

Лабораторная работа №5

Обмен данными по интерфейсу SPI

Цель работы:

- изучение структуры модуля SPI в микроконтроллере AVR,
- программирование передачи и приема данных по SPI.

Теоретическая часть

Общие сведения об интерфейсе

Синхронный интерфейс SPI (Serial Peripheral Interface) используется для организации высокоскоростного канала связи между микроконтроллером и различными периферийными устройствами, а также для обмена данными между микроконтроллерами.

Схема сопряжения двух устройств по SPI показана на рисунке 1. Одно устройство является ведущим (master), другое – ведомым (slave). В общем случае ведомых может быть несколько. Ведущее устройство управляет процессом, вырабатывая синхронизирующий сигнал. Обмен данными происходит с использованием четырех сигнальных линий:

- *SCK* (Serial Clock) – тактовый сигнал,
- *MOSI* (Master Out, Slave In) – линия передачи данных от ведущего к ведомому,
- *MISO* (Master In, Slave Out) – линия передачи данных от ведомого к ведущему,
- \overline{SS} (Slave Select) – выбор ведомого ($\overline{SS}=0$ означает, что ведомое устройство участвует в обмене данными).

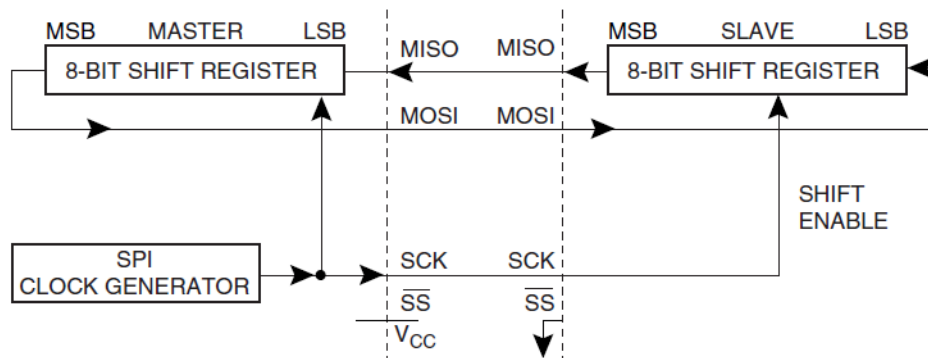


Рисунок 1 – Схема связи двух устройств по SPI

Таким образом, возможна как однонаправленная передача/прием данных, так и полнодуплексный режим.

Модуль SPI в МК ATmega8515

Схема модуля SPI микроконтроллера ATmega8515 представлена на рисунке 2. Жирным шрифтом выделены программно доступные регистры. В микроконтроллерах ATmega8515 для сигналов интерфейса SPI выделены 4 линии порта PB: PB5 – MOSI, PB6 – MISO, PB7 – SCK, PB4 – /SS.

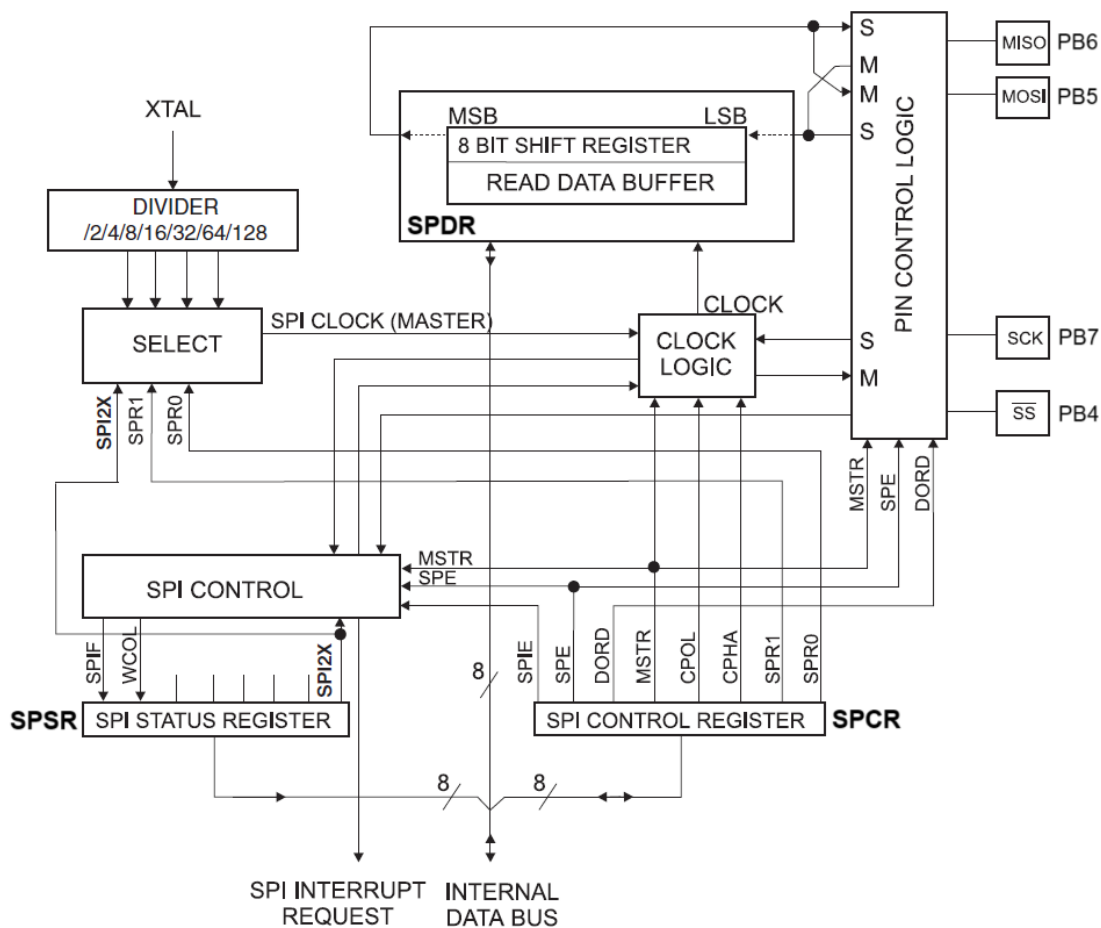


Рисунок 2 – Модуль SPI в ATmega8515

В состав модуля SPI входят:

- 8-разрядный сдвиговый регистр *SPDR*, который принимает байт данных с шины данных микроконтроллера, сдвигает его вправо или влево с выдачей последовательного кода на вывод микроконтроллера, одновременно с выводом принимает последовательный код с входа микроконтроллера и через буферный регистр передает его в шину данных микроконтроллера;
- 8-разрядный регистр управления *SPCR*;
- 8-разрядный регистр состояния *SPSR*;
- схемы управления, в том числе управление тактированием с делителем частоты.

На рисунках 3–4 показана структура регистров *SPCR* и *SPSR*, а в таблицах 1-2 описано назначение битов.

Bit	7	6	5	4	3	2	1	0
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Рисунок 3 – Структура регистра SPCR

Bit	7	6	5	4	3	2	1	0
	SPIF	WCOL	–	–	–	–	–	SPI2X
Read/Write	R	R	R	R	R	R	R	R/W
Initial Value	0	0	0	0	0	0	0	0

Рисунок 4 – Структура регистра SPSR

Таблица 1 – Назначение битов регистра SPCR

Биты	Назначение
SPE	SPI Enable. Разрешение работы SPI
MSTR	Master/Slave Select. Работа в роли ведущего (MSTR=1) или ведомого (MSTR=0)
SPR1:0	SPI Clock Rate. Задаёт частоту сигнала SCK путем выбора предделителя тактовой частоты МК, см. таблицу 3. При работе в роли ведомого эти биты не используются
DORD	Data Order. При DORD=1 первым передается младший бит. При DORD=0 то первым передается старший бит
CPOL	Clock Polarity. Полярность синхросигнала SCK
CPHA	Clock Phase. Фаза синхросигнала SCK
SPIE	SPI Interrupt Enable. Разрешение прерывания по флагу SPIF=1

Таблица 2 – Назначение битов регистра SPSR

Биты	Назначение
SPIF	SPI Interrupt Flag. Флаг завершения обмена
WCOL	Write Collision Flag. Флаг повторной записи. WCOL=1 при попытке записи в SPDR во время передачи очередного байта
SPI2X	Double SPI Speed Bit. Управляющий бит. Увеличение скорости обмена в 2 раза, бит работает в комбинации с SPR1, SPR0

При $MSTR=1$ микроконтроллер работает в роли ведущего. При этом линия *MOSI* является выходом данных, линия *MISO* – входом данных, линия *SCK* – выходом для тактовых импульсов, используемых в качестве сдвиговых при приеме данных ведомым микроконтроллером. Функция линии *PB4* ($/SS$) зависит от состояния разряда *DDRB.4*. При $DDRB.4=1$ (на вывод) линия *PB4* не подключена к SPI и используется как обычная линия вывода. При $DDRB.4=0$ (на ввод) если на линии *PB4* появляется 0, модуль SPI переключается в роль ведомого, бит *MSTR* сбрасывается в 0. Поэтому если изменение роли не планируется, линия *PB4* ведущего МК должна быть сконфигурирована на вывод.

При $MSTR=0$ микроконтроллер работает в роли ведомого. При этом *MOSI* является входом данных, *MISO* – выходом данных, *SCK* – входом для импульсов сдвига, линия $/SS$ – входом. Перевод ведомого в рабочее состояние происходит по сигналу $/SS=0$.

Скорость передачи данных определяется на стороне ведущего битами *SPR1*, *SPR0*, *SPI2X*, задающими значение предделителя тактовой частоты МК согласно таблице 3.

Таблица 3 – Настройка частоты синхросигнала SCK

SPI2X	SPR1	SPR0	Предделитель
0	0	0	4
0	0	1	16
0	1	0	63
0	1	1	128
1	0	0	2
1	0	1	8
1	1	0	32
1	1	1	64

Влияние битов *CPOL*, *CPHA* на синхросигнал *SCK* иллюстрируют временные диаграммы на рисунках 5-6. Комбинация этих настроек определяет, по какому перепаду *SCK* будет происходить сдвиг посылки, а по какому – чтение бита с линии передачи.

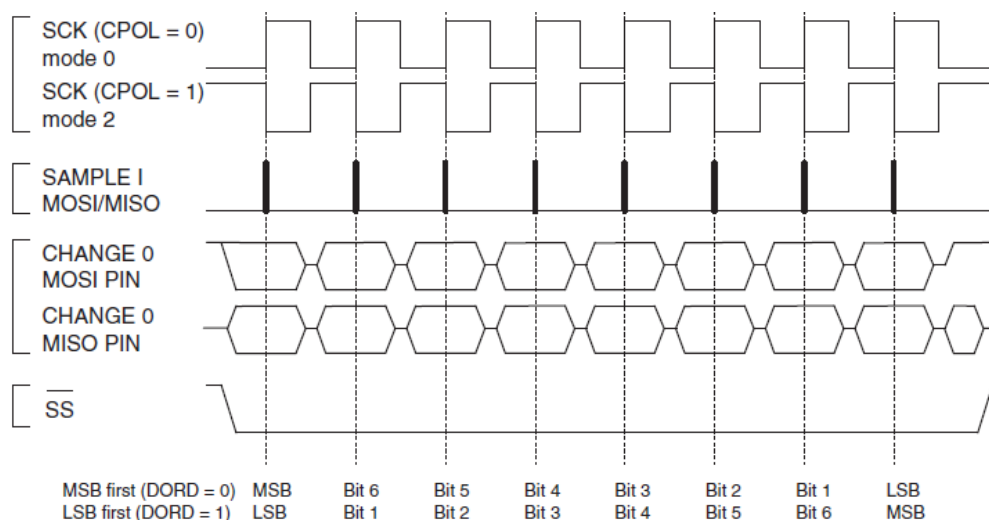


Рисунок 5 – Формат обмена при CPHA=0

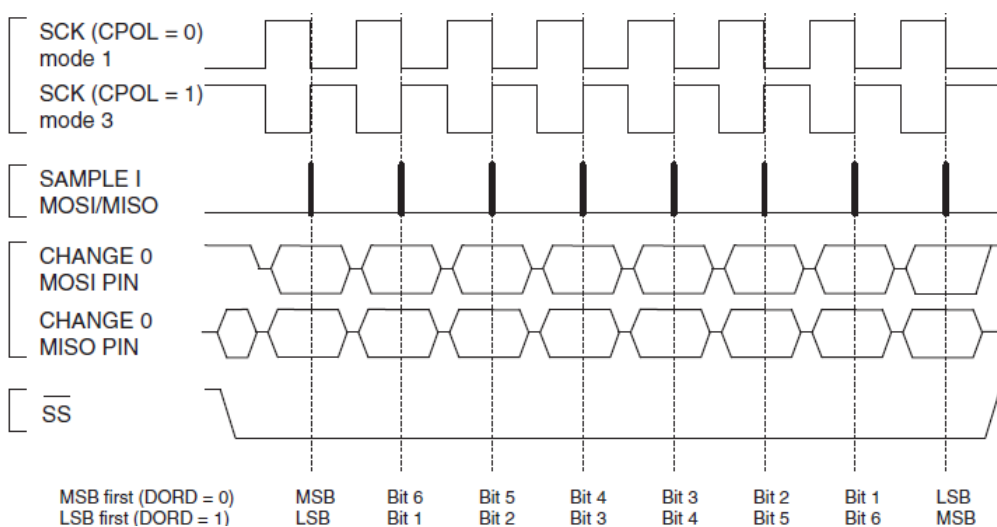



Рисунок 6 – Формат обмена при CPHA=1

Передача данных начинается после записи данных в регистр *SPRD* ведущего микроконтроллера при условии $SPE=1$.

Следует отметить, что SPI в МК AVR используется не только для связи с периферией, но и для внутрисхемного программирования (ISP – in-system programming), то есть загрузки программы во Flash-память и данных в EEPROM. В этом случае ведущим выступает программатор, а ведомым – программируемый МК.

Практическая часть

 Отчет по этой лабораторной работе оформляется один общий на бригаду из двух студентов.

В данной лабораторной работе предлагается исследовать работу SPI на примере связи двух микроконтроллеров в симплексном и дуплексном режимах. Одно устройство отправляет другому заранее заложенное в оперативную память сообщение.

Задание 1. Передача данных в симплексном режиме в Proteus

На рисунке 7 представлена схема соединения МК для передачи сообщения от ведущего МК ведомому. Передача начинается по нажатию кнопки *START* ведущего МК. После завершения приема зажигаются светодиоды, подключенные к ведомому МК. Последовательный вывод принятых байтов на светодиоды выполняется по нажатию кнопки *SHOW* ведомого МК. Для дополнительной проверки в схему добавлен SPI Debugger.

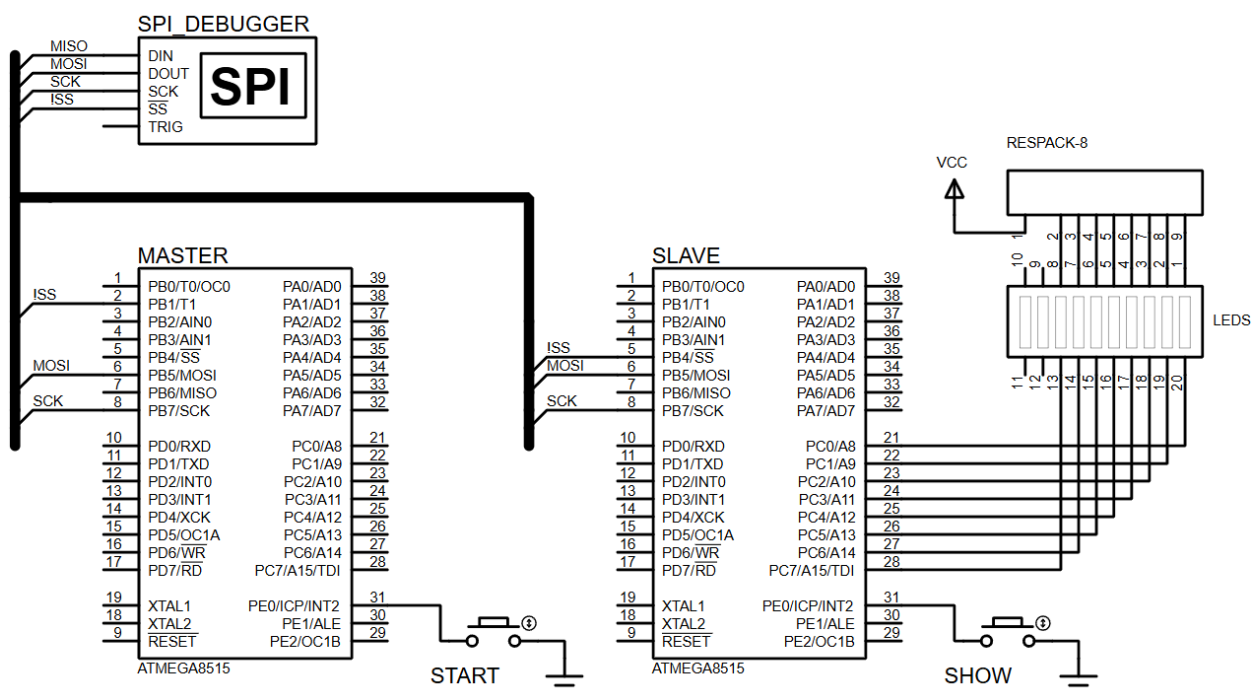






Рисунок 7 – Схема соединения микроконтроллеров (симплекс)

Соберите схему в среде Proteus. SPI Debugger можно найти на панели инструментов слева в меню  *Virtual Instruments Mode*. Чтобы изобразить шину, используйте пункт  *Buses Mode* на панели инструментов слева. Каждый подсоединенный к шине провод должен иметь имя, его можно задать, нажав на провод правой кнопкой мыши и выбрав пункт *Place Wire Label*.

 Подробности работы с шинами можно посмотреть в [видеоролике](#).

Установите частоту работы микроконтроллеров 3,69 МГц. Создайте проекты программ для обоих МК в Proteus (нажатие правой кнопкой мыши на МК – *Edit Source Code*), выбрав при создании проектов компилятор WinAVR. Вставьте приведенный ниже исходный код двух программ, соберите проекты кнопкой  *Build Project* на панели инструментов.

Программа 1 (ведущий)

```
#include <avr/io.h>

/* Кнопка START - PE0 */
#define BUTTON_START 0

/* Выводы SPI - PB */
#define PIN_SS 4
#define PIN_SS1 1
#define PIN_MOSI 5
#define PIN_SCK 7

/* Передаваемые данные */
#define DATA_LENGTH 3
const unsigned char data[DATA_LENGTH] = {'M', 'P', 'S'};

int main() {
    /* Инициализация SPI */
    /* Настройка выводов MOSI, SCK, SS на вывод */
    DDRB = (1<<PIN_MOSI)|(1<<PIN_SCK)|(1<<PIN_SS);
    /* Включение SPI в режиме ведущего, частота передачи f_clk/16 */
    SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);

    /* Инициализация портов ввода-вывода */
    /* Настройка PIN_SS1 на вывод */
    DDRB |= (1<<PIN_SS1);
    PORTB |= (1<<PIN_SS1);
    /* Настройка PE0 на ввод с подтягивающим резистором */
    PORTE = (1<<BUTTON_START);

    /* Бесконечный цикл */
    while(1) {
        /* Проверка нажатия кнопки */
        if (!(PINE & (1<<BUTTON_START))) {
            /* Ожидание отпускания кнопки */
            while (!(PINE & (1<<BUTTON_START)))
                ;

            /* Цикл передачи данных */
            for (uint8_t i = 0; i < DATA_LENGTH; i++) {
                /* Выбор ведомого */
```

```

        PORTB &= ~(1<<PIN_SS1);

        /* Отправка i-го байта */
        SPDR = data[i];
        /* Ожидание освобождения буфера */
        while( !(SPSR & (1<<SPIF)) )
            ;

        /* Завершение обмена с ведомым */
        PORTB |= (1<<PIN_SS1);
    }
}

return 0;
}

```

Программа 2 (ведомый)

```

#include <avr/io.h>
#include <avr/interrupt.h>

/* Кнопка SHOW - PE0 */
#define BUTTON_SHOW 0

/* Принимаемые данные */
#define DATA_LENGTH 3
unsigned char data[DATA_LENGTH] = { 0 };
uint8_t receivedBytes = 0;

/* Обработчик прерывания SPI_STC */
ISR(SPI_STC_vect) {
    if (receivedBytes < DATA_LENGTH) {
        data[receivedBytes++] = SPDR;
    }
    /* Включить светодиоды по завершении приема */
    if (receivedBytes == DATA_LENGTH) {
        PORTC = 0x00;
    }
}

int main() {
    /* Инициализация SPI */
    /* Включение SPI в режиме ведомого */
    SPCR = (1<<SPE) | (1<<SPIE);

    /* Инициализация портов ввода-вывода */
    /* Настройка PE0 на ввод с подтягивающим резистором */
    PORTE = (1<<BUTTON_SHOW);
    /* Настройка PC на вывод */
    DDRC = 0xFF;
    /* Погасить светодиоды, подключенные к PC */
    PORTC = 0xFF;

    /* Глобальное разрешение прерываний */
    sei();

    /* Вывод полученных данных на светодиоды, байт за байтом */
    uint8_t i = 0; /* Счетчик байтов */
    /* Бесконечный цикл */
    while(1) {
        /* Проверка нажатия кнопки */
        if (!(PINE & (1<<BUTTON_SHOW))) {
            /* Ожидание отпускания кнопки */

```

```

        while (!(PINE & (1<<BUTTON_SHOW)))
            ;

        /* Вывод данных с инверсией битов для светодиодов */
        PORTC = ~data[i];
        i = (i + 1) % DATA_LENGTH;
    }

    return 0;
}

```

Выполните пошаговую отладку программ в Proteus. С помощью окон *Variables* и *Data memory* отследите запись полученных данных в оперативную память ведомого МК. Просмотрите полученные байты на светодиодах. Просмотрите передаваемые байты в SPI Debugger, предварительно указав в нем формат обмена, совпадающий с настройками в программах.

Средствами Proteus постройте временную диаграмму сигналов *SS*, *SCK*, *MOSI*. Расшифруйте диаграмму, подписав на ней передаваемые биты и байты. Измерьте длительность бита, сравните скорость передачи с запрограммированной.


Работающую схему, полученные байты в оперативной памяти ведомого, расшифрованную временную диаграмму покажите преподавателю.

Скриншот схемы в Proteus, полученных байтов в оперативной памяти ведомого, расшифрованную временную диаграмму поместите в отчет.

Задание 2. Обработка прерывания SPI

Замените в программе ведущего МК программный опрос флага *SPIF* на обработку прерывания по завершении передачи. Программу ведомого оставьте без изменений. Протестируйте измененную программу в Proteus. Результат покажите преподавателю.

Задание 3. Передача данных в симплексном режиме на макете

 Для выполнения этого задания потребуется две платы STK500. Объединитесь с другой бригадой и покажите задание совместно.

Загрузите в МК на одной плате STK500 программу ведущего, а в МК на другой плате – программу ведомого. Выполните соединения при отключенном питании согласно рисунку 7: соедините выводы *PB7* (*SCK*) обеих плат, выводы *PB5* (*MOSI*) обеих плат, вывод *PB1* ведущего с выводом *PB4* (*/SS*) ведомого, также соедините выводы *GND* обеих плат. Подключите кнопки и светодиоды. Включите питание плат и проверьте работу программ. Результат покажите преподавателю. Фотографию макета с принятым байтом на светодиодах поместите в отчет.

Задание 4. Передача данных в дуплексном режиме в Proteus

Для связи МК по SPI в дуплексном режиме соберите показанную на рисунке 8 схему в среде Proteus. Используйте приведенный ниже исходный код двух программ 3 и 4.

Дуплекс предполагает одновременную передачу данных в двух направлениях. Обмен начинается по нажатию кнопки *START* ведущего МК. Чтобы уведомить ведомый МК о необходимости выдать первый байт сообщения в *SPDR*, ведущий посылает заранее определенный байт *0xAA* ведомому, после чего оба устройства побайтово передают друг другу сообщения. После завершения обмена зажигаются светодиоды ведомого. Последовательно вывести принятые байты на светодиоды можно нажатием кнопок *SHOW* ведущего и ведомого МК.

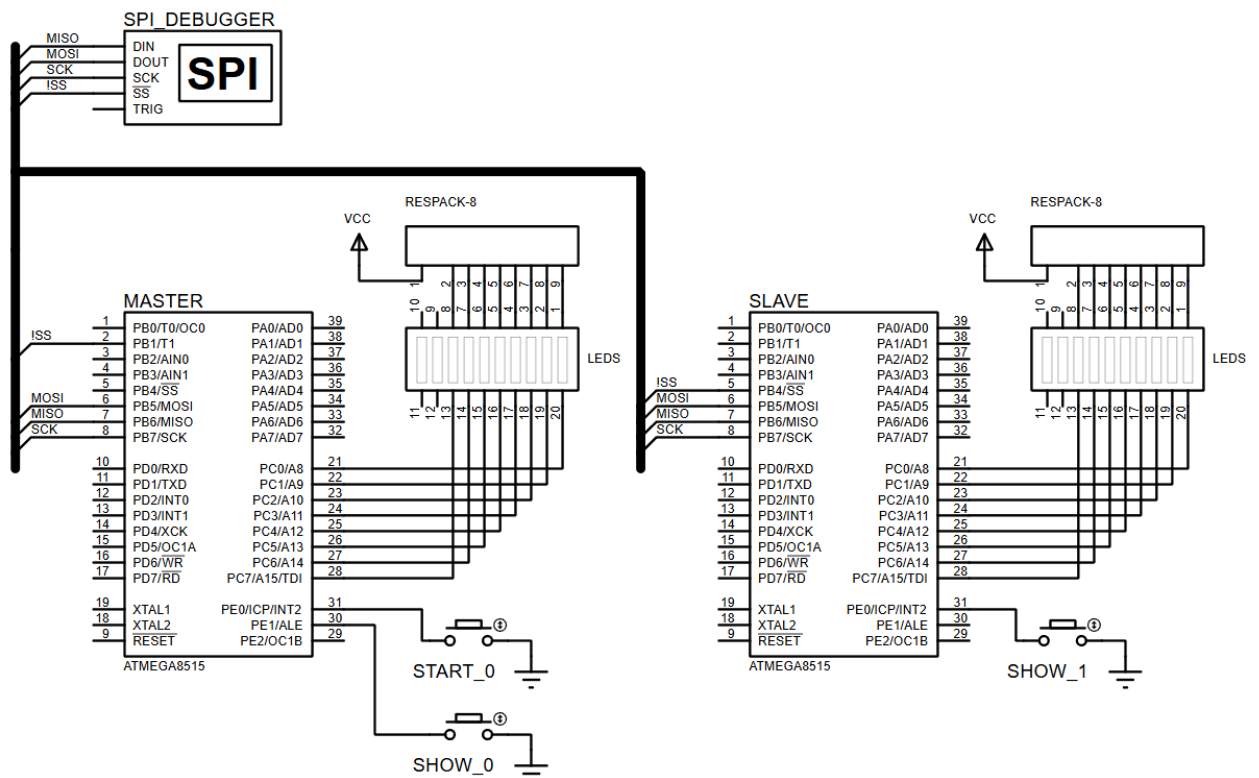


Рисунок 8 – Схема соединения микроконтроллеров (дуплекс)

Программа 3 (ведущий)

```
#include <avr/io.h>

#define F_CPU 3690000
#include <util/delay.h>

/* Кнопки управления - PE */
#define BUTTON_START 0
#define BUTTON_SHOW 1

/* Выводы SPI - PB */
#define PIN_SS 4
#define PIN_SS1 1
#define PIN_MOSI 5
#define PIN_SCK 7
```

```

/* Массив данных */
#define DATA_LENGTH 6
unsigned char data[DATA_LENGTH] = {0x41, 0x54, 0x6D, 0x65, 0x67, 0x61};
uint8_t currentByte = 0; /* Выводимый байт */

/* Байт для подачи сигнала о начале обмена */
#define START_SYM 0xAA

char SpiSendReceive(char c) {
    /* Выбор ведомого */
    PORTB &= ~(1<<PIN_SS1);
    /* Отправка байта */
    SPDR = c;
    /* Ожидание освобождения буфера */
    while( !(SPSR & (1<<SPIF)) )
        ;
    /* Завершение обмена с ведомым */
    PORTB |= (1<<PIN_SS1);
    /* Чтение байта */
    return SPDR;
}

int main() {
    /* Инициализация SPI */
    /* Настройка выводов MOSI, SCK, SS на вывод */
    DDRB = (1<<PIN_MOSI)|(1<<PIN_SCK)|(1<<PIN_SS);
    /* Включение SPI в режиме ведущего, частота передачи f_clk/16 */
    SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0)|(1<<CPHA);

    /* Инициализация портов ввода-вывода */
    /* Настройка PIN_SS1 на вывод */
    DDRB |= (1<<PIN_SS1);
    PORTB |= (1<<PIN_SS1);
    /* Настройка PE0, PE1 на ввод с подтягивающим резистором */
    PORTE = (1<<BUTTON_START)|(1<<BUTTON_SHOW);
    /* Настройка PC на вывод */
    DDRC = 0xFF;
    /* Погасить светодиоды, подключенные к PC */
    PORTC = 0xFF;

    /* Бесконечный цикл */
    while(1) {
        /* Проверка нажатия кнопки START */
        if (!(PINE & (1<<BUTTON_START))) {
            /* Ожидание отпускания кнопки */
            while (!(PINE & (1<<BUTTON_START)))
                ;
            /* Отправка сигнала о начале обмена */
            SpiSendReceive(START_SYM);
            /* Задержка 10 мкс */
            _delay_us(10);

            /* Цикл передачи данных */
            for (uint8_t i = 0; i < DATA_LENGTH; i++) {
                /* Отправка и прием i-го байта */
                data[i] = SpiSendReceive(data[i]);
                /* Задержка 10 мкс */
                _delay_us(10);
            }
        }

        /* Проверка нажатия кнопки SHOW */
        if (!(PINE & (1<<BUTTON_SHOW))) {

```

```

        /* Ожидание отпускания кнопки */
        while (!(PINE & (1<<BUTTON_SHOW)))
            ;

        /* Вывод данных с инверсией битов для светодиодов */
        PORTC = ~data[currentByte];
        currentByte = (currentByte + 1) % DATA_LENGTH;
    }

    return 0;
}

```

Программа 4 (ведомый)

```

#include <avr/io.h>
#include <avr/interrupt.h>

/* Кнопка SHOW - PE0 */
#define BUTTON_SHOW 0

/* SPI MISO - PB6 */
#define PIN_MISO 6

/* Массив данных */
#define DATA_LENGTH 6
unsigned char data[DATA_LENGTH] = {0x53, 0x54, 0x4B, 0x35, 0x30, 0x30};
uint8_t receivedBytes = 0; /* Счетчик принятых байтов */
uint8_t currentByte = 0; /* Выводимый байт */

/* Байт для подачи сигнала о начале обмена */
#define START_SYM 0xAA

/* Обработчик прерывания SPI_STC */
ISR(SPI_STC_vect) {
    unsigned char incomingByte = SPDR;
    if (incomingByte == START_SYM) {
        /* Загрузка 0-го байта для дальнейшей отправки */
        SPDR = data[0];
    }
    else if (receivedBytes < DATA_LENGTH) {
        SPDR = data[receivedBytes + 1];
        data[receivedBytes] = incomingByte;
        receivedBytes++;
        /* Включить светодиоды по завершении приема */
        if (receivedBytes == DATA_LENGTH) {
            PORTC = 0x00;
        }
    }
}

int main() {
    /* Инициализация SPI */
    /* Включение SPI в режиме ведомого */
    SPCR = (1<<SPE) | (1<<SPIE) | (1<<CPHA);

    /* Инициализация портов ввода-вывода */
    /* Настройка MISO на вывод */
    DDRB = (1<<PIN_MISO);
    /* Настройка PE0 на ввод с подтягивающим резистором */
    PORTE = (1<<BUTTON_SHOW);
    /* Настройка PC на вывод */
    DDRC = 0xFF;
    /* Погасить светодиоды, подключенные к PC */
}

```

```

PORTC = 0xFF;

/* Глобальное разрешение прерываний */
sei();

/* Вывод полученных данных на светодиоды, байт за байтом */
/* Бесконечный цикл */
while(1) {
    /* Проверка нажатия кнопки */
    if (!(PINE & (1<<BUTTON_SHOW))) {
        /* Ожидание отпускания кнопки */
        while (!(PINE & (1<<BUTTON_SHOW)))
            ;

        /* Вывод данных с инверсией битов для светодиодов */
        PORTC = ~data[currentByte];
        currentByte = (currentByte + 1) % DATA_LENGTH;
    }
}

return 0;
}

```

i В программе ведущего объявлен макрос `F_CPU`, равный тактовой частоте МК. Эта константа нужна, чтобы функции библиотеки [delay.h](#) вычисляли параметры циклов задержки. Можно задать `F_CPU` в исходном коде или в настройках проекта программы. В AVR Studio: меню Project – Configuration Options – «Frequency». В Proteus: меню Project – Project Settings – значение «Clock for delays» с вкладки Controller передается в параметр компилятора на вкладке Options.

Выполните пошаговую отладку программ в Proteus. С помощью окон *Variables* и *Data memory* отследите запись полученных данных в оперативную память обоих МК. Просмотрите полученные байты на светодиодах. Просмотрите передаваемые байты в SPI Debugger.

Средствами Proteus постройте временную диаграмму сигналов *SS*, *SCK*, *MOSI*, *MISO*. Расшифруйте диаграмму, подписав на ней передаваемые байты.

Работающую схему, полученные байты в оперативной памяти, расшифрованную временную диаграмму покажите преподавателю.

Скриншот схемы в Proteus, полученных байтов в оперативной памяти, расшифрованную временную диаграмму поместите в отчет.

Задание 5. Передача данных в дуплексном режиме на макете

i Для выполнения этого задания потребуется две платы STK500. Объединитесь с другой бригадой и покажите задание совместно.

Загрузите в МК на одной плате STK500 программу 3 ведущего, а в МК на другой плате – программу 4 ведомого. Выполните соединения при отключенном питании согласно рисунку 8: соедините выводы *PB7 (SCK)* обеих плат, выводы *PB5 (MOSI)* обеих

плат, выводы *PB6 (MISO)* обеих плат, вывод *PB1* ведущего с выводом *PB4 (/SS)* ведомого, также соедините выводы *GND* обеих плат. Подключите кнопки и светодиоды. Включите питание плат и проверьте работу программ. Результат покажите преподавателю. Фотографию макета с принятыми байтами на светодиодах обеих плат поместите в отчет.

Задание 6. Передача произвольного сообщения

Измените программы 3-4 для передачи в обе стороны самостоятельно придуманных сообщений длиной 4-6 символов. Установите формат передачи байтов: начиная с младшего бита.

Выполните проверку программ в Proteus, постройте и расшифруйте временную диаграмму сигналов. Результаты покажите преподавателю, временную диаграмму и измененный исходный код программ поместите в отчет.

Оформление отчета

Отчет должен содержать следующие элементы:

- цель работы и выводы;
- две схемы, собранные в среде Proteus, с открытым окном SPI Debugger;
- скриншот содержимого памяти ведомого с выделенными байтами, которые были получены по SPI (задание 1);
- расшифрованную временную диаграмму симплексной передачи (задание 1);
- измененный исходный код программы ведущего для симплексной передачи (задания 1-2);
- фотографию макета с принятым байтом на светодиодах (задание 3);
- скриншот содержимого памяти обоих МК с выделенными байтами, которые были получены по SPI (задание 4);
- расшифрованную временную диаграмму дуплексной передачи (задание 4);
- фотографию макета с принятыми байтами на светодиодах (задание 5);
- измененный исходный код программ ведущего и ведомого для дуплексной передачи (задание 6);
- расшифрованную временную диаграмму для задания 6.

Контрольные вопросы и задания

1. Как осуществляется синхронный обмен данными по SPI? Опишите роль каждой сигнальной линии.

2. Как соединить два устройства для обмена данными по SPI?
3. В чем заключается главенствующая роль ведущего устройства при обмене данными по SPI?
4. Какими способами можно подключить несколько ведомых устройств к одному ведущему? Нарисуйте схемы подключения и опишите логику работы программ.
5. Как настраивается скорость передачи данных по SPI?
6. Зачем настраивать линию PB4 (/SS) на вывод в ведущем микроконтроллере?
7. Можно ли в программе 1 использовать для выбора ведомого линию PB4 вместо PB1? Если нельзя, обоснуйте. Если можно, измените программу и продемонстрируйте ее работу.
8. Какую роль играет константа F_CPU в программе 3?
9. Что произойдет, если в программе 3 убрать вызовы подпрограммы задержки? Смоделируйте это в Proteus, оцените причины и последствия.
10. Измените значения управляющего бита DORD в программах 1-2. Снимите временную диаграмму, оцените отличие от исходной диаграммы.
11. Измените значения управляющих битов CPOL, CPHA в программах 1-2. Снимите временную диаграмму, оцените отличие от исходной диаграммы.
12. Измените программы 1-2 для передачи сообщения от ведомого ведущему.
13. Измените программы 1-2 для передачи сообщения произвольной длины (ведомое устройство не знает длину сообщения).

Справочная литература

1. [AVR-LibC](#) – справочник по стандартной библиотеке C для AVR
2. [AVR Software User Guide](#) – подсказки по улучшению C-кода под AVR
3. [Документация на микроконтроллер ATmega8515](#)
4. [Руководство пользователя отладочной платой STK500](#)