & (1<<SPIF)));//подождем пока данные передадутся**

То есть как только мы занесли в регистр данных значение, контроллер начнёт их пытаться передавать. После этого мы в цикле ждём того момента, когда включится бит SPIF в регистре SPSR. Как только он включится, это будет означать, что весь полный байт передался на ведомое устройство.

После этого, чтобы данные в регистре сдвига перешли в нижний регистр хранения, то мы должны сформировать там отрицательный фронт. Но, так как данная ножка SS уже находится в низком логическом уровне, то мы сначала включим ей высокий, а затем низкий, тем самым формируя отрицательный фронт или нисходящий

while(!(SPSR & (1<<SPIF)));//подождем пока данные передадутся

**//сгенерируем отрицательный фронт для записи в STORAGE REGISTER**

**PORTB |= (1<<PORTB2); //высокий уровень**

**PORTB &= ~(1<<PORTB2); //низкий уровень**

Тем самым мы в данном коде сформировали на всех ножках параллельного выхода микросхемы низкий логический уровень и светодиоды у нас должны будут погаснуть, если вдруг они загорелись как-то при старте.

Подождём две секунды и то же самое повторим ещё раз, но со всеми логическими единицами в регистре данных, тем самым все светодиоды, поддключенные к параллельному выходу микросхемы, должны будут засветиться. Затем подождём ещё две секунды

PORTB &= ~(1<<PORTB2); //низкий уровень

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

**SPDR = 0b11111111;**

**while(!(SPSR & (1<<SPIF)));//подождем пока данные передадутся**

**//сгенерируем отрицательный фронт для записи в STORAGE REGISTER**

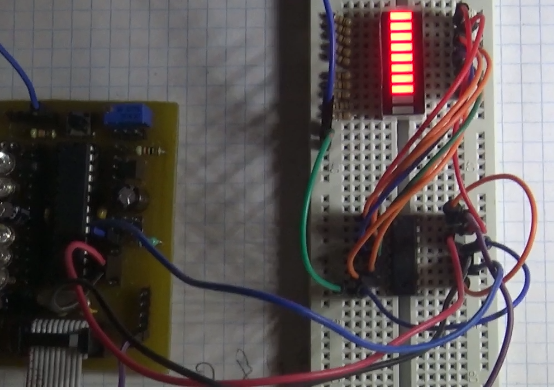
**PORTB |= (1<<PORTB2); //высокий уровень**

**PORTB &= ~(1<<PORTB2); //низкий уровень**

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

while(1)

Посмотрим теперь схему на практике, как у нас всё собрано



Всё подключено также, как мы и видели в протеусе. Вместо 8 светодиодов у нас подключена матрица светодиодов.

Попробуем собрать код и прошить наш контроллер. Сначала светодиоды находятся в потухшем состоянии и через 2 секунды они все начинают светиться. Значит всё у нас передалось.

Ну, и теперь для пущей красоты и для точного убеждения, что байты доходят правильно, не перевёртываются и не искажаются, мы напишем код в бесконечный цикл, который будет отправлять в шину SPI числа от 0 до 255, тем самым у нас на светодиодов будет иммитация инкрементирования двоичных чисел. В main() мы сначала добавим локальную переменную для счёта

int main(void)

{

**unsigned int i=0;**

Напишем теперь тело бесконечного цикла

while(1)

{

**for (i=0;i<256;i++)**

**{**

**SPDR = i;**

**while(!(SPSR & (1<<SPIF)));//подождем пока данные передадутся**

**//сгенерируем отрицательный фронт для записи в STORAGE REGISTER**

**PORTB |= (1<<PORTB2); //высокий уровень**

**PORTB &= ~(1<<PORTB2); //низкий уровень**

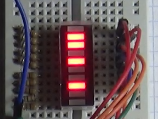
**\_delay\_ms(50);**

**}**

**i=0;**

}

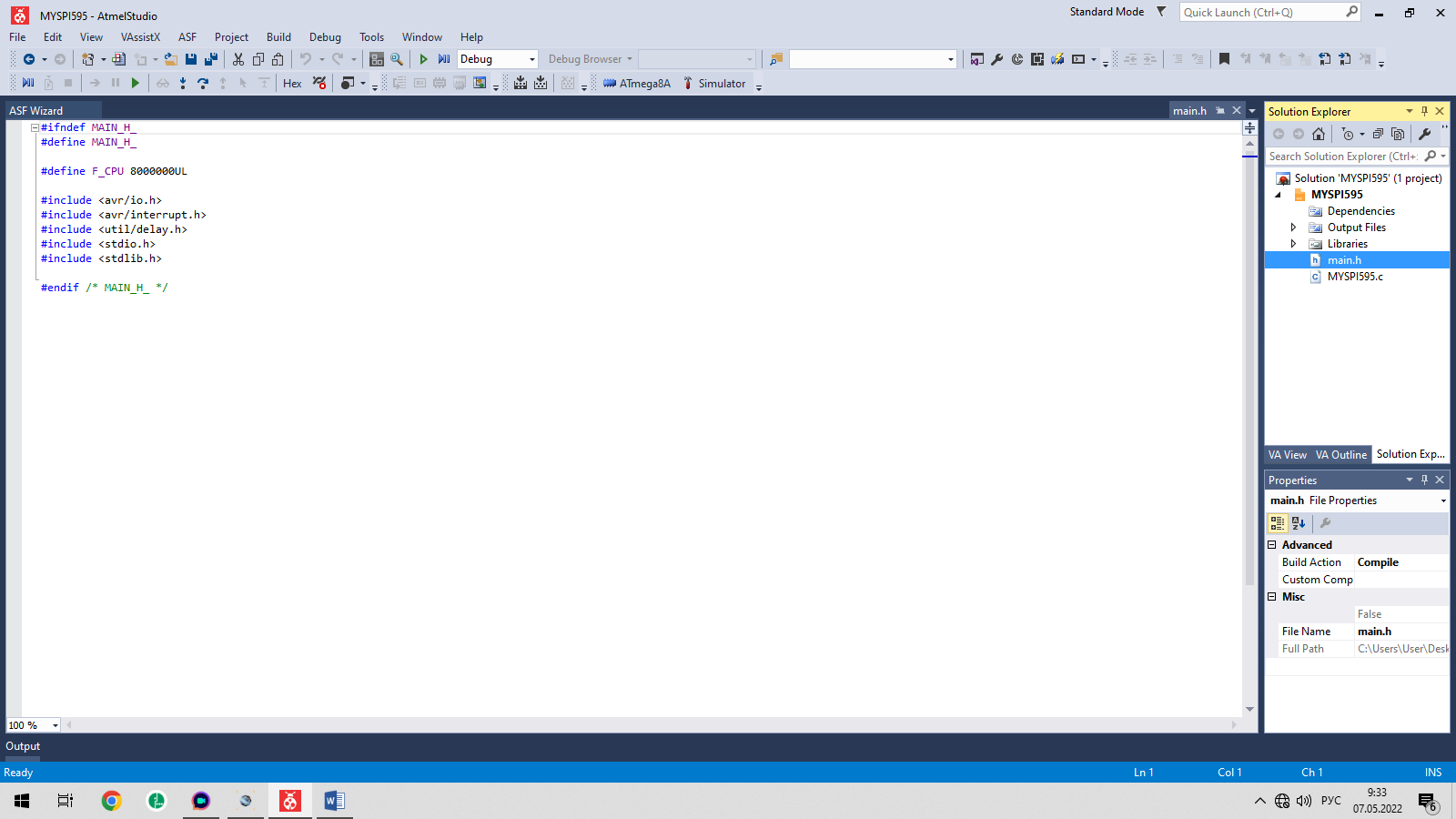
Соберём код, прошьём контроллер и посмотрим результат нашей работы над кодом

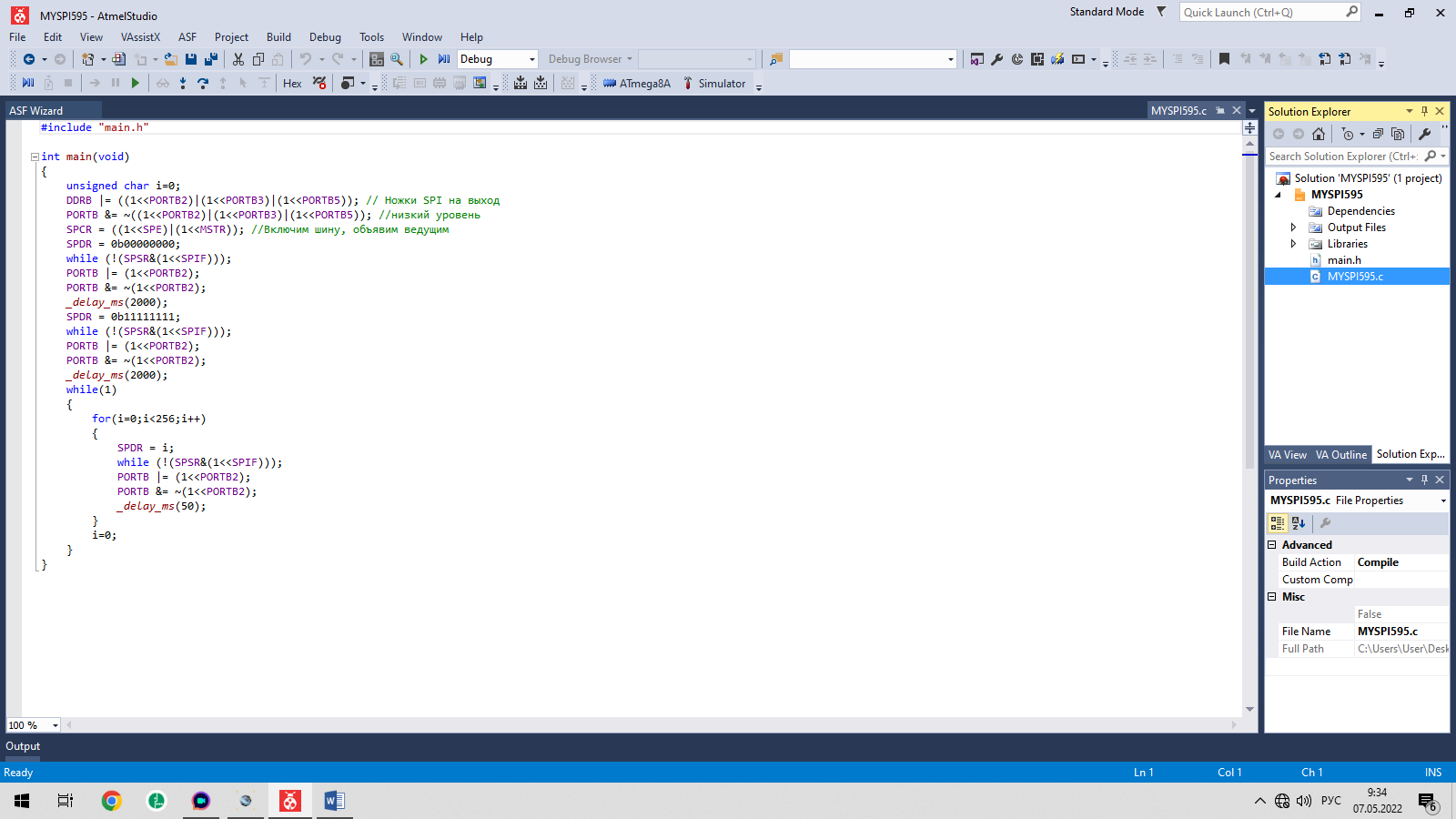


Всё отлично работает.

Таким образом, мы теперь потренировались с шиной SPI на практике.

ПРОГРАММНЫЙ КОД НИЖЕ





Тестировка в протеусе

