МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Белорусский национальный технический университет

Кафедра «Робототехнические системы»

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

Лабораторный практикум

Часть 2

Минск
БНТУ
2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Белорусский национальный технический университет

Кафедра «Робототехнические системы»

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

Лабораторный практикум
для студентов специальностей 1-53 01 01
«Автоматизация технологических процессов и производств»
и 1-53 01 06 «Промышленные роботы
и робототехнические комплексы»

В 2 частях

Часть 2

Минск
БНТУ
2016

УДК 004.31-181.48(075.8)
ББК 32.97я7
П78

Составители части:
Ф. Л. Сиротин, Е. Р. Новичихина

Рецензенты:
доцент БГТУ *А. С. Кравченко*;
доцент БГУИР *В. Н. Урядов*

П78 **Программирование** микроконтроллеров: лабораторный практикум для студентов специальностей 1-53 01 01 «Автоматизация технологических процессов и производств» и 1-53 01 06 «Промышленные роботы и робототехнические комплексы» : в 2 ч. Ч. 2 / сост.: Ф. Л. Сиротин, Е. Р. Новичихина. – Минск: БНТУ, 2016. – 33 с.
ISBN 978-985-550-352-2 (Ч. 2).

Приведены методические указания к лабораторным работам в, которых рассматриваются различные варианты стандартных последовательных интерфейсов микроконтроллеров. Лабораторные работы выполняются на микроконтроллерах Atmega в режиме моделирования.

Часть 1 «Программирование микроконтроллеров» вышла в 2013 г. (сост. Ф. Л. Сиротин, А. М. Капустина, Ю. А. Агейчик).

УДК 004.31-181.48(075.8)
ББК 32.97я7

ISBN 978-985-550-352-2 (Ч. 2)
ISBN 978-985-550-262-4

ОБМЕН ДАННЫМИ МЕЖДУ МИКРОКОНТРОЛЛЕРАМИ

Лабораторная работа № 5

ИСПОЛЬЗОВАНИЕ USART ДЛЯ ОБМЕНА ДАННЫМИ МЕЖДУ МИКРОКОНТРОЛЛЕРАМИ

Цель работы

изучить аппаратный способ передачи данных по средствам протокола *USART*; научится использовать протокол *USART* для обмена данными между устройствами.

Общие сведения

Универсальный синхронный и асинхронный последовательный приемопередатчик (*USART*) предназначен для организации гибкой последовательной связи.

Отличительные особенности:

– полнодуплексная работа (раздельные регистры последовательного приема и передачи);

– асинхронная или синхронная работа;

– ведущее или подчиненное тактирование связи в синхронном режиме работы;

– высокая разрешающая способность генератора скорости связи

– поддержка формата передаваемых данных с 5, 6, 7, 8 или 9 битами данных и 1 или 2 стоп-битами;

– аппаратная генерация и проверка бита паритета (четность/нечетность);

– определение переполнения данных;

– определение ошибки в структуре посылки;

– фильтрация шума с детекцией ложного старт-бита и цифровым ФНЧ;

– три раздельных прерывания по завершении передачи, освобождении регистра передаваемых данных и завершении приема;

– режим многопроцессорной связи;

– режим удвоения скорости связи в асинхронном режиме.

Последовательная посылка состоит из бит данных, бит синхронизации (старт и стоп-биты), а также опционального бита паритета для поиска ошибок. *USART* поддерживает все 30 комбинаций следующих форматов посылок:

1 старт-бит

5, 6, 7, 8 или 9 бит данных

без паритета, с битом четности, с битом нечетности

1 или 2 стоп-бита

Посылка начинается с старт-бита, далее следует передача бит данных (с самого младшего разряда). Затем передача остальных бит данных (макс. число бит данных 9), которая заканчивается старшим разрядом данных. Если разрешена функция контроля паритета, то сразу после бит данных передается бит паритета, а затем стоп-биты. После завершения передачи посылки имеется возможность либо передавать следующую посылку, либо перевести линию связи в состояние ожидания (высокий уровень). Рис. 5.1 иллюстрирует возможность сочетания форматов посылки. Наличие прямоугольной скобки указывает на опциональность данного формата посылки.

Рис. 5.1. Форматы посылки

St – старт-бит имеет всегда низкий уровень.

0…8 – номер бита данных.

P – бит паритета: четность или нечетность.

*Sp*1, *Sp*2 – стоп-бит имеет всегда высокий уровень.

IDLE – состояние ожидания, в котором приостановлена передача на $R \times D$ или $T \times D$. В состоянии ожидания на линии должен быть высокий уровень.

Формат посылки, который используется *USART*, задается битами *UCSZ*2:0, *UPM*1:0 и *USBS* в регистрах *UCSRB* и *UCSRC*. Приемник и передатчик используют одни и те же установки форматов. Обра-

тите внимание, что изменение установок любого из этих бит может привести к повреждению текущего сеанса связи, как для приемника, так и для передатчика.

Биты выбора длины передаваемых данных (*UCSZ*2:0) определяют, из скольких бит данных состоит посылка. Биты режима паритета *USART* (*UPM*1:0) разрешают передачу/контроля бита паритета и устанавливают тип паритета: четность, нечетность. Выбрать один или два стоп-бита позволяет бит выбора стоп-бита *USART* (*USBS*). Приемник игнорирует второй стоп-бит. Флаг ошибки посылки *FE* позволяет выявить ситуацию, когда первый стоп-бит равен 0.

Ход работы

1. Создать проект программы передатчика при помощи *CodeWizard*. Выставить весь порт B на вход с подтягивающими резисторами.
2. Во вкладке *USART* настроить передачу данных. Для этого необходимо выставить режимы согласно рис. 5.2.

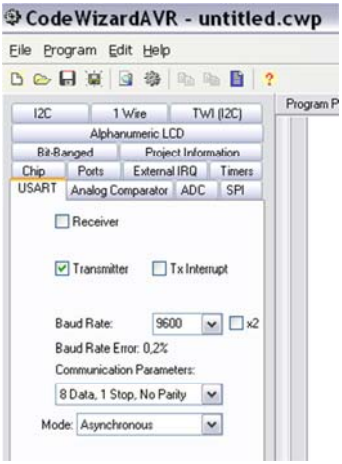

Рис. 5.2. Настройка *USART* на режим передачи данных

3. Получить программу, для передачи параллельного кода с порта *B*.

Листинг программы передатчика:

```
#include<mega8.h> // библиотека atmega8
#include<stdio.h>   // библиотека ввода/вывода

void main(void)
{
PORTB=0xFF;   // Порт B –  с подтягивающими резисторами
DDRB=0x00;   //  Порт B –  вход

UCSRA=0x00;  // регистр состояния UART
UCSRB=0x08;  // Регистры управления UART
UCSRC=0x86;  // формат кадра UART
UBRRH=0x00;
UBRRL=0x33;  // скорость 9600

while (1)
    { putchar(PINB); } // вывести символ с кодом порта B
}
```

4. Создать проект программы приемника. Выставить весь порт B на выход, с предустановкой логического «0».

5. Во вкладке *USART* настроить прием данных. Для этого необходимо выставить режимы согласно рис. 5.3.

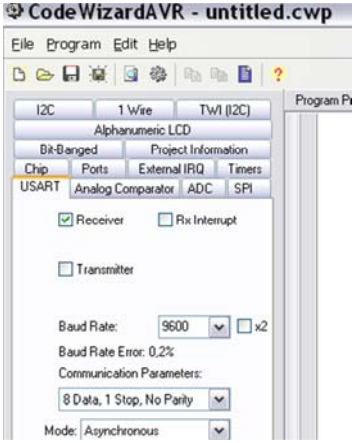

Рис. 5.3. Настройка *USART* на режим приема данных

6. Получить программу для приема последовательного кода и передачи его на порт *B* в виде параллельного кода.

Листинг программы приемника:

```
#include<mega8.h> // библиотека atmega8
#include<stdio.h> // библиотека ввода/вывода

void main(void)
{
DDRB=0xFF; // Порт B –  вывод

UCSRA=0x00;  // Регистр состояния UART
UCSRB=0x10;  // Регистр управления UART
UCSRC=0x86;  // Формат кадра UART
UBRRH=0x00;
UBRRL=0x33;  // Скорость 9600

while (1)
{
if (USR & 0x80){ // Если данные поступили, то вывести их на порт B
PORTB=UDR;}
}
}
```

7. Собрать схему в программе *PROTEUS* аналогичную рис. 5.4.

8. Зафиксировать в отчете формат пакета данных и соответствующий код для восьми вариантов передаваемой информации. Для этого с помощью набора переключателей задать любой код на порте *B* передатчика в двоичной форме. Параллельный код преобразуется в последовательный и передается в соответствии с протоколом на приемник. К порту *B* приемника подключена группа светодиодов, которые отображают полученный двоичный код.

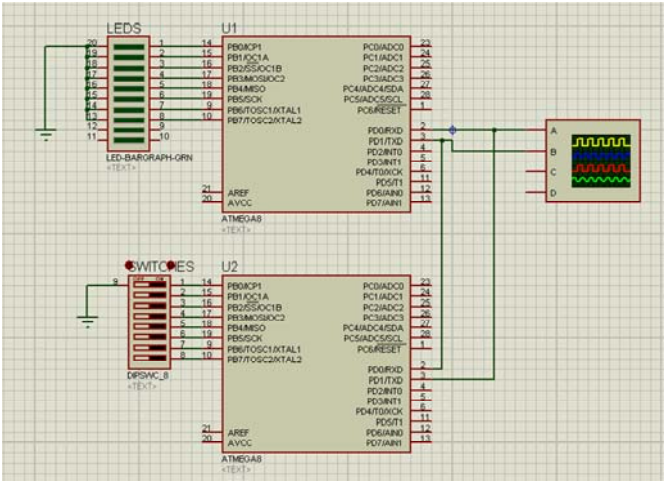

Рис. 5.4. Схема подключения микроконтроллеров по протоколу *USART*

Примечания к Proteus. Для наблюдения изменения сигнала воспользуйтесь осцилографом.

Содержание отчета

1. Цель работы.
2. Схема установки.
3. Двоичные коды и временные диграммы передаваемой информации.

Контрольные вопросы

1. Назначение и область применения использованного способа передачи данных.
2. Основные элементы аппаратно-прогрвммной реализации обмена с использованием микроконтроллера.

Лабораторная работа № 6

ИСПОЛЬЗОВАНИЕ SPI ДЛЯ ОБМЕНА ДАННЫМИ МЕЖДУ МИКРОКОНТРОЛЛЕРАМИ

Цель работы

изучить аппаратный способ передачи данных по средствам протокола *SPI*; научится использовать протокол *SPI* для обмена данными между устройствами.

Общие сведения

Интерфейс *SPI* позволяет организовать последовательную синхронную высокоскоростную передачу данных между *ATmega*128 и другим периферийным устройством или между несколькими *AVR*-микроконтроллерами.

Отличительные особенности интерфейса *SPI*:

– полнодуплексная, трехпроводная синхронная передача данных;
– ведущая или подчиненная работа;
– передача первым младшего или старшего бита;
– семь программируемых скоростей связи;
– флаг прерывания для индикации окончания передачи данных;
– защитный флаг при повторной записи;
– пробуждение из режима холостого хода (*Idle*);
– режим ведущего (мастера) *SPI* с удвоением скорости (*CK*/2).

Внешние соединения между ведущим (мастером) и подчиненным ЦПУ через интерфейс *SPI* показаны на рис. 6.1.

Система состоит из двух сдвиговых регистров и генератора ведущей синхронизации. Ведущий *SPI* инициирует сеанс связи подачей низкого уровня на вход *SS* того подчиненного устройства, с которым необходимо обмениваться данными. Оба респондента (ведущий и подчиненный) подготавливают данные к передаче в своем сдвиговом регистре, при этом на стороне ведущего генерируются также импульсы синхронизации на линии *SCK*. По линии *MOSI* всегда осуществляется передача данных от ведущего к подчиненному, а по *MISO*, наоборот, от подчиненного к мастеру. По окончании передачи каждого пакета данных ведущий *SPI* дол-

жен засинхронизировать подчиненный путем подачи высокого уровня на линию *SS* (выбор подчиненного интерфейса).

Если *SPI* настроен как ведущий (мастер), то управление линией *SS* происходит не автоматически. Данная операция должна быть выполнена программно перед началом сеанса связи. После этого, запись в регистр данных *SPI* инициирует генерацию синхронизации и аппаратный сдвиг 8-ми разрядов в подчиненное устройство. По окончании сдвига одного байта генератор синхронизации *SPI* останавливается, при этом устанавливая флаг окончания передачи (*SPIF*). Если установлен бит *SPIE* в регистре *SPCR*, то разрешается прерывание *SPI* и по окончании передачи байта будет генерирован запрос на прерывание. Мастер может продолжить сдвигать следующий байт, если записать его в регистр *SPDR*, или подать сигнал окончания пакета путем установки низкого уровня на линии *SS*. Последний принятый байт сохраняется в буферном регистре. В режиме подчиненного, интерфейс *SPI* находится в состоянии ожидания, в котором *MISO* переводится в третье состояние, до тех пор, пока на выводе *SS* присутствует высокий уровень. В этом состоянии программа может обновлять содержимое регистра данных *SPI* (*SPDR*), но при этом входящие импульсы синхронизации не сдвигают данные до подачи низкого уровня на вывод SS. После того как один байт был полностью сдвинут, устанавливается флаг окончания передачи *SPIF*. Если установлен бит разрешения прерывания *SPI* (*SPIE*) в регистре *SPCR*, то установка флага *SPIF* приводит к генерации запроса на прерывание. Подчиненный может продолжать размещать новые данные для передачи в регистр *SPDR* перед чтением входящих данных. Последний принятый байт хранится в буферном регистре.

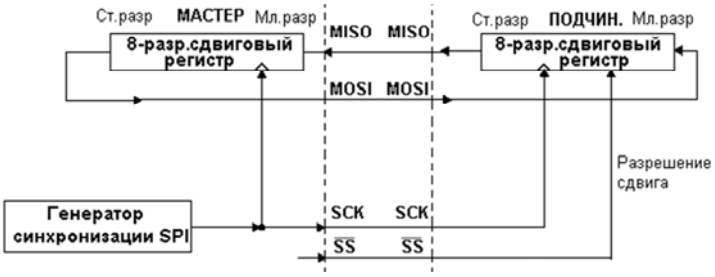

Рис. 6.1. Внешнее соединение ведущего (мастера) и подчиненного SPI

В направлении передачи данных система выполнена как однобуферная, а в направлении приема используется двойная буферизация. Это означает, что передаваемые байты не могут быть записаны в регистр данных *SPI*, до того, как полностью завершится цикл сдвига. Во время приема данных необходимо следить, чтобы принятая посылка была считана из регистра данных *SPI*, прежде чем завершится цикл входящего сдвига новой посылки. В противном случае первый байт будет потерян.

В подчиненном режиме *SPI* управляющая логика осуществляет выборку входящего сигнала *SCK*. Чтобы гарантировать корректность выборки тактового сигнала необходимо использовать частоту синхронизации *SPI* не более *fosc*/4.

Ход работы

1. Создать проект программы передатчика. Выставить весь порт *D* на вход с подтягивающими резисторами; *B*4 – на вход с подтягивающим резистором; *B*3, *B*4 – на выход с предустановкой логического «0».

2. Во вкладке *SPI* настроить передачу данных. Для этого необходимо выставить режимы согласно рис. 6.2.

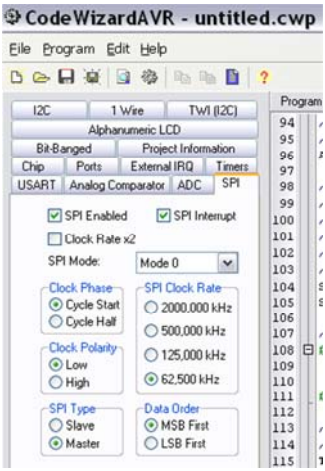

Рис. 6.2. Настройка *SPI* на режим передачи данных

3. Получить программу передачи параллельного кода с порта *D*.

Листинг программы передатчика:

```
#include <mega8.h> // библиотека atmega8

// SPI прерывание
interrupt [SPI_STC] void spi_isr(void)
{
SPDR=PIND; // Записать данные из порта D
}

void main(void)
{
PORTB=0x10; // B4 – с подтягивающим резистором
DDRB=0x28; // B3,B5 – на выход
PORTD=0xFF; // порт D – с подтягивающими резисторами

SPCR=0xD3; // регистр управления

#asm     // очистка флага прерывания SPI
in   r30,spsr
   in   r30,spdr
#endasm

#asm("sei") // глобальное разрешение прерываний
SPDR=0x00;  //вызов прерывания
while (1);
}
```

4. Создать проект программы приемника. Выставить весь порт *D* и *B*4 на выход с предустановкой логического «0»; *B*2, *B*3, *B*5 – на вход с подтягивающими резисторами.

5. Во вкладке *SPI* настроить прием данных. Для этого необходимо выставить режимы согласно рис. 6.3.

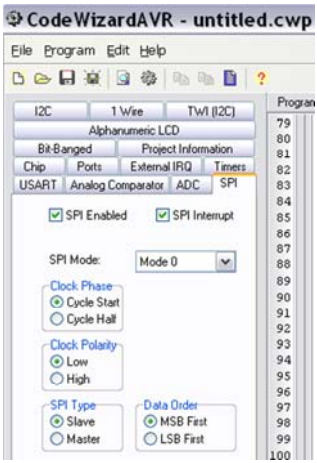

Рис. 6.3. Настройка *SPI* на режим приема данных

6. Получить программу для приема последовательного кода и передачи его на порт *D* в виде параллельного кода.

Листинг программы приемника:

```
#include<mega8.h> //библиотека atmega8

// прерывание SPI
interrupt [SPI_STC] void spi_isr(void)
{
PORTD=SPDR; // считать данные в порт D
}

voidmain(void)
{
PORTB=0x2C; // B2, B3, B5 – с подтягивающими резисторами
DDRB=0x10;  // B4 – на выход
DDRD=0xFF; // порт D – на выход

SPCR=0xC0; // регистр управления

#asm  // очистка флага прерывания SPI
```

```
in   r30,spsr
in   r30,spdr
#endasm

#asm("sei")  // глобальное разрешение прерываний
SPDR=0x00;  //вызов прерывания
while (1);
}
```

7. Собрать схему в программе *PROTEUS* аналогично рис. 6.4.

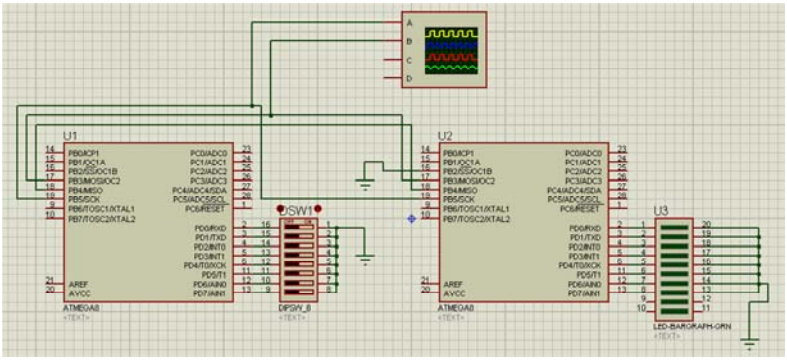

Рис. 6.4. Схема подключения микроконтроллеров по протоколу *SPI*

8. Зафиксировать в отчете формат пакета данных и соответствующий код для восьми вариантов передаваемой информации. Для этого с помощью набора переключателей задать любой код на порте *D* передатчика в двоичной форме. Параллельный код преобразуется в последовательный и передается в соответствии с протоколом в приемник. К порту *D* приемника подключена группа светодиодов, которые отображают полученный двоичный код.

Примечания к Proteus. Для наблюдения изменения сигнала воспользуйтесь осцилографом.

Содержание отчета и контрольные вопросы см. лабораторную работу № 5.

Лабораторная работа № 7

ИСПОЛЬЗОВАНИЕ TWI ДЛЯ ОБМЕНА ДАННЫМИ МЕЖДУ МИКРОКОНТРОЛЛЕРАМИ

Цель работы

изучить аппаратный способ передачи данных по средствам протокола *TWI*; научится использовать протокол *TWI* для обмена данными между устройствами.

Общие сведения

Отличительные особенности:

– гибкий, простой, при этом эффективный последовательный коммуникационный интерфейс, требующий только две линии связи;

– поддержка как ведущей, так и подчиненной работы;

– возможность работы, как приемника, так и как передатчика;

– 7-разр. адресное пространство позволяет подключить к шине до 128 подчиненных устройств;

– поддержка многомастерного арбитрирования;

– скорость передачи данных до 400 кГц;

– выходы драйверов с ограниченной скоростью изменения сигналов;

– схема шумоподавления повышает стойкость к выбросам на линиях шины;

– программируемый адрес для подчиненного режима с поддержкой общего вызова;

– пробуждение микроконтроллера из режима сна при определении заданного адреса на шине.

Определение шины TWI

Двухпроводной последовательный интерфейс *TWI* идеально подходит для типичных применений микроконтроллера. Протокол *TWI* позволяет проектировщику системы внешне связать до 128 различных устройств через одну двухпроводную двунаправленную шину, где одна линия – линия синхронизации *SCL*, а другая – линия данных *SDA*. В качестве внешних аппаратных компонентов, которые требуются для реализации шины, необходимы только подтягивающий

к плюсу питания резистор на каждой линии шины. Все устройства, которые подключены к шине, имеют свой индивидуальный адрес, а механизм определения содержимого шины поддерживается протоколом *TWI*.

Внешнее электрическое соединение

Как показано на рис. 7.1, обе линии шины подключены к положительной шине питания через подтягивающие резисторы. Среди всех совместимых с *TWI* устройствами в качестве драйверов шины используются транзистор или с открытым стоком или с открытым коллектором. Этим реализована функция монтажного И, которая очень важна для двунаправленной работы интерфейса. Низкий логический уровень на линии шины *TWI* генерируется, если одно или более из *TWI*-устройств выводит «0». Высокий уровень на линии присутствует, если все *TWI*-устройства перешли в третье высокоимпедансное состояние, позволяя подтягивающим резисторам задать уровень лог «1». Обратите внимание, что при подключении к шине *TWI* нескольких *AVR*-микроконтроллеров, для работы шины должны быть запитаны все из этих микроконтроллеров.

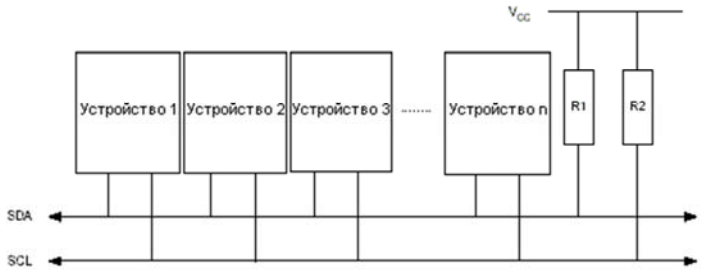

Рис. 7.1. Внешние подключения к шине *TWI*

Количество устройств, которое может быть подключено к одной шине ограничивается предельно допустимой емкостью шины (400 пФ) и 7-разрядным адресным пространством. Поддерживаются два различных набора технических требований, где один набор для шин со скоростью передачи данных ниже 100 кГц и один действителен для скоростей свыше 400 кГц.

На рис. 7.2 показана типичная передача данных.

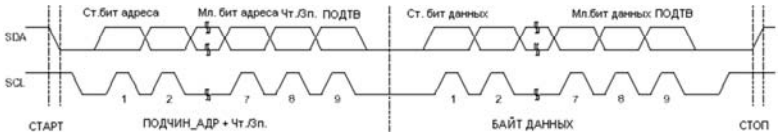

Рис.7.2. Типичная передача данных

Ход работы

1. Создать проект программы передатчика. Выставить весь порт B на вход с подтягивающими резисторами.

2. Во вкладке TWI настроить передачу данных. Для этого необходимо выставить режимы согласно рис. 7.3.

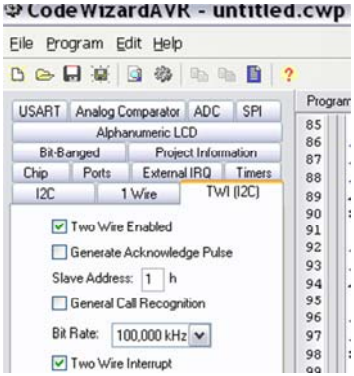

Рис.7.3. Настройки *TWI* для режима передачи данных

3. Получить программу передачи параллельного кода с порта B.

Листинг программы передатчика:

```
#include <mega8.h>  // библиотека atmega8
#include <delay.h>  // библиотека задержки

// прерывание TWI
interrupt [TWI] void twi_isr(void)
{
if (TWSR==0x08){TWDR=0x04; // Проверка флага готовности, отправка адреса
```

```
TWCR=0x85;}
if (TWSR==0x18){TWDR=PINB; // Проверка флага готовности, отправка данных
TWCR=0x85;}
if (TWSR==0x28){TWCR=0x95;} // Проверка готовности, завершение передачи
}

void main(void)
{
DDRB=0x00;  // Порт B – навход
PORTB=0xFF;  // Порт B – с подтягивающими резисторами
TWBR=0x20; // регистр скорости TWI
TWAR=0x02; // регистр адреса TWI
TWCR=0x05; // регистр управления TWI

#asm("sei") // глобальное разрешение прерываний
while (1){delay_ms(20); TWCR=0xA5;} //  старт передачи
}
```

4. Создать проект программы приемника. Выставить весь порт D на выход с предустановкой логического «0»;

5. Во вкладке *TWI* настроить прием данных. Для этого необходимо выставить режимы согласно рис. 7.4.

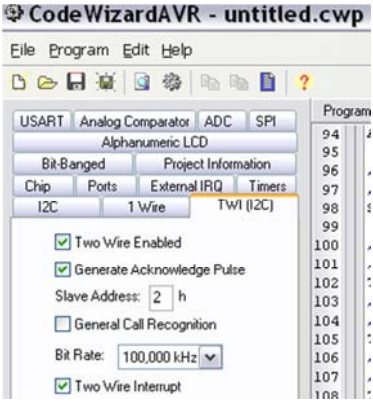

Рис.7.4. Настройки *TWI* для режима приема данных

6. Получить программу для приема последовательного кода и передачи его на порт *D* в виде параллельного кода.

Листинг программы приемника:

```
#include <mega8.h> // библиотека atmega8

// прерывание TWI
interrupt [TWI] void twi_isr(void)
{
if(TWSR==0x60){TWCR=0xC5;} // Еслиадреспринят, сбросфлага TWINT
if(TWSR==0x80){PORTD=TWDR;} // Если данные приянты, то записать их в порт D
}

void main(void)
{
DDRD=0xFF; // порт D – выход

TWBR=0x20; // регистр скорости TWI
TWAR=0x04; // регистр адреса TWI
TWCR=0x45; // регистр управления TWI

#asm("sei") // глобальное разрешение прерываний

while (1);
}
```

7. Собрать схему в программе *PROTEUS* аналогично рис. 7.5.

8. Зафиксировать в отчете формат пакета данных и соответствующий код для восьми вариантов передаваемой информации. Для этого с помощью набора переключателей задать любой код на порте B передатчика в двоичной форме. Параллельный код преобразуется в последовательный и передается в соответствии с протоколом в приемник. К порту *D* приемника подключена группа светодиодов, которые отображают полученный двоичный код.

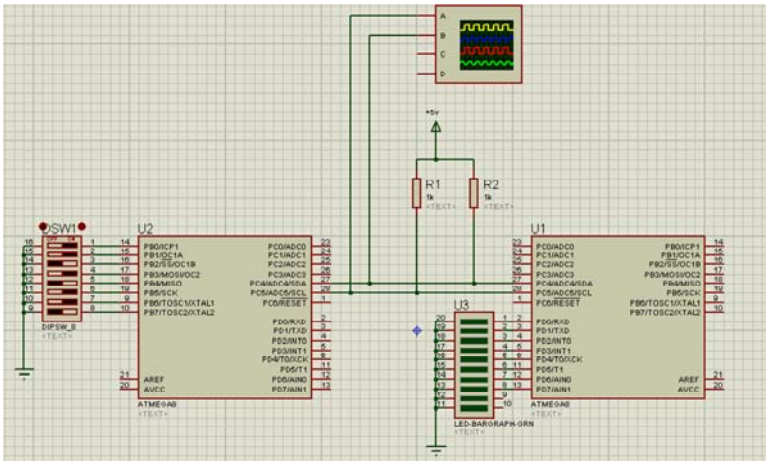

Рис. 7.5. Схема подключения микроконтроллеров по протоколу *TWI*

Примечания к Proteus. Для наблюдения изменения сигнала воспользуйтесь осцилографом.

Содержание отчета и контрольные вопросы см. лабораторную работу № 5.

Лабораторная работа № 8

ЗНАКОМСТВО С МАНЧЕСТЕРСКИМ КОДИРОВАНИЕМ ПРИ ОБМЕНЕ ДАННЫХ МЕЖДУ МИКРОКОНТРОЛЛЕРАМИ

Цель работы

познакомится с программным способом передачи данных посредством манчестерского кодирования.

Общие сведения

В манчестерском коде для кодирования единиц и нулей используется перепад потенциала, т. е. фронт импульса. При манчестерском кодировании каждый такт делится на две части. Информация кодируется перепадами потенциала, происходящими в середине каждого такта. Единица кодируется перепадом от низкого уровня сигнала к высокому, а ноль – обратным перепадом. В начале каждого такта может происходить служебный перепад сигнала, если нужно представить несколько единиц или нулей подряд. Так как сигнал изменяется по крайней мере один раз за такт передачи одного бита данных, то манчестерский код обладает хорошими самосинхронизирующими свойствами. Полоса пропускания манчестерского кода уже, чем у биполярного импульсного. У него также нет постоянной составляющей, а основная гармоника в худшем случае (при передаче последовательности единиц или нулей) имеет частоту N Гц, а в лучшем (при передаче чередующихся единиц и нулей) она равна $N/2$ Гц, как и у кодов *AMI* или *NRZ*. В среднем ширина полосы манчестерского кода в полтора раза уже, чем у биполярного импульсного, а основная гармоника колеблется вблизи значения $3N/4$. Манчестерский код имеет еще одно преимущество перед биполярным импульсным кодом. В последнем для передачи данных используются три уровня сигнала, а в манчестерском – два.

Формат пакета данных представлен на рис. 8.1.

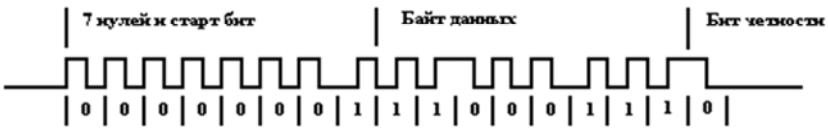

Рис. 8.1. Формат пакета данных

Ход работы

1. Построить в *PROTEUS* схему аналогичную рис. 8.2.

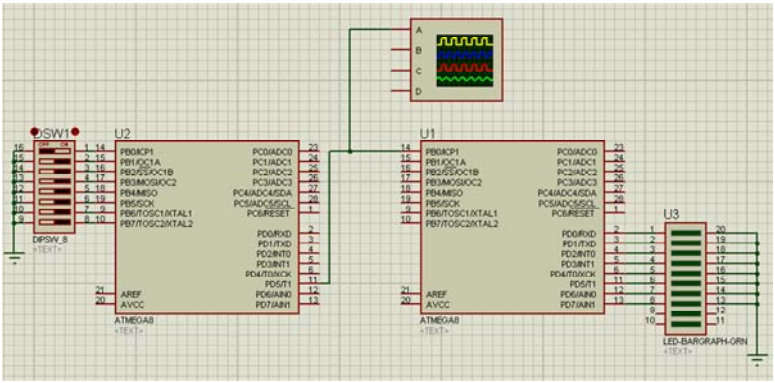

Рис. 8.2. Схема подключения микроконтроллеров

2. Записать программы передатчика и приемника в мк.

3. Изменяя параллельный код на мк-передатчике пронаблюдать за изменением передаваемого пакета данных в последовательном коде.

4. Зафиксировать в отчете формат пакета данных и соответствующий код для восьми вариантов передаваемой информации.

Листинг программы передатчика:

```
#include<mega8.h>
#include <stdio.h>
#include <delay.h>
#define BUFFSIZE 17
#define TIMING 1
```

```
#define SAMPLING 0
#define IOPORT PORTD
#define DATAOUT 5
#define CODINGTIMERCNT  TCNT1
#define CODINGTIMERCTRA TCCR1A
#define CODINGTIMERMSK  TIMSK
#define CODINGTIMERCTRB TCCR1B
#define CODINGTIMERFLR  TIFR

#define sbi(port,bit)   (port |= (1<<bit))
#define cbi(port,bit)   (port &= ~(1<<bit))
#define tgl(port,bit)   (port ^= (1<<bit))
#define tst(port,bit) (((port)&(1<<(bit)))>>(bit))

volatile unsigned intRdTime = 0;
volatile unsigned char *cDataBuffPtr;
volatile unsigned char cDataBuff[BUFFSIZE] = {0};

interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
 CODINGTIMERCNT = 0x0000;
 RdTime = 0xFFFF;
 }

interrupt [TIM1_CAPT] void timer1_capt_isr(void)
{
 CODINGTIMERCNT = 0x0000;
 tgl(CODINGTIMERCTRB,ICES1);
 RdTime = ICR1;
 }

interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
 CODINGTIMERCNT = 0x0000;
 RdTime = 1;
 }

voidCoding_TimerInit()
```

```
    {
    CODINGTIMERMSK = 0x00;      //Disable TC1 interrupts
    cDataBuffPtr = &cDataBuff[0]; // Place pointer at beginning of buffer
    OCR1A = 128;
    CODINGTIMERCNT = 0x00;     //Load TC1 count value
    sbi(CODINGTIMERMSK,OCIE1A);       // Timer/Counter1 Output
Compare A
    CODINGTIMERFLR |= 0x27; //clear interrupt flags for TC1
    CODINGTIMERCTRA = 0x00;  //Normal mode
    CODINGTIMERCTRB |= (1<<CS11); //prescale  = clock  source/  8
                                  //exactly 1 us for every timer step
    }

   unsignedintCoding_Timer_Poll(void)
   {
    #asm("sei");                // Turn on global Interrupts
    RdTime = 0; // Clear timing measurement
    while(RdTime == 0){} // Wait for interrupt to generate measurement
    returnRdTime;
    }

   voidCoding_ManchesterEncode(unsigned char numBits)
   {
   volatile unsigned intcNumBits = 0,i;
   cDataBuffPtr = &cDataBuff[0]; // Place pointer at beginning of buffer
   Coding_TimerInit(); // Init timer w/ periodic interrupt
   for(i=0; i<numBits; i++) // Повторять пока все данные не будут
переданы
   {
   if(cNumBits == 8) // Когда весь байт передан
   {
   cDataBuffPtr++; // Следующий байт
   if(cDataBuffPtr == &cDataBuff[0]){i=numBits+1;}
   cNumBits = 0; // Очиститьсчетчикбитов
   }
   if((*cDataBuffPtr& 0x80) == 0x80) // Check bit value, process logic one
       {
   cbi(IOPORT,DATAOUT); // Установить I/O низкий
```

```
Coding_Timer_Poll(); // Catch next interrupt
sbi(IOPORT,DATAOUT); // Установить I/O высокий
Coding_Timer_Poll(); // Catch next interrupt
}
else
{
sbi(IOPORT,DATAOUT); // Установить I/O высокий
Coding_Timer_Poll(); // Catch next interrupt
cbi(IOPORT,DATAOUT); // Установить I/O низкий
Coding_Timer_Poll(); // Catch next interrupt
 }
*cDataBuffPtr = *cDataBuffPtr<<1; // Shift buffer to get next bit
cNumBits++; // Increment number of bits sent
 }
 }

unsigned char coun(unsigned char data)
{
 char c=0,i;
 for(i=0;i<8;i++)
{
 if ((data&0x01)==0x01){c++;};
 data=data>>1;
 }
 if (c%2==0)
 {
  c=0xFF;
  }
  else{c=0x00;}
  return c;
  }

void main(void)
{  DDRD.5=1;
    DDRB=0x00;
    PORTB=0xFF;
    #asm("sei");
while (1){
```

```
cDataBuff[0] = 0x01;
cDataBuff[1] = PINB;
cDataBuff[2] = coun(cDataBuff[1]);
delay_ms(10);
Coding_ManchesterEncode(17); }
}
```

Листинг программы приемника:

```
#include <mega8.h>
#include <stdio.h>
#include <delay.h>
#define IOPIN PINB
#define DATAIN 0
#define CODINGTIMERCNT TCNT1
#define CODINGTIMERCTRA TCCR1A
#define CODINGTIMERCTRB TCCR1B
#define CODINGTIMERMSK TIMSK
#define CODINGTIMERFLR TIFR
#define BUFFSIZE 16
#define SAMPLING 0
#define TIMING 1
#define sbi(port,bit)   (port |= (1<<bit))  // Set bit in port
#define cbi(port,bit)   (port &= ~(1<<bit))   // Clear bit in port
#define tgl(port,bit)   (port ^= (1<<bit))    // Toggle bit in port
#define tst(port,bit) (((port)&(1<<(bit)))>>(bit))// Test bit in port
volatile unsigned char cDataBuff[BUFFSIZE] = {0};
volatile unsigned char *cDataBuffPtr;
volatile unsigned char numSampleBits = 0;
unsigned char tmp=0,tmp1=0;

void Coding_TimerInit(unsigned char mode)
{
CODINGTIMERMSK = 0x00;      //Отключение прерываний с T1
cDataBuffPtr = &cDataBuff[0]; // Place pointer at beginning of buffer
OCR1A = 32;      // Сравнение с числом
CODINGTIMERCNT = 0x00;     //Сброс таймера*
if (mode==TIMING)
```

```
    {
     sbi(CODINGTIMERMSK,TICIE1);
     }
     else {sbi(CODINGTIMERMSK,OCIE1A);}   // По результату
сранения А
     CODINGTIMERFLR |= 0x27; //очистить флаги прерываний
таймера
     CODINGTIMERCTRA = 0x00;  //Нормальная работа порта
     sbi(CODINGTIMERCTRB,ICES1);         //Прерывание по
нарастающему фронту
     CODINGTIMERCTRB |= (1<<CS11);  //предделение  =частота/  8
                                //1 мкс на каждый шаг
    }

    interrupt [TIM1_CAPT] void timer1_capt_isr(void)
        {
    cDataBuff[BUFFSIZE-1] = 0x00;
    Coding_TimerInit(SAMPLING);
    }

    unsigned char coun(unsigned char data,unsigned char m)
    {
    unsigned char c=0,i;
    for(i=0;i<8;i++)
    {
     if ((data&0x01)==0x01){c++;};
     data=data>>1;
     }
    if (c%2==0)
     {
      c=0x01;
      }
    else{c=0x00;}
    if (c!=m){c=0;}else{c=1;};
    return c;
    }

    void Coding_DSP()
```

```
        {
        unsigned char count=0, cLong=0, cShort=0;
        unsigned char i,c=0, logicFlag, cNumBit=0, syncFlag=0;
        unsigned char tmpData,j, bitVal=0, buff=0x00, buff1=0x00;
        if((*cDataBuffPtr & 0x80) == 0x80){logicFlag = 1;} // Initialize
logic flag
        else{logicFlag = 0;}
        for(j=0; j<BUFFSIZE; j++) // Process entire buffer
        {
         tmpData = *cDataBuffPtr++; // Pull out working byte
         for(i=0; i<8; i++) // Process entire byte
         {
         if(!syncFlag)
         {
          if(logicFlag == 1 && (tmpData & 0x80) == 0x80){count++;}
          else if(logicFlag == 0 && (tmpData & 0x80) ==
0x00){count++;}
          else
          {
          logicFlag = logicFlag^0x01; // Инверсия флага
          if(count > 4)
          {
          syncFlag=1; // 2T найден
          bitVal = logicFlag;
          }
          count=1; // Очистить счетчик
          }
          }
         else
         {
          if(logicFlag == 1 && (tmpData & 0x80) == 0x80){count++;}
          else if(logicFlag == 0 && (tmpData & 0x80) ==
0x00){count++;}
          else
          {
           // Check if count below threshold, inc short
          if(count <=4){cShort++;}
          else{cLong++;} // else inc long
```

```
count=1; // Reset count
logicFlag = logicFlag^0x01; // Инверсия флага
if(cLong == 1)
{
cLong = 0;
bitVal = bitVal^0x01;
if(bitVal == 1)
{
 if (c<8){
 if (buff1!=0x00){c=8;};
 buff=buff << 1;
 buff=buff | 0x01;
 c++;}
 }
 else if(bitVal == 0)
 {
 if (c<8){
 if (buff1!=0x00){c=8;};
 buff=buff << 1;
 c++;}
 }
 cNumBit++;
 }
 else if(cShort == 2)
 {
  cShort = 0;
  if(bitVal == 1)
 {
 if (c<8){
 if (buff1!=0x00){c=8;};
 buff=buff << 1;
 buff=buff | 0x01;
 c++;}
 }
 else if(bitVal == 0)
 {
  if(c<8){
  if (buff1!=0x00){c=8;};
```

```
        buff=buff << 1;
        c++;}
        }
        cNumBit++;
        }
        if(cNumBit == 8) // When full byte is read
        {
        buff1=buff;
        buff=0;
        c=0;
        cNumBit = 0; // Clear bit counter
        }
        }
        }
        tmpData = tmpData << 1; // Shift working byte to next bit
         }
         }
         if (coun(buff1,buff)==1){PORTD=buff1;};
         }

interrupt [TIM1_COMPA] void timer1_compa_isr(void)
    {
    CODINGTIMERCNT = 0x0000;
    if(numSampleBits == 8)
    {
     numSampleBits = 0;
    cDataBuffPtr++;
    }
    *cDataBuffPtr = *cDataBuffPtr<<1;
    if(tst(IOPIN,DATAIN) == 1)
    {
     *cDataBuffPtr = *cDataBuffPtr|0x01;
     tmp=0;
     tmp1++;
     }
    else
     {
      tmp++;
```

```
        tmp1=0;
        }
      numSampleBits++;
      if ((tmp>9)||(tmp1>9))
      {
       tmp=0;
       tmp1=0;
       Coding_TimerInit(TIMING);
       Coding_DSP();
       }
      }

  void main(void)
  { #asm("sei")
   PORTD=0x00;
   DDRD=0xFF;
   Coding_TimerInit(TIMING);
   PORTD.2=0; DDRD.2=1;
  }
```

Содержание отчета и контрольные вопросы см. лабораторную работу № 5.

Список используемой литературы

1. Евстифеев, А.В. Микроконтроллеры AVR семейства Tiny и Mega фирмы «Atmel» / А. В. Евстифеев. – М.: Издательский дом «Додэка-XXI», 2004. – 560 с.

2. Мортон, Дж. Микроконтроллеры AVR. Вводный курс / пер. с англ. – М.: издательский дом «Додэка-XXI», 2006. – 272 с.: ил. – (Серия «Мировая электроника»).

3. Ревич, Ю.В. Практическое программирование микроконтроллеров AtmelAVR на языке ассемблера / Ю. В. Ревич. – СПб.: БХВ-Петербург, 2008. – 384 с. – (Аппаратные средства).

4. Программирование на языке С для AVR и PIC микроконтроллеров / сост.: Ю. А. Шпак. – Киев: МК-Пресс, 2006. – 400 с., ил.

5. Лебедев, М.Б. CodeVisionAVR: пособие для начинающих / М. Б. Лебедев. – М.: Додэка-XXI, 2008. – 592 с.: ил.

ОГЛАВЛЕНИЕ

Учебное издание

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

Лабораторный практикум
для студентов специальностей 1-53 01 01
«Автоматизация технологических процессов и производств»
и 1-53 01 06 «Промышленные роботы
и робототехнические комплексы»

В 2 частях

Часть 2

Составители:
СИРОТИН Феликс Львович
НОВИЧИХИНА Елена Романовна

Редактор *О. В. Ткачук*
Компьютерная верстка *А. Е. Дарвина*

Подписано в печать 20.04.16. Формат 60×84 $^{1}/_{16}$. Бумага офсетная. Ризография.
Усл. печ. л. 1,98. Уч.-изд. л. 1,54. Тираж 100. Заказ 87.

Издатель и полиграфическое исполнение: Белорусский национальный технический университет.
Свидетельство о государственной регистрации издателя, изготовителя, распространителя
печатных изданий № 1/173 от 12.02.2014. Пр. Независимости, 65. 220013, г. Минск.