Лабораторная работа №1 «Изучение микроконтроллеров и их программирование»

Цель работы: ознакомиться с принципом работы и основами программирования 8-разрядных микроконтроллеров.

Задачи: изучить структуру микроконтроллера, его интерфейсы, работу внутренних регистров микроконтроллера, программные средства WinAVR, PonyProg, последовательность действий для программирования микроконтроллера, лабораторный макет проведения лабораторных работ.

Приборы и принадлежности: плата на основе микроконтроллера ATtiny2313, Datasheet к микроконтроллеру ATtiny2313, LPT- программатор, ПК с установленными PonyProg и пакетом программ WinAVR.

Краткие теоретические сведения

Микроконтроллер − компьютер на одной микросхеме, предназначенный для управления различными электронными устройствами и осуществления взаимодействия между ними в соответствии с заложенной в него программой.

В отличие от микропроцессоров, используемых в персональных компьютерах, микроконтроллеры содержат встроенные дополнительные устройства: устройства памяти, порты ввода/вывода (I/O), интерфейсы связи, таймеры, системные часы. Эти устройства выполняют свои задачи под управлением микропроцессорного ядра микроконтроллера. На рисунке 1 изображена обобщенная структурная схема микроконтроллера.

В состав микроконтроллера входят: генератор тактового сигнала (GCK); процессор (CPU); постоянное запоминающее устройство для хранения программы, выполненное по технологии Flash (FlashROM); оперативное запоминающее устройство статического типа для хранения данных (SRAM); постоянное запоминающее устройство для хранения данных, выполненное по технологии EEPROM(EEPROM); набор периферийных устройств для ввода и вывода данных и управляющих сигналов и выполнения других функций.

Микроконтроллеры семейства AVR являются устройствами синхронного типа. Действия, выполняемые в микроконтроллере, привязаны к импульсам тактового сигнала. В качестве генератора тактового сигнала может использоваться внутренний генератор с внешним кварцевым или керамическим резонатором (XTAL); внутренний RC-генератор (IRC); внутренний генератор с внешней RC-цепочкой (ERC); внешний генератор (EXT).

Процессор формирует адрес очередной команды, выбирает команду из памяти и организует ее выполнение. Код команды имеет формат "слово" (8 бит) или "два слова" (16 бит). В состав процессора входят счетчик команд, арифметико-логическое устройство и блок регистров общего назначения.

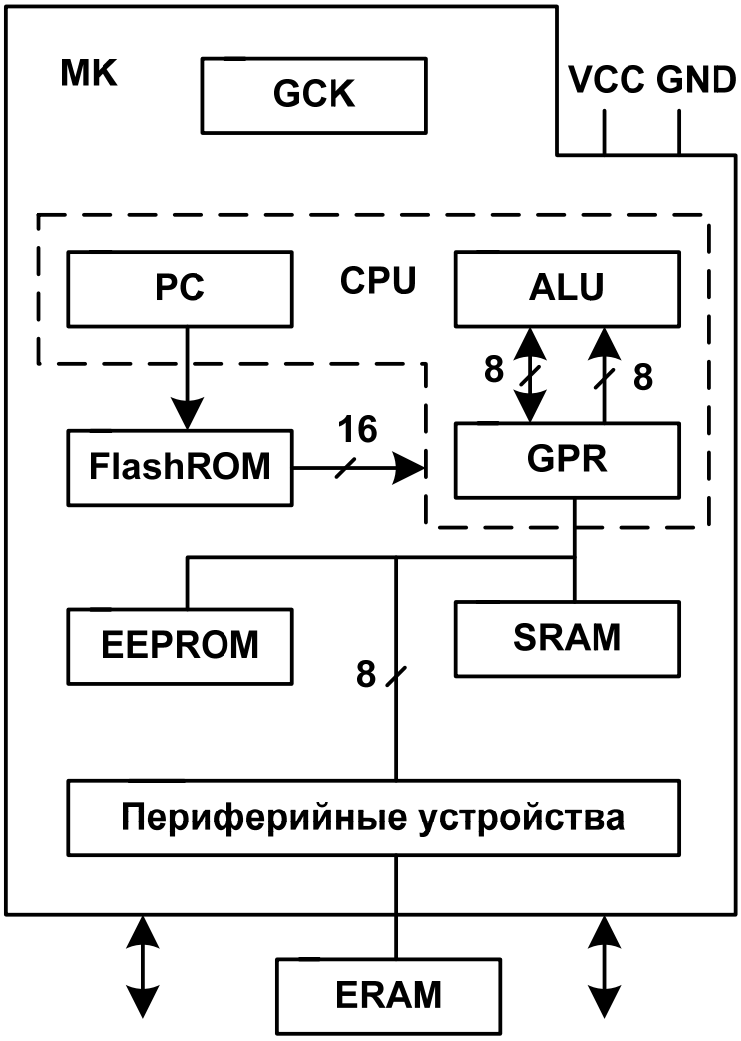


Рисунок 1 – Обобщенная структуная схема микроконтроллера

Счетчик команд представляет собой регистр, в котором содержится адрес следующей исполняемой команды. Размер счетчика команд зависит от объема имеющейся памяти программ. При нормальном выполнении программы содержимое счетчика автоматически увеличивается на 1 или на 2 в каждом такте. Этот порядок нарушается при выполнении команд перехода, вызова и возврата из подпрограмм, а также при возникновении прерываний.

После включения питания, а также после сброса микроконтроллера счетчик команд автоматически обнуляется, и первая команда выбирается из FlashROM по адресу 0, а при возникновении прерываний в него загружается адрес соответствующего вектора прерывания.

Блок регистров общего назначения служит для хранения промежуточных результатов вычислений и состоит из 32 регистров (R0…R31).

В арифметико-логическом устройстве выполняются арифметические и логические операции. Операнды поступают из регистров общего назначения. Каждая команда (16 арифметических и 16 логических) в качестве операндов использует либо содержимое двух разных регистров общего назначения, либо содержимое регистра и константу. Результат вычислений также помещается в один из регистров.

Постоянное запоминающее устройство FlashROM предназначено для хранения кодов команд программы и констант. Ячейка памяти содержит 16 разрядов. В ней могут храниться код команды формата "слово", половина кода команды формата "два слова" или коды двух констант. При чтении кодов команд адрес в FlashROM поступает из счетчика команд. При чтении констант адрес поступает из пары R30-R31 регистров общего назначения. Запись кодов в FlashROM выполняется в процессе программирования побайтно.

Оперативное запоминающее устройство статического типа SRAM предназначено для хранения данных, получаемых в процессе работы микроконтроллера. При выключении напряжения питания микроконтроллера данные в SRAM теряются.

Постоянное запоминающее устройство EEPROM предназначено для хранения данных, записанных при программировании микроконтроллера и получаемых в процессе выполнения программы. При выключении напряжения питания данные сохраняются. Ячейка памяти содержит 8 разрядов.

В группу периферийных устройств входят: параллельные порты ввода-вывода; последовательный периферийный интерфейс SPI; универсальный синхронный/асинхронный приемо-передатчик USART; двухпроводный последовательный интерфейс TWSI (I2C); таймеры-счетчики общего назначения; сторожевой таймер и аналого-цифровой преобразователь; аналоговый компаратор; программируемый аппаратный модулятор; блок прерываний.

Параллельные порты ввода-вывода (PORTx) предназначены для ввода и вывода данных. Порт может иметь от трех до восьми выводов. Вывод порта может работать в режиме входа или в режиме выхода. Направление передачи бита устанавливается для каждого вывода в отдельности.

Некоторые выводы портов кроме ввода и вывода битов данных могут использоваться для выполнения альтернативных функций при работе дру-

гих устройств, например, для подключения кварцевого резонатора, устройств по интерфейсам SPI, USART и других.

Последовательный периферийный интерфейс SPI имеет два назначения: с его помощью может осуществляться обмен данными между микроконтроллером и различными периферийными устройствами, такими как цифровые потенциометры, ЦАП/АЦП,Flash-ПЗУ, между несколькими микроконтроллерами, а также программирование микроконтроллера.

Универсальный синхронный/асинхронный приемо-передатчик USART предназначен для передачи и приема байтов данных по двухпроводным линиям связи (например, по интерфейсу RS-232). Длина посылки может варьироваться от 5 до 9 бит. Во всех модулях USART присутствуют схемы контроля и формирования бита четности.

Интерфейс TWSI или его, еще обозначают I2C, служит для передачи данных и позволяет объединить до 128 различных устройств с помощью двунаправленной шины, состоящей всего из двух линий: линии тактового сигнала и линии данных. Суммарная скорость передачи по интерфейсу– 400 Кбит/с.

Таймер-счетчик общего назначения предназначен для формирования запроса прерывания при истечении заданного интервала времени (режим таймера) или свершении заданного числа событий (режим счетчика). Основным элементом таймера-счетчика является базовый счетчик, который ведет счет на увеличение. При его переполнении формируется запрос прерывания.

Таймер-счетчик общего назначения может выполнять дополнительные функции: функцию захвата, сравнения, широтно-импульсного модулятора, счета реального времени.

Сторожевой таймер предназначен для ликвидации последствий сбоя в ходе программы путем перезапуска микроконтроллера при обнаружении сбоя.

Аналого-цифровой преобразователь формирует разрядный двоичный код числа, пропорционального величине напряжения аналогового сигнала на входе микроконтроллера. В микроконтроллерах AVR к преобразователю могут подключаться от четырех до восьми входов.

Аналого-цифровой компаратор сравнивает по величине аналоговые сигналы, поступающие на два входа микроконтроллера, и формирует запрос прерывания, когда разность их значений меняет знак. При этом также может быть выдан сигнал для выполнения функции захвата в таймере-счетчике общего назначения.

Блок прерываний (Interrupt Unit, IU) организует переход к выполнению прерывающей программы при поступлении запроса прерывания, если прерывание по данному запросу разрешено и он имеет более высокий при-

оритет, чем другие запросы, поступившие одновременно с ним. Приоритетность запросов задана аппаратно.

В микроконтроллере особая четырехбайтовая область памяти отведена для конфигурационных бит (fuse bits) и бит защиты(lock bits), которые служат для глобальной настройки микрочипа. Если содержание регистров памяти микроконтроллер может изменять сам в процессе выполнения программы, то изменение fuse и lock битов доступно только с использованием программатора. При программировании этих битов нужно быть предельно внимательным и постоянно сверяться с Datasheet, так как можно навсегда испортить микроконтроллер.

Программа для микроконтроллера может быть написана на языках: Assembler, C/C++, Pascal, Basic, FlowCode или Scratah и т.д. Каждый из них имеет свои преимущества и недостатки в решении определенных задач, например, Assembler позволяет наиболее полно раскрыть все возможности микроконтроллеров и получить максимальное быстродействие и компактный код, но сама программа громоздка и труднопонимаема, визуальные языки наиболее просты, однако могут быть проблемы с оптимизацией кода. Широкое распространение получил язык C/C++ благодаря огромному числу программных средств и библиотек, позволяющих просто создавать необходимый код. C/C++ сегодня стал основным языком разработки управляющих программ.

Вышеприведенные языки не понятны микроконтроллеру. Для перевода программы в машинный код, который будет вшит в память микрочипа, используют специальные программы - компиляторы.

Программирование осуществляется с помощью специальных компьютерных программ-программаторов. Для сопряжения интерфейсов ПК с интерфейсами микроконтроллера применяют аппаратные программаторы-устройства, зачастую на основе другого микрочипа. Программатор является промежуточным звеном между компьютером и программируемым микроконтроллером. Со стороны микроконтроллера программатор может использовать интерфейс SPI для последовательного программирования, JTAG – для программирования и отладки в режиме реального времени, или несколько выводов, специально предназначенных для параллельного программирования.

На рисунке 2 изображены структура и электрическая схема платы для выполнения лабораторной работы на основе микроконтроллера ATtiny2313.

Используемый в лабораторной работе микроконтроллер ATtiny2313 выполнен в корпусе типа DIP и имеет 20 выводов, из которых 18 приходятся на 3 параллельных порта ввода-вывода PORTA, PORTB и PORTD и

2 на питание: к выводу VCC – подключается положительное напряжение до +5В, а вывод GND заземляется. PORTA и PORTD не полные: отсутствуют выводы PA3-PA7 и PD7 соответственно.

Как видно в схеме (рис. 2), выводы PORTB подключены через сопротивления к светодиодным индикаторам и одновременно выводятся на 10-пиновый порт №1 для подключения внешних устройств. Для отключения светодиодной панели индикации предусмотрены джамперы.

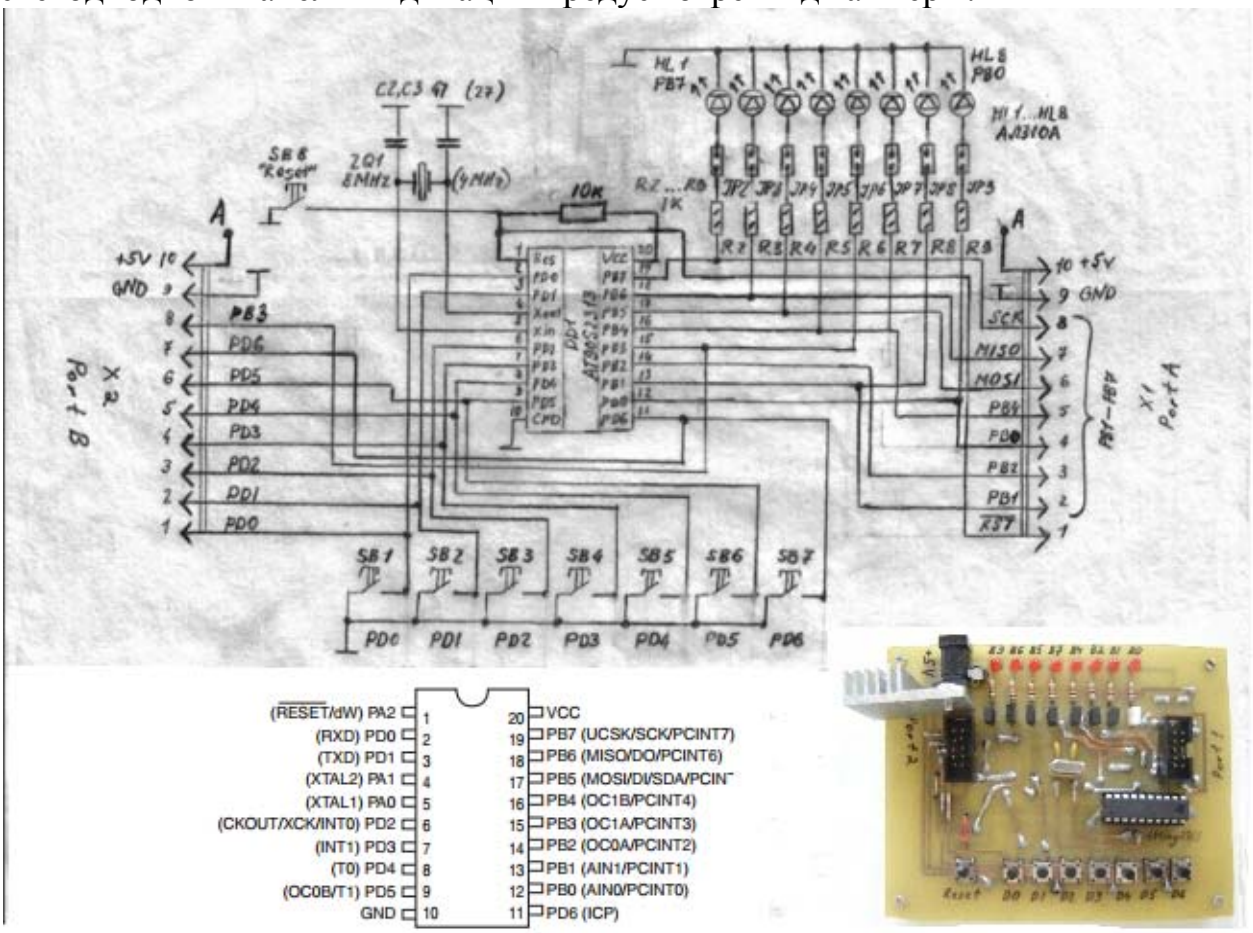


Рисунок 2 – Структура и электрическая схема платы на основе ATtiny2313

К выводам PORTD подключены кнопки, при нажатии которых происходит подключение их к земле. Одновременно выводы PORTD выводятся на 10-пиновый порт №2 для подключения внешних устройств.

Питание микроконтроллера осуществляется с помощью стабилизатора напряжения на 5 В через вывод VCC и GND.

Каждый из 18 выводов портов ввода-вывода выполняет альтернативные функции, например, для тактирования микроконтроллера к выводам PA0 и PA1 (XTAL1, XTAL2) подключен кварцевый резонатор на 8 Мгц, SPI интерфейс, через который будет программироваться микроконтроллер использует выводы PB5, PB6, PB7 (MOSI, MISO, SCK соответственно).

Интерфейс USART для приема/передачи данных использует выводы PD0, PD1, PD2 (RXD,TXD,XCK соответственно). Вывод PA2 (RESET) предназначен для внешнего сброса микроконтроллера и при подаче на него напряжения низкого уровня с помощью кнопки Reset, порты ввода-вывода немедленно возвращаются к их первоначальному состоянию, микроконтроллер перезагружается и активизируется только при подаче напряжения высокого уровня. В процессе программирования на выводе RESET должно присутствовать напряжение низкого уровня, а в процессе выполнения программы микроконтроллером – высокое.

Параллельные порты ввода-вывода предназначены для ввода и вывода бит данных. На каждом выводе порта может присутствовать напряжение высокого уровня, соответствующее логической единице, либо напряжение низкого уровня, соответствующее логическому нулю. Для каждого порта ввода/вывода в адресном пространстве микроконтроллера выделены 3 регистра: регистр данных порта PORTx, регистр направления данных DDRx и регистр выводов порта PINx (к примеру, для порта B – DDRB, PORTB, PINB).

Регистр DDRx устанавливает выводы порта x как вход или как выход данных. Если в него записаны 0, то порт сконфигурирован на вход данных, если записаны 1 - на выход. Определить направление данных можно для каждого вывода порта. Например, если записать DDRB=0b00000000, то все 8 выводов порта B сконфигурированы как входы, DDRB=0b11111111, то все 8 выводов порта B сконфигурированы как выходы,DDRB=0b00001111, то первые 4 вывода порта PB0…PB3 сконфигурированы как выходы, а остальные 4 вывода PB4…PB7 – как входы.

Регистр PORTx имеет двойное назначение. Если порт настроен на выход данных, то значение регистра PORTx определяет напряжение на выводах порта. Например, если записать DDRB=0b11111111 и PORTB=0b10111000 ,то на выводах PB0, PB1, PB2, PB6 окажется напряжение низкого уровня, на выводах PB3, PB4, PB5, PB7 напряжение высокого уровня.

Если же порт настроен на вход (DDRx=0b00000000) , то значение регистра PORTx определяет состояние внутреннего подтягивающего резистора на каждом выводе порта. Если в регистр PORTx записать единицы, то ко всем выводам порта x будут подключены подтягивающие резисторы. Подтягивающий резистор соединяет вывод порта с питанием микроконтроллера и служит для защиты от помех, при этом сигнал, на свободно висячей ножке, микроконтроллером расценивается как единица.

Регистр PINx указывает текущее состояние на выводах порта x и доступен только для чтения. Фактически регистр PINx используется при про-

граммировании в качестве переменной, а его значение соответствует значениям потенциалов на входных ножках.

Используя эти регистры, напишем программу, реализующую непрерывное последовательное мигание светодиодов, подключенных к выводам PB0 и PB1, если кнопка D1 не нажата, а при нажатии и удерживании кнопки D1, подключенной к выводу PD1, будет осуществляться последовательное мигание светодиодов, подключенных к выводам PB3 и PB4. После написания программы ее нужно будет скомпилировать и записать в память микроконтроллера (последнее действие называется программированием). Итак, выполним все последовательно:

1) Откроем «Programmer's Notepad», входящий в пакет программ WinAVR. Перейдем по вкладкам File→new→C/C++. Появится окно для написания программы на языке C/C++.

Сначала необходимо с помощью оператора #include подключить библиотеки для нашего микроконтроллера, в данном случае это ATtiny2313, располагающиеся в папках программы WinAVR. Понадобится библиотека iotn2313.h, в которой сопоставляются названия регистров с их адресами, и библиотека delay.h, в которой осуществлена программа функции задержки _delay_(ms), пересчитывающая секунды в количество тактов GCK генератора:

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

Любая программа на языке C/C++ начинается с функции main:

```
int main (void)
{
}
```

Слово int означает, что функция main является функцией целого типа, то есть возвращаемое ею значение – целое число. Слово void в скобках означает, что функция не принимает значений.

Порт B, на котором расположена светодиодная панель индикации, сконфигурируем как выход, а порт D с кнопочной панелью управления как вход:

```
DDRB=0xff;
DDRD=0x00;
```

Для защиты от помех при нажатии/отжатии кнопок, подключим все выводы порта D к внутренним подтягивающим резисторам:

```
PORTD=0xff;
```

Для того, чтобы светодиоды изначально не горели установим на все выводы порта B напряжение низкого уровня:

```
PORTB=0x00;
```

Для того, чтобы светодиоды B0…B3 загорались необходимо подать на соответствующие им выводы напряжение высокого уровня(логическая единица), потухли – напряжение низкого уровня (логический 0):

```
PORTB=0x01;
PORTB=0x00;
PORTB=0x02;
PORTB=0x00;
PORTB=0x04;
PORTB=0x00;
PORTB=0x08;
PORTB=0x00;
```

Если оставить все без изменений, то светодиоды будут загораться и гаснуть с частотой работы тактирующего генератора микроконтроллера (8Мгц), и мы увидим 4 горящих светодиода, так как они не будут успевать гаснуть. Для того, чтобы визуально было заметно, когда светодиод гаснет, применим функцию задержки _delay_(ms), в качестве аргумента которой указывается число, соответствующее времени задержки в мс:

```
PORTB=0x01;
_delay_(500);
PORTB=0x00;
_delay_(500);
PORTB=0x02;
_delay_(500);
PORTB=0x00;
……….
```

Теперь светодиоды будут загораться и гаснуть с интервалом 500 мс. Первые два светодиода должны мигать при отжатой кнопке, другие два – при нажатой. При отжатой кнопке D1 на выводе PD1 присутствует напряжение высокого уровня, благодаря внутреннему подтягивающему резистору, а при нажатой происходит утечка тока на землю и на выводе PD1 напряжение становится низкого уровня. Используем операторы if-else и регистр PIND:

```
if (PIND==125) //соответствует значениям битов в регистре 01111101
  {
   PORTB=0x01;
   _delay_(500);
   PORTB=0x00;
   _delay_(500);
   PORTB=0x02;
   _delay_(500);
   PORTB=0x00;
  }
else
  {
```

```
    PORTB=0x04;
      _delay_(500);
      PORTB=0x00;
      _delay_(500);
      PORTB=0x08;
      _delay_(500);
      PORTB=0x00;
    }
```

Для получения бесконечного цикла используем оператор условия while(), который выполняется до тех пор, пока значение в скобках истинно, то есть отлично от нуля.

Конечная программа имеет вид:

```
#include <avr/io.h>
#include <util/delay.h>
int main (void)
  {
    DDRB=0xff;
    DDRD=0x00;
    PORTD=0xff;
    PORTB=0x00;
    while (1)
      {
       if (PIND=125)
         {
         PORTB=0x01;
         _delay_(500);
         PORTB=0x00;
         _delay_(500);
         PORTB=0x02;
         _delay_(500);
         PORTB=0x00;
        }
       else
         {
         PORTB=0x04;
         _delay_(500);
         PORTB=0x00;
         _delay_(500);
         PORTB=0x08;
         _delay_(500);
         PORTB=0x00;
        }
      }
  }
```

2) Сохраняем файл с программой как main.c.

Для того чтобы скомпилировать написанную программу, необходимо в программе Mfile, входящую в пакет программ WinAVR, создать Makefile и сохранить его в папке с основной программой main.c.

3) Открываем программу Mfile. Проходим по вкладкам Makefile→ Main file name и прописываем имя нашего компилируемого файла main.c без расширения. Выбираем модель микроконтроллера, пройдя по вкладкам Makefile→MCU type→ATtiny→attiny2313. Выбираем расширение hex для скомпилированного файла, пройдя по вкладкам Makefile→Output format→ihex. Сохраняем Makefile в папке с основной программой main.c.

4) В программе Programmer's Notepad производим компиляцию, пройдя по вкладкам Tools→ [WinAVR] Make All. Скомпилированный файл main.hex появится в той же папке с main.c.

5) Открываем программу PonyProg и производим ее настройку. Для начала выбираем тип программатора Setup→Interface Setup, выбираем LPT порт подключения программатора и тип программирования Avr ISP I/O. В главном окне программы выбираем тип ядра микроконтроллера AVR micro и модель ATtiny2313. Подключаем LPT-программатор к порту №1платы, а блок пита-ния в гнездо питания микроконтроллера. Пройдя по вкладкам Command→Read Program(FLASH), прочитаем flash память микроконтроллера, таким образом, проверив правильность выбранных настроек программатора и микроконтроллера. Если память прочитана, то можно начинать программировать микроконтроллер. Для этого загружаем файл с программой main.hex, пройдя по вкладкам File→Open Program(FLASH) File и записываем программу во flash-память микроконтроллера, пройдя по вкладкам Command→Write Program (FLASH).

6) Отключаем программатор от порта №1 платы. Перезагружаем микроконтроллер кнопкой Reset. Проверяем, что программа работает правильно.

Задание: на основе теоретических знаний и примеров программ, изложенных в лабораторной работе, написать программы, осуществляющие:

1) Непрерывное последовательное переключение светодиодов (в виде змейки), подключенных к порту PORTB микроконтроллера.

2) Непрерывное последовательное переключение светодиодов, подключенных к порту PORTB микроконтроллера, которое при нажатии и удерживании кнопки D0, меняет направление переключения на обратное.

3) При нажатии и удерживании кнопок, подключенных к D регистру выводов, светятся соответствующие номера светодиодов. При отпускании кнопки соответствующий светодиод выключается.

Контрольные вопросы

1. Назовите основные структурные блоки микроконтроллера и их функциональное назначение?

2. Как осуществляется тактирование микроконтроллера и какие источники при этом могут использоваться?

3. Из каких функциональных блоков состоит процессор микроконтроллера? Назовите функцию каждого блока.

4. Какие устройства микроконтроллера предназначены для хранения кодов программ?

5. Какие периферийные устройства входят в состав микроконтроллеров?

6. Назовите функциональное назначение fuse и lock битов. В чем их отличия?

7. Какие языки программирования могут быть использованы для написания программы микроконтроллера?

Рекомендуемая литература

1 Гребнев В. В. Микроконтроллеры семейства AVR фирмы Atmel / / В. В. Гребнев. – М.: ИП РадиоСофт, 2002 – 176 с: ил. С. 9−23.

2 Евстифеев А. В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя / А. В. Евстифеев. – М: Издательский дом «Додека-XXI», 2007. − 432 с.: ил. С. 9−77.

3 ATTINY2313 – 8-bit AVR Microcontroller with 2K Bytes In-System Programmable Flash – ATMEL Corporation. – Сайт документации на электронные компоненты. – (Англ.). – URL: http://www.alldatasheet.com/ /datasheet-pdf/pdf/80317/ATMEL/ATTINY2313.html

4 Белов А. В. Микроконтроллеры AVR в радиолюбительской практике. – СПб.: Наука и Техника, 2007. – 352 с.: ил.

5 Белов А. В. Самоучитель по микропроцессорной технике. – СПб.: Наука и Техника, 2003. – 224 с.: ил.

6 Белов А. В. Создаем устройства на микроконтроллерах. – СПб.: Наука и Техника, 2007. – 304 с.: ил.

Лабораторная работа №2 «Арифметико-логические операции над регистрами микроконтроллера»

Цель: научиться производить арифметико-логические операции с использованием регистров микроконтроллера.

Задачи: изучить АЛУ микроконтроллера, регистры ввода/вывода, внутренние регистры и арифметико-логические операции, которые можно к ним применять.

Приборы и принадлежности: плата на основе микроконтроллера ATtiny2313, Datasheet к микроконтроллеру ATtiny2313, LPT- программатор, ПК с установленными PonyProg и пакетом программ WinAVR.

Краткие теоретические сведения

Регистры микроконтроллера представляют собой устройства, используемые для хранения n-разрядных двоичных чисел. У микроконтроллера ATtiny2313 регистровая память включает 96 8-разрядных регистра, из которых 32 регистра общего назначения (РОН) и 64 регистра ввода/вывода (РВВ). РОН служат для хранения промежуточных результатов вычислений процессора. РВВ делятся на 2 группы: служебные регистры для настройки микроконтроллера и регистры для настройки периферийных устройств: таймеров/счетчиков, интерфейсов, портов ввода/вывода и других.

Для выполнения арифметико-логических операций над регистрами служит арифметико-логическое устройство микроконтроллера. Операнды поступают из регистров общего назначения. Каждая команда в качестве операндов использует либо содержимое двух разных регистров общего назначения, либо содержимое регистра и константу. Результат вычислений также помещается в один из регистров.

Арифметические операции производятся только над регистрами общего назначения и выполняются за один такт, кроме операции умножения, требующей 2 такта. Логические операции производятся между регистрами общего назначения, либо между регистром и константой за один такт, а результат сохраняется в РОН.

Микроконтроллер также может выполнять побитовые операции и операции сдвига. Синтаксис некоторых операций в WinAVR представлен в таблице 1

Таблица 1 – Синтаксис операций в WinAVR

Операция	Обозначение
Сложения	+
Вычитание	-
Инкрементирование	++
Декрементирование	--

Умножение	*
Деление	/
Логическое «И»	&&
Логическое «ИЛИ»	||
Логическое «НЕ»	!
Побитовое «И»	&
Побитовое «ИЛИ»	|
Побитовое «НЕ»	~
Побитовое исключающее «ИЛИ»	^
Побитовый сдвиг вправо	>>
Побитовый сдвиг влево	<<

Если требуется произвести операции сложения, вычитания, умножения над 2-мя регистрами и записать результат в третий регистр, необходимо записать соответственно: PORTC=PORTB+PORTA, PORTC=PORTB-PORTA, PORTC=PORTB*PORTA.

Операция инкрементирования над регистром увеличит его значение на 1, а декрементирования - уменьшит. Запись «++PORTC» эквивалентна «PORTC=PORTC+1», а «--PORTC» - «PORTC=PORTC-1».

Результат логических операций над регистрами может быть истинным и равным 1 или ложным и равным 0. Обычно это используют в связке с операторами if-else или while. К примеру, регистр PORTC будет хранить число 7, если значения регистров PORTA и PORTB отличны от нуля. В противном случае он будет содержать число 1:

```
if (PORTA&&PORTB)
  {
   PORTC=7;
  }
else
  {
   PORTC=1;
  }
```

Следующая запись означает, что значение регистра PORTC будет инкрементироваться до тех пор, пока все биты регистра PORTB равны 0.

```
while (!PORTB)
  {
   ++PORTC;
  }
```

Операция побитового «НЕ» инвентирует каждый бит регистра, если PORTB=0b10111011, то ~PORTB=0b01000100.

Операции сдвига вправо и влево эквивалентны соответственно умножению и делению на 2^n, где n – число разрядов, на которое сдвигают регистр. Например, если PORTA=0b11101001, то PORTA<<1=0b11010010 и PORTA>>1=0b01110100.

Побитовые операции часто используются при программировании микроконтроллеров. Например, если нужно установить шестой бит регистра PORTC, а значения остальных бит не изменить, используют запись: PORTC=PORTC|(1<<6), что эквивалентно PORTC= =PORTC|0b01000000. Если нужно установить пятый бит регистра, при этом остальные биты очистить (обнулить): PORTC=PORTC&(1<<5), что эквивалентно PORTC= PORTC&0b00100000. Сокращенная запись в двух случаях будет выглядеть так: PORTC|=(1<<6) и PORTC&=(1<<5). Соответственно, для того, чтобы проверить установлен ли n-бит регистра PORTB, пишем if (PINB&(1<<n)==(1<<n)) {} или сокращенно if (PINB&(1<<n)){}.

В библиотеке микроконтроллера ATtiny2313 iotn2313.h с помощью оператора #define каждому биту любого регистра присвоено число, соответствующее его расположению в регистре. Например, для регистра PORTB:

```
#define PB0 0
#define PB1 1
#define PB2 2
#define PB3 3
#define PB4 4
#define PB5 5
#define PB6 6
#define PB7 7
```

Поэтому вместо установки n-бита, для удобства читаемости программы устанавливают биты PB0…PB7. Например, для установки бита PB5 регистра PORTB запишем PORTB|=(1<<PB5), для проверки установлен ли этот бит – if (PINB&(1<<PB5)){}.

Таблица 2 – Сравнение результатов вычислений при задании функции и макроопределения

Макроопределение	Функция

#define SQURE(x) (x)*(x)
int main (void)
{
int Z=2;
int Y=0;
Y = SQURE (++z);
}
	Int SQURE (int x);
	Int SQURE (int x);
	{
	Return (x)*(x);
	}
	Int Y;
	Int Z=2;
	Y= SQURE (++z);

Y=3*4=12	Y=3*3=9

Директива #define называется макроподстановкой. Если записать, к примеру, #define A 15, то WinAVR заменит идентификатор A, в какой бы части программы он не находился, на число 15. Если ниже записать #define B (A+20), то идентификатор B заменится на (15+20). Если использовать макроподстановку #define X(a,b,c) ((a)*(b)-(c)), а в программе прописать функцию Y=X(k+m, k-m, n), то она будет заменена на Y=((k+m)*(k-m)-(n)).

Стоит отметить, что макроопределения иногда используются вместо определений функций, обычно из соображений эффективности. Но следует помнить, что препроцессор может лишь автоматически заменять одну строку на другую, не разбираясь, зачем это нужно.

В отличие от параметра функции, параметр макроопределения вычисляется при каждом вхождении в макроопределение (Табл. 2)

В качестве примера рассмотрим программу «двоичного калькулятора», осуществляющего сложение, вычитание, умножение и деление чисел. Кнопка D0 будет использоваться для ввода первого операнда, кнопка D1 – для ввода второго. Значение вводимых операндов будет отображаться светодиодными индикаторами. При нажатии кнопки D2, D3, D4, D5 соответственно будут производиться операции сложения, вычитания, умножения и деления над операндами и вывод результата на светодиодные индикаторы. Сброс калькулятора осуществляется при нажатии кнопки D6. Программа имеет следующий вид:

```
#include <avr/io.h> //подключаем библиотеку ATtiny2313
#include <Util/delay.h> //подключаем библиотеку функции задержки
int a,b=0; //инициализируем целочисленные переменные a,b и присваиваем им значение 0
int main (void) //главная функция программы
  {
   DDRB=0XFF; //порт B как выход (диоды)
   DDRD=0X00; //порт D как вход (кнопки)
   PORTB=0X00; //обнуляем порт B
   PORTD=0XFF; //осуществляем подтяжку внутреннего резистора
   while (1) //задаем бесконечный цикл
     {
      while (~PIND&(1<<PD0)) //пока кнопка D0 нажата
        {
         a++; //инкрементируем значение переменной a
         _delay_ms (300); //ждем 300 мс
         PORTB=a; //записываем в регистр PORTB значение переменной a-первого операнда
         _delay_ms (300);
        }
```

```
            while (~PIND&(1<<PD1)) //аналогично
              {
               b++;
               _delay_ms (300);
               PORTB=b;
               _delay_ms (300);
              }
            if (~PIND&(1<<PD2)) //если нажата кнопка D2
              {
               PORTB=a+b; //вывести на светодиодную панель результат суммы операндов
              }
            if (~PIND&(1<<PD3)) //если нажата кнопка D3
              {
               PORTB=a-b; //вывести на светодиодную панель результат разности операндов
              }
            if (~PIND&(1<<PD4)) //если нажата кнопка D4
              {
               PORTB=a*b; //вывести на светодиодную панель результат умножения операндов
              }
            if (~PIND&(1<<PD5)) //если нажата кнопка D5
              {
               PORTB=a/b; //вывести на светодиодную панель результат деления операндов
              }
            if (~PIND&(1<<PD6)) //если нажата кнопка D6
              {
               a=0; //обнуляем операнд a
               b=0; //обнуляем операнд b
               PORTB=0x00; //обнуляем регистр PORTB
              }
          }
      }
```

Задание: на основе теоретических знаний и примеров программ, изложенных в описании теории лабораторной работы, написать программу, осуществляющую операции сложения, вычитания, умножения, деления над регистрами. При этом подключенные к выводам микроконтроллера кнопки D0, D1, D2 используются для ввода 3-битных чисел, значение вводимого числа должно отображаться на диодной панели. Ввод выбранного числа осуществляется с помощью кнопки D5. Кнопкам D3, D4 присвоить операцию сложения и умножения. При нажатии кнопок D3 или D4 на диодной панели выводится результат соответствующей операции над двумя вводимыми числами.

Контрольные вопросы

1. Какие функции выполняет АЛУ микроконтроллера?
2. Какие виды регистров имеются в микроконтроллере и какое их функциональное назначение?
3. Сколько тактов требуется микроконтроллеру на выполнение арифметических операций сложения, умножения и вычитания?
4. Над какими регистрами микроконтроллера возможны арифметические операции?
5. Сколько тактов требуется микроконтроллеру на выполнение логических операций?
6. Над какими регистрами микроконтроллера возможны логические операции?
7. Назовите основные арифметические и логические операции.
8. Для чего служит макроподстановка #define?

Рекомендуемая литература

1 Гребнев В. В. Микроконтроллеры семейства AVR фирмы Atmel / / В. В. Гребнев. – М.: ИП РадиоСофт, 2002 – 176 с: ил. С. 24−36.

2 Евстифеев А. В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя / А. В. Евстифеев. – М: Издательский дом «Додека-XXI», 2007. − 432 с.: ил. С. 80−92.

3 ATTINY2313 – 8-bit AVR Microcontroller with 2K Bytes In-System Programmable Flash – ATMEL Corporation. – Сайт документации на электронные компоненты. – (Англ.). – URL: http://www.alldatasheet.com/ /datasheet-pdf/pdf/80317/ATMEL/ATTINY2313.html

4 Белов А. В. Микроконтроллеры AVR в радиолюбительской практике. – СПб.: Наука и Техника, 2007. – 352 с.: ил.

5 Белов А. В. Самоучитель по микропроцессорной технике. – СПб.: Наука и Техника, 2003. – 224 с.: ил.

6 Белов А. В. Создаем устройства на микроконтроллерах. – СПб.: Наука и Техника, 2007. – 304 с.: ил.

Лабораторная работа №3 «Изучение прерываний микроконтроллеров»

Цель: научиться использовать программные прерывания по переполнению таймеров/счетчиков T0 и T1, а также настройку и управление сигналами тактирования микроконтроллеров.

Задачи: изучить виды прерываний микроконтроллера, их принцип действия, векторы прерывания, регистры управления прерываниями SREG, GIMSK, работу 8-битного таймера/счетчика T0, 16-битного таймера/счетчика T1, регистры таймеров/счетчиков T0 и T1: TCCR0A, TCCR0B, TCNT0, TCCR1A,TCCR1B,TCNT1, и регистры управления прерываниями TIMSK и флагов прерываний TIFR. Изучить принципы тактирования микроконтроллера и задания частот тактирования.

Приборы и принадлежности: плата на основе микроконтроллера ATtiny2313, Datasheet к микроконтроллеру ATtiny2313, LPT- программатор, ПК с установленными PonyProg и пакетом программ WinAVR.

Краткие теоретические сведения

Прерывание прекращает нормальный ход программы для выполнения приоритетной задачи, определяемой внутренним или внешним событием микроконтроллера. При возникновении прерывания микроконтроллер сохраняет в стеке содержимое счетчика команд PC и загружает в него адрес соответствующего вектора прерывания. По этому адресу, как правило, находится команда безусловного перехода к подпрограмме обработки прерывания. Адреса векторов прерываний, их приоритетность для микроконтроллера ATtiny2313 приведены в таблице 3.

Таблица 3 – Прерывания микроконтроллера ATtiny2313

Номер вектора	Адрес перехода	Источник	Описание прерывания
1	0x0000	RESET	Внешний сброс, сброс при включении питания, сброс по срабатыванию охранного таймера
2	0x0001	INTO	Внешний запрос на прерывание по входу INTO
3	0x0002	INT1	Внешний запрос на прерывание по входу INT1
4	0x0003	TIMER1 CAPT	Прерывание по захвату таймера/счетчика 1
5	0x0004	TIMER1 COMPA	Прерывание по совпадению таймера/счетчика 1. Канал A
6	0x0005	T1MER1 OVF	Прерывание по переполнению таймера/счетчика 1
7	0x0006	TIMER0 OVF	Прерывание по переполнению таймера/счетчика 0
8	0x0007	USART0 RX	USARTO, прием завершен
9	0x0008	USART0 UDRE	USARTO буфер данных пуст
10	0x0009	USART0 TX	USARTO, передача завершена
11	0x000A	ANALOG COMP	Прерывание от аналогового компаратора
12	0x000B	PCINT	Прерывание по изменению на любом из выводов
13	0x000C	TIMER1 COMPB	Прерывание по совпадению таймера/счетчика 1. Канал B
14	0x000D	TIMER0 COMPA	Прерывание по совпадению таймера/счетчика 0. Канал B
15	0x000E	TIMER0 COMPB	Прерывание по совпадению таймера/счетчика 0. Канал A
16	0x000F	USI START	Прерывание по USI. Готовность к старту

17	0x0010	USI OVERFLOW	Прерывание по USI, Переполнение
18	0x0011	EE READY	Готовность EEPROM
19	0x0012	WDT OVERFLOW	Переполнение охранного таймера

Для глобального разрешения/запрещения прерываний предназначен флаг I регистра SREG. Для разрешения прерываний он должен быть установлен в 1, а для запрещения — сброшен в 0. При возникновении прерывания флаг I регистра SREG аппаратно сбрасывается, запрещая тем самым обработку следующих прерываний. Однако в подпрограмме обработки прерывания этот флаг можно снова установить в 1 для разрешения вложенных прерываний. При возврате из подпрограммы обработки прерывания флаг I устанавливается аппаратно.

При вызове подпрограмм обработки прерываний регистр состояния SREG не сохраняется. Поэтому пользователь должен самостоятельно запоминать содержимое этого регистра при входе в подпрограмму обработки прерывания (если это необходимо) и восстанавливать его значение.

Микроконтроллеры AVR поддерживают очередь прерываний, которая работает следующим образом: если условия генерации одного или более прерываний возникают в то время, когда флаг общего разрешения прерываний сброшен (все прерывания запрещены), соответствующие флаги устанавливаются в 1 и остаются в этом состоянии до установки флага общего разрешения прерываний. После разрешения прерываний выполняется их обработка в порядке приоритета.

Наименьшее время отклика для любого прерывания составляет 4-5 тактов. В течение этого времени происходит сохранение счетчика команд в стеке. В течение последующих 2-3 тактов выполняется команда перехода к подпрограмме обработки прерывания. Если прерывание произойдет во время выполнения команды, длящейся несколько циклов, то генерация прерывания произойдет только после выполнения этой команды. Возврат в основную программу занимает 4-5 такта, в течение которых происходит восстановление счетчика команд из стека. После выхода из прерывания процессор всегда выполняет одну команду основной программы, прежде чем обслужить любое отложенное прерывание. Таким образом, в сумме на подготовку и завершение прерываний микроконтроллер использует 10-12 тактов, что следует учитывать при написании программы.

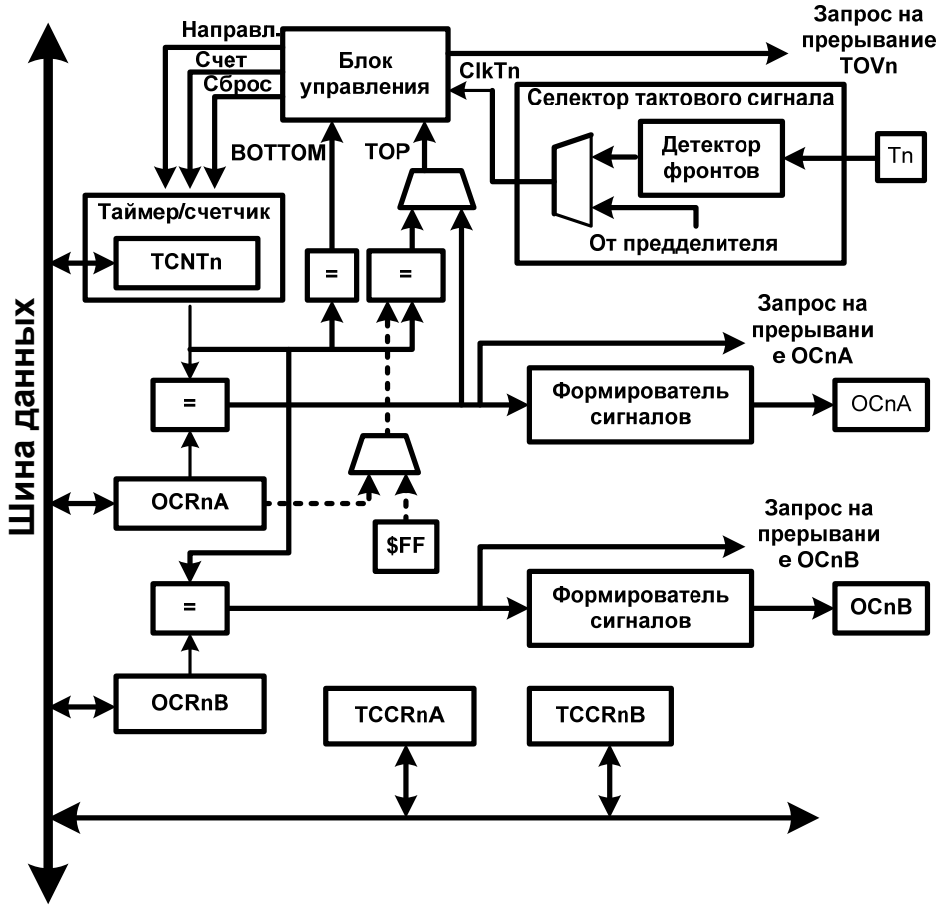


Рисунок 3 – Структурная схема таймера/счетчика T0 микроконтроллера ATtiny2313

Основными источниками прерываний являются различные периферийные устройства, одними из которых являются таймеры/счетчики.

Таймеры-счетчики предназначены для формирования запроса прерывания при истечении заданного интервала времени (режим таймера) или свершении заданного числа событий (режим счетчика). К дополнительным функциям таймеров/счетчиков относятся: функцию захвата, сравнения, широтно-импульсного модулятора, счета реального времени.

Микроконтроллер ATtiny2313 имеет 2 таймера/счетчика: 8-битный таймер/счетчик T0 и 16-битный T1. На рисунке 3 изображена структурная схема 8-битного таймера/счетчика T0 микроконтроллера ATtiny2313, который обладает функциями сравнения и двухканального 8-битного генератора ШИМ-сигналов. Для настройки таймера/счетчика предусмотрены

управляющие регистры TCCR0A и TCCR0B, формат которых представлен в таблице 4.

Таблица 4 – Формат управляющих регистров TCCR0A, TCCR0B таймера/ счетчика T0

Регистр TCCR0A								
№ бита	7	6	5	4	3	2	1	0
Названия битов	COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00
Чтение/запись	R/W	R/W	R/W	R/W	R	R	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0
Регистр TCCR0B								
№ бита	7	6	5	4	3	2	1	0
Названия битов	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00
чтение/запись	R/W	R/W	R	R	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Назначение бит следующее:

– COM0A1, COM0A0, COM0B1, COM0B0 служат для выбора режимов работы блоков сравнения A и B;

– WGM02, WGM01, WGM00 определяют режим работы таймера/счетчика;

– FOC0A, FOC0B служат для настройки принудительного изменения состояния выводов OC0A, OC0B;

– CS02, CS01, CS00 управляют тактовым сигналом таймера/счетчика.

Более подробно назначение управляющих бит и их значения изложены в Datasheet микроконтроллера.

Тактирование таймера/счетчика осуществляется либо от системного тактового сигнала, либо от внешнего источника, подключенного к выводу T0. 10-битный предделитель таймера/счетчика масштабирует системный тактовый сигнал CLK в 8, 64, 256 и 1024 раз в зависимости от состояния бит CS02…CS00. Настройка этих бит при выборе источника тактового сигнала приведена в таблице 5.

Таблица 5 – Выбор источника тактового сигнала таймера/счетчика T1

CS12	CS11	CS10	Источник тактового сигнала
0	0	0	Таймер/счетчик остановлен
0	0	1	CLK
0	1	0	CLK/8
0	1	1	CLK/64
1	0	0	CLK/256
1	0	1	CLK/1024
1	1	0	T0 по спадающему фронту
1	1	1	T0 по нарастающему фронту

Частота CLK определяется системным тактовым генератором микроконтроллера, выбор типа которого осуществляется через fuse-биты CKSEL3…CKSEL0 согласно таблице 6.

Таблица 6 – Настройка fuse-бит для выбора источника тактирования микроконтроллера

Значения битов CKSEL3…CKSEL0	Источник тактирования
0000	Внешний сигнал синхронизации
0010	Внутренний калибровочный RC-генератор с частотой 4 МГц
0100	Внутренний калибровочный RC-генератор с частотой 8 МГц
0110	Внутренний RC генератор с частотой 128 КГц
100x	Внешний кварцевый генератор с частотой 0,4…0,9 МГц
101x	Внешний кварцевый генератор с частотой 0,9…3,0 МГц
110x	Внешний кварцевый генератор с частотой 3,0…8,0 МГц
111x	Внешний кварцевый генератор с частотой более 8,0МГц

Кроме того, имеется возможность программного уменьшения частоты тактирования системного генератора, в результате которого замедляется работа всех периферийных устройств. Управляет предделителем регистр CLKPR, формат которого представлен в таблице 7.

Таблица 7 – Формат регистра CLKPR предделителя тактового сигнала

Регистр CLKPR								
№ бита	7	6	5	4	3	2	1	0

Названия битов	CPCE	-	-	-	CLKPS3	CLKPS2	CLKPS1	CLKPS0
Чтение/запись	R/W	R	R	R	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Установка бита CPCE в единицу разрешает делить тактовый сигнал, а биты CLKPS3…0, устанавливают коэффициент деления согласно таблице 8.

Таблица 8 – Выбор коэффициента деления предделителя тактового сигнала

CLKPS3	CLKPS2	CLKPS1	CLKPS0	Коэффициент деления
0	0	0	0	1
0	0	0	1	2
0	0	1	0	4
0	0	1	1	8
0	1	0	0	16
0	1	0	1	32
0	1	1	1	64
1	0	0	0	128
1	0	0	1	256

Начальное состояние битов CLKPS3…0 определяется fuse-битом деления частоты на 8 CKDIV. Если он запрограммирован (равен 0), то принимает значение 0011, в противном случае – 0000.

Чтобы записать регистр CLKPR необходимо выполнить 2 действия:

1) Записать 1 в бит CPCE, 0 – в биты CLKPS3…0;
2) В течение четырех тактов записать требуемое значение битов CLKPS3…0, бит CPCE при этом должен быть сброшен в 0.

Таймер/счетчик T1 отличается от T0 разрядностью счетного регистра (TCNT1 – шестнадцатиразрядный и состоит из 2-х: старшего регистра TCNT1H и младшего TCNT1L) и наличием функции захвата.

Таймеры/счетчики T0 и T1 генерирует прерывание по следующим событиям: по переполнению регистра TCNTn, по совпадению A значений регистров TCNTn и OCRnA, по совпадению B значений регистров TCNTn и OCR0B, где n соответствует номеру таймера/счетчика.

Для разрешения прерываний от таймеров/счетчиков служит регистр TIMSK, а для индикации наступления прерывания - регистр состояния TIFR. Форматы этих регистров представлены в таблицах 9 и 10 соответственно.

Таблица 9 – Формат регистра TIMSK разрешения прерываний от таймеров/счетчиков T0, T1

№ бита	7	6	5	4	3	2	1	0
Названия битов	TOIE1	OCIE1A	OCIE1B	-	ICIE1	OCIE0B	TOIE0	OCIE0A
Чтение/запись	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

Назначение бит регистра TIMSK следующее:

– TOIEn разрешает прерывание по событию «переполнение»;

– OCIEnA(OCIEnB) разрешает прерывание по событию «совпадение A(B)»;

– ICIE1 разрешает прерывание по событию «захват».

Таблица 10 – Формат регистра флагов прерываний TIFR таймеров/ счетчиков T0,T1

№ бита	7	6	5	4	3	2	1	0
Названия битов	TOV1	OCF1A	OCF1B	–	ICF1	OCF0B	TOV0	OCF0A
Чтение/запись	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
Начальное значение	0	0	0	0	0	0	0	0

При возникновении прерывания устанавливается в 1 соответствующий этому прерыванию флаг регистра TIFR. При запуске подпрограммы обработки прерывания он аппаратно сбрасывается в 0. Программно сбросить его в ноль можно, записав в него 1.

Так как основной функцией таймеров/счетчиков является отсчет времени, напишем программу, реализующую мигание светодиода PB6 с периодом 1 с, из которого 0.5с он должен гореть и 0.5с не гореть. Будем использовать прерывание по переполнению таймера/счетчика T1. Подпрограмма обработки прерываний должна инвентировать состояние вывода PB6 и вызываться каждые 0.5 с, то есть таймер/счетчик T1 должен переполняться за 0.5 секунд. С учетом того, что частота кварцевого резонатора CLK равна 8Мгц, за 1 с таймер/счетчик T1 будет переполняться 8000000/65536=122 раза, при этом светодиод будет мигать с частотой 61 Гц, вместо требуемой частоты 1 Гц. Необходимо использовать предделитель таймера/счетчика и понизить частоту CLK в 64 раза. Теперь диод будет мигать с частотой 61/64=0,954 Гц. Уменьшим число отсчетов таймера/счетчика в 0,954 раз: 65536/0,954=62475. То есть до переполнения он должен сделать 62475 отсчетов, а начальное значение счетного регистра

должно равняться 65536-62475=3061 или 0BF5 в шестнадцатеричной системе.

Программа в WinAVR будет выглядеть так:

```
#include <avr/io.h> /*Подключаем библиотеку микроконтроллера Attiny2313*/
#include <avr/interrupt.h> /*Подключаем библиотеку прерываний*/
ISR(TIMER1_OVF_vect) /*запускаем функцию прерывания по переполнению таймера/счетчика T1*/
  {
   PORTB=~PORTB&(0x40); /*инвентируем значение на выводе PB6*/
   TCNT1H=0x0B;
   TCNT1L=0xF5; /*устанавливаем значение 0BF5 счетного регистра  после возникновения прерывания*/
  }
int main(void)
  {
   DDRB=0xff; //порт B устанавливаем как выход
   PORTB=0x00; //обнуляем регистр PORTB
   TCNT1H=0x0B;
   TCNT1L=0xF5; /*устанавливаем значение 0BF5 счетного регистра  до возникновения прерывания*/
   TCCR1A=0x00; //нормальный режим работы таймера
   TCCR1B=(1<<CS10)|(1<<CS11); //clk/64
   TIMSK=(1<<TOIE1); //разрешение прерывания по переполнению
   sei(); //разрешение глобального прерывания (SREG7=1)
   while (1){} //запускаем бесконечный цикл, чтобы микроконтроллеру было что выполнять после прерывания
  }
```

Задание: на основе теоретических знаний и примеров программ, изложенных в лабораторной работе, написать программы, осуществляющие:

1) Мигание светодиода, подключенного к выводу PORTB0 по переполнению таймера/счетчика с периодом 4 с. Затем изменить период до 8 с.
2) Запрограммировать остановку таймера/счетчика при нажатии на кнопку, подключенную к выводу PORTD0, и возобновление его работы при повторном нажатии кнопки.

Контрольные вопросы

1. Что такое прерывания микроконтроллера и для чего они применяются?
2. Назовите основные источники прерываний.
3. Исходя из чего определяется очередность обработки прерываний при возникновении нескольких прерываний одновременно?
4. Какой регистр предназначен для глобального разрешения/запрещения прерываний?

5. Какое функциональное назначение имеют таймеры/счетчики микроконтроллера?

6. Какие регистры служат для управления режимами работы таймер/счетчиков?

7. Какими способами можно изменить частоту переполнения таймер/счетчиков?

8. Настройку каких регистров необходимо произвести для разрешения прерываний от таймер/счетчиков?

Рекомендуемая литература

1 Гребнев В. В. Микроконтроллеры семейства AVR фирмы Atmel / / В. В. Гребнев. – М.: ИП РадиоСофт, 2002 – 176 с: ил. С. 80−84.

2 Евстифеев А. В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя / А. В. Евстифеев. – М: Издательский дом «Додека-XXI», 2007. − 432 с.: ил. С. 195−207.

3 ATTINY2313 – 8-bit AVR Microcontroller with 2K Bytes In-System Programmable Flash – ATMEL Corporation. – Сайт документации на электронные компоненты. – (Англ.). – URL: http://www.alldatasheet.com/ /datasheet-pdf/pdf/80317/ATMEL/ATTINY2313.html

4 Белов А. В. Микроконтроллеры AVR в радиолюбительской практике. – СПб.: Наука и Техника, 2007. – 352 с.: ил.

5 Белов А. В. Самоучитель по микропроцессорной технике. – СПб.: Наука и Техника, 2003. – 224 с.: ил.

6 Белов А. В. Создаем устройства на микроконтроллерах. – СПб.: Наука и Техника, 2007. – 304 с.: ил.

Лабораторная работа №4 «Изучение сопряжения микроконтроллера с ЖКИ и вывод символьной информации»

Цель: научиться использовать ЖКИ совместно с микроконтроллером.

Задачи: изучить устройство работы и обмен данными с 16-символьным ЖКИ, его инициализацию, вывод символов на экран.

Приборы и принадлежности: плата на основе микроконтроллера ATtiny2313, Datasheet к микроконтроллеру ATtiny2313, LPT- программатор, ПК с установленными PonyProg и пакетом программ WinAVR, LCD WH0802A.

Краткие теоретические сведения

Жидкокристаллические индикаторы (ЖКИ) являются одними из основных средств вывода информации для современных цифровых систем. Представляют собой недорогое и удобное решение, позволяющее сэкономить время и ресурсы при разработке новых устройств. Обеспечивают отображение большого объема информации при хорошей различимости и низком энергопотреблении, благодаря чему широко используются в измерительных приборах, медицинском оборудовании, промышленном оборудовании, информационных системах, аппаратуре с автономным питанием.

Структурная схема символьного ЖКИ представлена на рисунке 4. Основу составляет специализированный контроллер, обычно выполненный в виде одной или двух «микросхем-капелек», реже − в виде фирменной SMD-микросхемы. Контроллер синхронизируется внутренним RC-генератором G1, имеющим частоту 250 ±50 кГц, и управляет интерфейсом, а драйвер "зажигает" сегменты. Напряжение подсветки подается через выводы A и K на светодиоды, которые освещают ЖК-панель с торца или обратной стороны корпуса. Светодиоды включены матрицей и соединены параллельно-последовательно. В связи с этим напряжение подсветки довольно высокое 4,0...4,2 В.

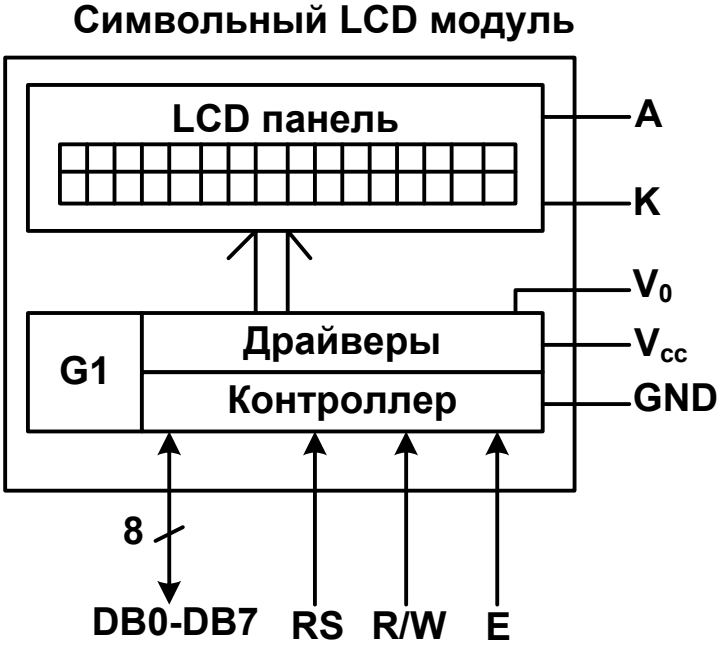


Рисунок 4 – Структурная схема символьного ЖКИ

Электрический интерфейс ЖКИ состоит из 3-х шин: шины данных (DB0…DB7), шины управления (RS, R/W, E) и шины питания (Vss, Vdd,

Vee). Выполняемые функции каждого вывода ЖКИ приведено в таблице 11.

Таблица 11 – Функции, выполняемые каждым выводом ЖКИ.

№ вывода	Обозначение	Выполняемая функция
1	Vee	Общий провод
2	Vcc	Питание +5В
3	Vdd	Напряжение фокусировки
4	RS	«0»- запись команд «1»-запись данных
5	R/W	«0» - запись в ЖКИ «1» - чтение из ЖКИ
6	E	Перепад уровня из «1» в «0» - ввод данных
7-14	DB0-DB7	Двунаправленная шина данных

В лабораторной работе используется 2-строчный 8-символьный ЖКИ WH0802A-NGA-CT, который подключается к микроконтроллеру ATtiny2313 согласно схеме, представленной на рисунке 5. Для передачи данных используется 4-битная шина DB4-DB7, то есть байт данных будет передаваться в 2 этапа: сначала старший полубайт, затем − младший.

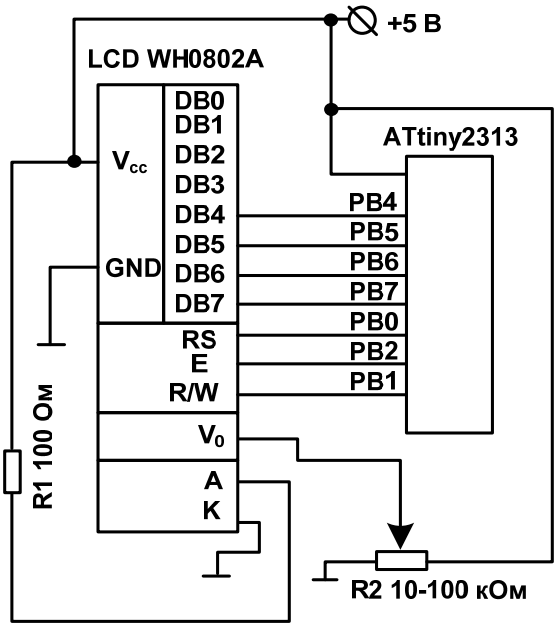


Рисунок 5 – Схема подключения микроконтроллера ATtiny2313 к ЖКИ

Микроконтроллер ATtiny2313 управляет работой WH0802A через систему команд. Все команды можно посмотреть в Datasheet ЖКИ. Основные приведены в таблице 12.

Таблица 12 – Основные команды WH0802A

Команда ЖКИ	Hex-код
Очистка дисплея	0x01
Возврат курсора в начало	0x02
Сдвиг курсора влево	0x04
Сдвиг курсора вправо	0x06
Выключение дисплея	0x08
Выключение курсора	0x0C
Интерфейс 4 бита, 2 строки	0x28
Установка позиции курсора	0x80 – 0xC7

Для отправки команды ЖКИ необходимо, чтобы на выводах RS и R/W было напряжение низкого уровня. Для отправки данных требуется, чтобы на выводе RS было напряжение высокого уровня, а на выводе R/W – низкого. В обоих случаях разрешение записи осуществляется изменением состояния вывода E с высокого напряжения на низкое.

Функция отправки команды в ЖКИ на языке СИ может выглядеть так:

```
void LCDsendCommand(uint8_t cmd)
  {
   LDP=(cmd&0xF0); //загружаем в порт данных старший полубайт команды
   LCP|=1<<LCD_E; //отправляем старший полубайт команды (E=1)
   _delay_ms(1);
   LCP&=~(1<<LCD_E); //E=0
   _delay_ms(1);
   LDP=((cmd&0x0F)<<4); //загружаем в порт данных младший полубайт команды
   LCP|=1<<LCD_E; //отправляем старший полубайт команды (E=1)
   _delay_ms(1);
   LCP&=~(1<<LCD_E); //E=0
   _delay_ms(1);
  }
```

Аргументом функции является 8-битная команда cmd, отправка которой происходит по полубайтам. Обязательным условием является передача данных контроллеру ЖКИ с паузами не менее 40 мкс, чтобы контроллер ЖКИ успел выгрузить полубайт и загрузить новый, так как частота работы ATtiny2313 в несколько раз превышает частоту работы контроллера ЖКИ; для этой цели в примере программы используется функция задержки _delay_ms().

Функция отправки символа в ЖКИ на языке СИ может выглядеть так:

```
void LCDsendChar(uint8_t ch)
  {
   LDP=(ch&0xF0); //загружаем в порт данных старший полубайт символа
   LCP|=1<<LCD_RS;  //настраиваем порт на отправку данных (RS=1)
   LCP|=1<<LCD_E; //отправляем старший полубайт символа(E=1)
   _delay_ms(1);
   LCP&=~(1<<LCD_E); //E=0
   LCP&=~(1<<LCD_RS); //RS=0
   _delay_ms(1);
   LDP=((ch&0x0F)<<4); //загружаем в порт данных старший полубайт символа
   LCP|=1<<LCD_RS; //настраиваем порт на отправку данных (RS=1)
   LCP|=1<<LCD_E; //отправляем старший полубайт символа(E=1)
_delay_ms(1);
   LCP&=~(1<<LCD_E); //E=0
   LCP&=~(1<<LCD_RS); // RS=1
    _delay_ms(1);
}
```

Как и в предыдущем случае передача байта символа осуществляется по полубайтам.

Перед использованием ЖКИ, необходимо произвести его инициализацию – начальную настройку, суть которой состоит в отправке определенного набора команд. Согласно Datasheet, алгоритм инициализации ЖКИ WH0802A-NGA-CT для использования 4-х разрядной шины данных:

1) После включения питания, не ранее 40 мс инициализации микроконтроллера с ЖКИ, отправить по шине данных команду 0011 и подождать после этого не менее 39 мкс. Это время необходимо для корректного срабатывания программы контроллера ЖКИ.
2) Отправить в шину данных команду 0010.
3) Отправить в шину данных команду NF10. Если N=1, то будут задействованы 2 строки дисплея, N=0 – 1 строка. Если F=1, то формат выводимых на дисплей символов будет 5x11 пикселей, F=0 – 5x8. Снова сделать паузу не менее 39 мкс.
4) Повторить пункты 2 и 3.
5) Отправить в шину данных команду 0000.
6) Отправить в шину данных команду 1DCB. Если D=1, C=1 и B=1, то включены соответственно дисплей, курсор и функция мигания курсора. Если D=0, C=0 и B=0, то соответственно дисплей выключен, курсор не виден и не мигает. Сделать паузу не менее 37 мкс.
7) Отправить в шину данных команду 0000, затем 0001. Подождать не менее 1.53 мс.
8) Отправить в шину данных команду 0000, затем 01KZ. Если K=1, то направление движения курсора слева направо, если K=0 – в обратную сто-

рону. Если Z=1, то разрешен сдвиг экрана после полного заполнения символами, если Z=0 – запрещен. Сделать паузу не менее 1.53 мс перед тем, как посылать далее значения символов.

Функция инициализации WH0802A на языке СИ будет выглядеть следующим образом:

```
void LCDinit(void)
  {
   _delay_ms(40);
   LDP=0x00; //обнуляем порт данных
   LCP=0x00; //обнуляем порт управления
   LDDR|=1<<LCD_D7|1<<LCD_D6|1<<LCD_D5|1<<LCD_D4; //порт данных настраиваем на выход
   LCDR|=1<<LCD_E|1<<LCD_RW|1<<LCD_RS; //порт управления настраиваем на выход
   LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4; //загружаем команду 0011
   LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS; //отправляем команду 0011
   _delay_ms(1);
   LCP&=~(1<<LCD_E);
   _delay_ms(1);
   LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|0<<LCD_D4; //загружаем команду 0010
   LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS; //отправляем команду 0010
   _delay_ms(1);
   LCP&=~(1<<LCD_E);
   _delay_ms(1);
   LDP=1<<LCD_D7|1<<LCD_D6|1<<LCD_D5|0<<LCD_D4; //загружаем команду NF10
   LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;     // отправляем команду NF10
   _delay_ms(1);
   LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|0<<LCD_D4; //загружаем команду 0010
   LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS; //отправляем команду 0010
   _delay_ms(1);
   LCP&=~(1<<LCD_E);
   _delay_ms(1);
   LDP=1<<LCD_D7|1<<LCD_D6|1<<LCD_D5|0<<LCD_D4; //загружаем команду NF10
   LCP&=~(1<<LCD_E);
   _delay_ms(1);
   LCDsendCommand(0b00001111); //отправляем команду 00001DCB
   LCDsendCommand(0b00000001); //отправляем команду 00000001
   LCDsendCommand(0b00000111); //отправляем команду 000001KZ
  }
```

Вышеприведенные функции являются основными инициализации, отправки команд и отправки данных для ЖКИ и, пользуясь только ими, можно в полной мере раскрыть все его возможности. Однако, не всегда удобно запоминать команды, адреса ячеек памяти ЖКИ и прочее в цифровом коде. Для упрощения используют библиотеки ЖКИ, обычно это файлы с расширениями «.c» и «.h». В файлах расширения «.c» обычно располагают основные и дополнительные функции, которые определены в файлах с расширением «.h». Чтобы воспользоваться функцией, необходимо в

главной программе ее вызвать. К примеру, в библиотеке bibl.h определяем целочисленную функцию int function (int a), в bibl.c реализуем эту функцию int function (int a)=a*8, в главной же программе main.c ее вызываем c=function(a). Необходимые для выполнения лабораторной работы функции используемой библиотеки приведены в таблице 13.

Таблица 13 – Основные и дополнительные функции используемой библиотеки

Основные функции	
Обозначение	Предназначение
LCDinit(void)	Инициализирует ЖКИ
void LCDsendCommand(uint8_t)	Отправляет команду в ЖКИ
oid LCDsendChar(uint8_t)	Отправляет символ в ЖКИ
Дополнительные функции	
Обозначение	Предназначение
void LCDclr(void)	Очищает дисплей
void LCDhome(void)	Возвращает курсор в начало
void LCDstring(uint8_t*, uint8_t)	Выводит строку в ЖКИ
void LCDGotoXY(uint8_t, uint8_t)	Переводит курсор в ячейку с координатой (x,y)
void CopyStringtoLCD(const uint8_t*, uint8_t, uint8_t)	Выводит строку в ЖКИ с координаты (x,y)
void LCDdefinechar(const uint8_t *,uint8_t)	Записывает новый символ в память ЖКИ
void LCDshiftRight(uint8_t)	Смещает курсор вправо
void LCDshiftLeft(uint8_t)	Смещает курсор влево
void LCDcursorOn(void)	Отображает курсор
void LCDcursorOnBlink(void)	Включает мигание курсора
void LCDcursorOFF(void)	Скрывает курсор
void LCDblank(void)	Очищает ЖКИ без очистки памяти
void LCDvisible(void)	Включает отображение символов ЖКИ
void LCDcursorLeft(uint8_t)	Перемещает курсор влево на n позиций

Приведем пример программы, реализующей вывод строки «KubSU» на дисплей ЖКИ:

```
#include <avr/io.h> //Подключаем библиотеку для микроконтроллера ATtiny2313
#include <util/delay.h> //Подключаем библиотеку функции задержки
#include <lcd_lib.h> //Подключаем библиотеку для LCD
#include <lcd_lib.c> //Подключаем библиотеку для LCD
int main(void) //главная функция программы
   {
    DDRB=0xff; //порт B как выход
    PORTB=0x00; //обнуляем регистр порта B
```

```
  LCDinit(); //инициализируем LCD
  LCDclr(); //очищаем дисплей LCD
  LCDGotoXY(0, 0); //переведем курсор в ячейку с координатой (0,0)
  LCDsendChar('K'); //отправим символ «K» в LCD
  _delay_ms(500);  //подождем 500 мс
  LCDGotoXY(1, 0);  //переведем курсор в ячейку с координатой (1,0)
  _delay_ms(500); //подождем 500 мс
  LCDsendChar('u'); //отправим символ «u» в LCD
  _delay_ms(500); //подождем 500 мс
  LCDGotoXY(2, 0); //переведем курсор в ячейку с координатой (2,0)
  _delay_ms(500); //подождем 500 мс
  LCDsendChar('b'); //отправим символ «b» в LCD
  _delay_ms(500); //подождем 500 мс
  LCDGotoXY(3, 0); //переведем курсор в ячейку с координатой (3,0)
 _delay_ms(500); //подождем 500 мс
  LCDsendChar('S'); //отправим символ «S» в LCD
  _delay_ms(500); //подождем 500 мс
  LCDGotoXY(4, 0); //переведем курсор в ячейку с координатой (4,0)
  _delay_ms(500); //подождем 500 мс
  LCDsendChar('U'); //отправим символ «U» в LCD
  _delay_ms(500); //подождем 500 мс
  LCDGotoXY(5, 0); //переведем курсор в ячейку с координатой (5,0)
  _delay_ms(500); //подождем 500 мс
 }
```

Вышеприведенная программа занимает много места. Используя массивы символов, а также операторы цикла for и условия if, можно уменьшить ее размер и при этом выводить строки, состоящие из произвольного числа символов:

```
#include <avr/io.h> //Подключаем библиотеку для микроконтроллера ATtiny2313
#include <util/delay.h> // Подключаем библиотеку функции задержки
#include <lcd_lib.h> //Подключаем библиотеку для LCD
#include <lcd_lib.c> //Подключаем библиотеку для LCD
char moya_stroka[16]="Privet  Student!"; /*определяем массив символов и присваиваем каждому символу moy_stroka[n] значение*/
int a,b=0; //определяем целочисленные переменные a и b и присваиваем им нулевое значение
int main(void) //главная функция программы
  {
   DDRB=0xff; //порт B как выход
   PORTB=0x00; //обнуляем значение регистра порта B
   LCDinit(); //инициализируем LCD
   LCDclr(); //очищаем экран LCD
   for(int i=0;i<=15 ;i++) //задаем цикл с помощью оператора for
     {
      LCDGotoXY(a, b); //переводим курсор на координату (a,b)
      LCDsendChar(moya_stroka[i]); //записываем на дисплей i-символ
```

```
        _delay_ms(1000); //ждем 1000 мс
        a++; //инкрементируем координату a
        if (a>7) //если a будет больше 7, обнулим значение и перейдем на вторую строку
          {
           a=0;
           b=1;
          }
       }
    }
```

Используя ЖКИ, можно выводить результат вычислений над значениями в регистрах на дисплей. Однако следует помнить, что ЖКИ кодирует символы согласно таблице «ASCII» кода и символу 0 , например, соответствует десятичный код 48, а символу 9 – код 57. Пример программы «калькулятор» с отображением вводимых операндов, операций над ними и результата на ЖКИ приведен ниже:

```
#include <avr/io.h> //Подключаем библиотеку для микроконтроллера ATtiny2313
#include <util/delay.h> //Подключаем библиотеку функции задержки
#include <lcd_lib.h> //Подключаем библиотеку для LCD
#include <lcd_lib.c> //Подключаем библиотеку для LCD
int a,b=0; //инициализируем целочисленные переменные a,b и присваиваем им значение 0
int main (void) //главная функция программы
  {
   DDRB=0XFF; //порт B как выход (диоды)
   DDRD=0X00; //порт D как вход (кнопки)
   PORTB=0X00; //обнуляем порт B
   PORTD=0XFF; //осуществляем подтяжку внутреннего резистора
   LCDinit(); //инициализируем LCD
   LCDclr(); //очищаем экран LCD
   while (1) //задаем бесконечный цикл
     {
      while (~PIND&(1<<PD0)) //пока кнопка D0 нажата
        {
         a++; //инкрементируем значение переменной a
         if(a<10) //если a-односимвольное десятичное число
           {
            LCDGotoXY(0, 0);
            LCDsendChar(a+48); //выводим  односимвольное десятичное число a на
дисплей
           }
         if(a>=10) //если a – двусимвольное десятичное число
           {
            LCDGotoXY(0, 0);
            LCDsendChar(a/10+48); //выводим второй символ-разряд числа a на дисплей
            LCDGotoXY(1, 0);
            LCDsendChar(a%10+48); //выводим  первый символ-разряд числа a на дис-
плей
           }
         _delay_ms (300); //ждем 300 мс
        }
```

```
            while (~PIND&(1<<PD1)) //пока кнопка D1 нажата
              {
               b++; //инкрементируем значение переменной b
               if(b<10) //если b - односимвольное
                 {
                  LCDGotoXY(3, 0);
                  LCDsendChar(b+48); //выводим  односимвольное число b на дисплей
                 }
               if(b>=10)
                 {
                  LCDGotoXY(3, 0);
                  LCDsendChar(b/10+48); //выводим второй символ-разряд числа b на дисплей
                  LCDGotoXY(4, 0);
                  LCDsendChar(b%10+48); //выводим  первый символ-разряд числа b на дис-
плей
                 }
               _delay_ms (300); //ждем 300 мс
              }
            if (~PIND&(1<<PD2)) //если нажата кнопка D2
              {
               LCDGotoXY(2, 0);
               LCDsendChar(43); //выводим символ "+" на дисплей,код которого 43
               LCDGotoXY(5, 0);
               LCDsendChar(61); //выводим символ "=" на дисплей,код которого 61
               if(a+b<10) //если результат сложения a и b -односимвольное число
                 {
                  LCDGotoXY(6, 0);
                  LCDsendChar(a+b+48); //выводим результат сложения на дисплей
                 }
               if(a+b>=10) //если результат сложения a и b -двусимвольное число
                 {
                  LCDGotoXY(6, 0);
                  LCDsendChar((a+b)/10+48); //выводим первый разряд результата сложения
на дисплей
                  LCDGotoXY(7, 0);
                  LCDsendChar((a+b)%10+48); //выводим второй разряд результата сложения
на дисплей
                 }
              }
            if (~PIND&(1<<PD3)) //если нажата кнопка D3
              {
               LCDGotoXY(2, 0);
               LCDsendChar(45);
               LCDGotoXY(5, 0);
               LCDsendChar(61);
               if(a+b<10)
                 {
                  LCDGotoXY(6, 0);
                  LCDsendChar(a-b+48);
                 }
               if(a+b>=10)
                 {
```

```
            LCDGotoXY(6, 0);
            LCDsendChar((a-b)/10+48);
            LCDGotoXY(7, 0);
            LCDsendChar((a-b)%10+48);
           }
        }
      if (~PIND&(1<<PD4)) //если нажата кнопка D4
        {
         LCDGotoXY(2, 0);
         LCDsendChar(42);
         LCDGotoXY(5, 0);
         LCDsendChar(61);
         if(a*b<10)
           {
            LCDGotoXY(6, 0);
            LCDsendChar(a*b+48);
           }
         if(a*b>=10)
           {
            LCDGotoXY(6, 0);
            LCDsendChar((a*b)/10+48);
            LCDGotoXY(7, 0);
            LCDsendChar((a*b)%10+48);
           }
        }
      if (~PIND&(1<<PD5)) //если нажата кнопка D5
        {
         LCDGotoXY(2, 0);
         LCDsendChar(47);
         LCDGotoXY(5, 0);
         LCDsendChar(61);
         if(a+b<10)
           {
            LCDGotoXY(6, 0);
            LCDsendChar(a/b+48);
           }
         if(a+b>=10)
           {
            LCDGotoXY(6, 0);
            LCDsendChar((a/b)/10+48);
            LCDGotoXY(7, 0);
            LCDsendChar((a/b)%10+48);
           }
        }
    }
}
```

Кнопка D0 используется для ввода первого операнда, кнопка D1 – для ввода второго. Значение вводимых операндов будет отображаться на ЖКИ. При нажатии кнопок D2, D3, D4, D5 соответственно будут произво-

диться операции сложения, вычитания, умножения и деления над операндами и вывод операций и результата на ЖКИ.

Задание: на основе теоретических знаний и примеров программ, изложенных в краткой теории, а также примеров из лабораторной работы №3 написать программы, осуществляющие:

1) Вывод на дисплей надписей: «Hello world», «Слава КПСС».
2) Программа «Калькулятор» с использованием ЖКИ как устройства отображения информации. Калькулятор должен складывать, умножать, делить (с точностью до одного знака после запятой) и вычитать двузначные десятичные числа.

Контрольные вопросы

1. Для чего используют жидкокристаллические индикаторы?
2. Какова структура ЖКИ?
3. Какие выводы ЖКИ используются для ввода данных от микроконтроллера?
4. Для чего требуется производить инициализацию ЖКИ перед выводом информации?
5. Какую кодировку использует ЖКИ при выводе символов на экран?

Рекомендуемая литература

1 Гребнев В. В. Микроконтроллеры семейства AVR фирмы Atmel / / В. В. Гребнев. – М.: ИП РадиоСофт, 2002 – 176 с: ил. С. 85−90.

2 Евстифеев А. В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя / А. В. Евстифеев. – М: Издательский дом «Додека-XXI», 2007. − 432 с.: ил. С. 207−225.

3 ATTINY2313 – 8-bit AVR Microcontroller with 2K Bytes In-System Programmable Flash – ATMEL Corporation. – Сайт документации на электронные компоненты. – (Англ.). – URL: http://www.alldatasheet.com/ /datasheet-pdf/pdf/80317/ATMEL/ATTINY2313.html

4 Подключение LCD к микроконтроллерам AVR. – 2011. – Информационный сайт о микроконтроллерах и технологиях. – URL: http://radioparty.ru/prog-avr/program-c/258-lcd-avr-lesson1.

5 WH0802A – 8 X 2 CHARACTER – List of Unclassifed Manufacturers. – Сайт документации на электронные компоненты. – (Англ.). – URL: http://www.alldatasheet.com/ datasheet-pdf/pdf/86975/ETC/WH0802A.html

6 Белов А. В. Микроконтроллеры AVR в радиолюбительской практике. – СПб.: Наука и Техника, 2007. – 352 с.: ил.

7 Белов А. В. Самоучитель по микропроцессорной технике. – СПб.: Наука и Техника, 2003. – 224 с.: ил.

8 Белов А. В. Создаем устройства на микроконтроллерах. – СПб.: Наука и Техника, 2007. – 304 с.: ил.

Лабораторная работа №5 «Использование интерфейса USART микроконтроллера для приема/передачи данных»

Цель: научиться использовать интерфейс протокола USART микроконтроллера для двустороннего обмена данными с подключаемыми устройствами.

Задачи: изучить структуру интерфейса USART микроконтроллера, назначение функциональных блоков, принцип приема/передачи данных по интерфейсу USART, настройку скорости приема/передачи, формата кадра, функции регистров UCSRA, UCSRB, UCSRC, сопряжение интерфейса USART микроконтроллера с интерфейсом RS-232 компьютера, разобрать примеры программ по приему/передаче данных по интерфейсу USART.

Приборы и принадлежности: плата на основе микроконтроллера ATtiny2313, Datasheet к микроконтроллеру ATtiny2313, LPT- программатор, ПК с установленными PonyProg и пакетом программ WinAVR, устройство сопряжения микроконтроллера с ПК.

Краткие теоретические сведения

Модуль приемо-передатчика USART обеспечивает полнодуплексный обмен по последовательному каналу, при этом скорость передачи данных может варьироваться в довольно широких пределах. Длина посылки составляет от 5 до 9 битов. В состав USART входят схемы контроля и формирования бита четности, которые обнаруживают такие внештатные ситуации, как переполнение, ошибка кадрирования, неверный старт-бит. Для уменьшения вероятности сбоев в модулях также реализована такая полезная функция, как фильтрация помех. Для взаимодействия с программой предусмотрены три прерывания, запрос на генерацию которых формируется при наступлении следующих событий: «передача завершена», «регистр данных передатчика пуст» и «прием завершен».

Модуль состоит из трех основных частей: блока тактирования, блока передатчика и блока приемника (рис. 6). Блок тактирования модулей USART содержит схему синхронизации, которая используется при работе в синхронном режиме, и контроллер скорости передачи. Блок передатчика

включает одноуровневый буфер, сдвиговый регистр, схему формирования бита четности и схему управления. Блок приемника содержит схемы восстановления тактового сигнала и данных, схему контроля четности, двухуровневый буфер, сдвиговый регистр, а также схему управления.

Буферные регистры приемника и передатчика располагаются по одному адресу пространства ввода/вывода и обозначаются как регистр данных UDR. В этом регистре хранятся младшие 8 битов принимаемых и передаваемых данных. При чтении регистра UDR выполняется обращение к буферному регистру приемника, при записи − к буферному регистру передатчика.

В модулях USART буфер приемника является двухуровневым (FIFO-буфер), изменение состояния которого происходит при любом обращении к регистру UDR. В связи с этим не следует использовать регистр UDR в качестве операндов команд типа «чтение/модификация/ запись».

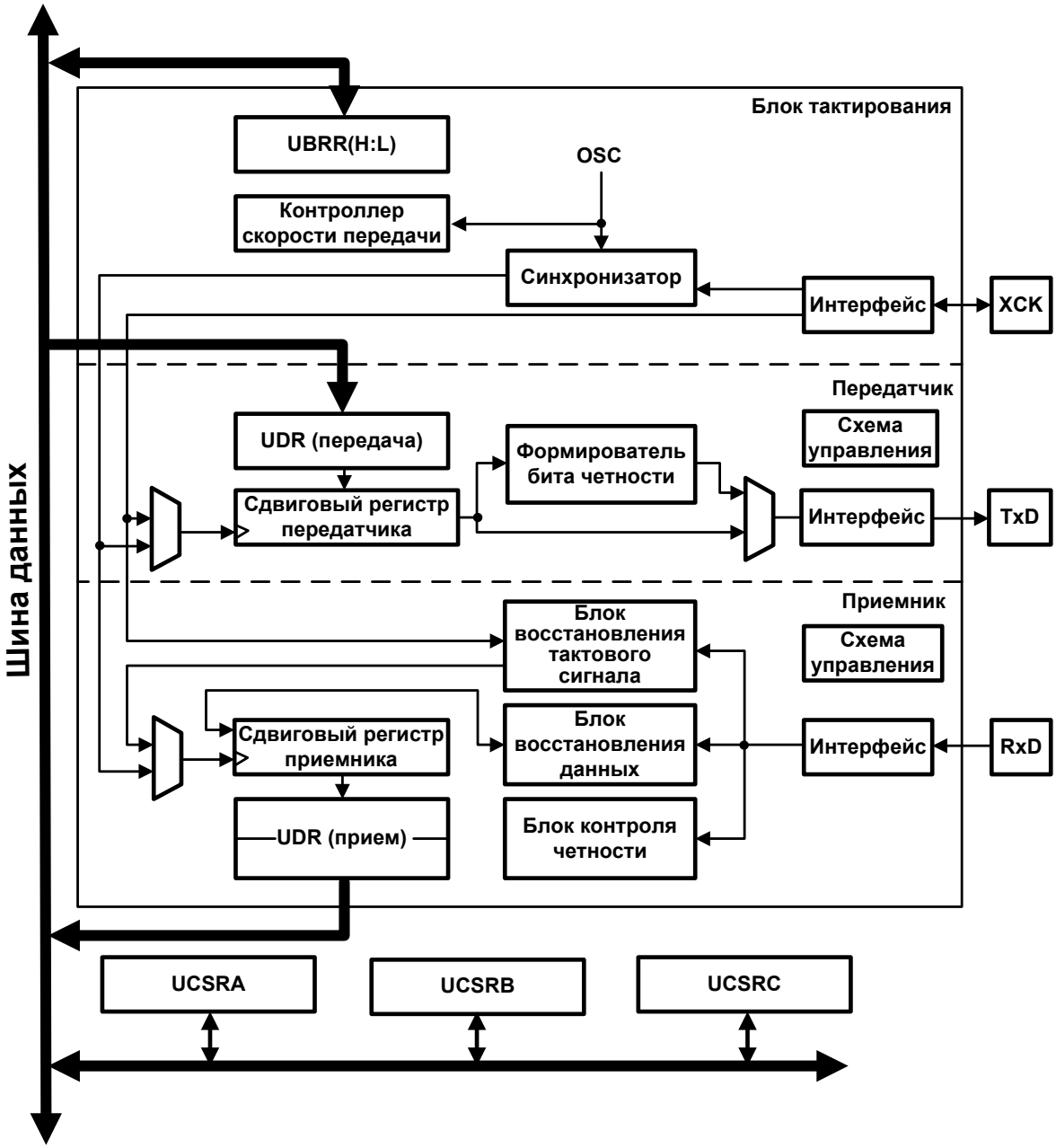


Рисунок 6 – Структура модуля приемо-передатчика USART

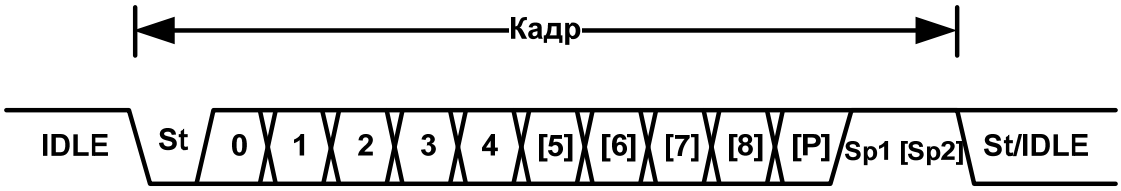
Для управления модулями USART используются три регистра: UCSRA, UCSRB и UCSRC. В асинхронном режиме, а также в синхронном режиме при работе в качестве ведущего скорость приема и передачи данных задается контроллером скорости передачи, работающим как делитель системного тактового сигнала с программируемым коэффициентом деления. Коэффициент определяется содержимым регистра контроллера

UBRR. В блок приемника сформированный сигнал поступает напрямую, а в блок передатчика − через дополнительный делитель, коэффициент деления которого (2, 8 или 16) зависит от режима работы модуля USART. Регистр UBRR является 12-битным и физически размещается в двух регистрах ввода/вывода.

При работе в асинхронном режиме скорость обмена определяется не только содержимым регистра UBRR, но и состоянием бита U2X регистра UCSRA. Если этот бит установлен в 1, коэффициент деления предделителя уменьшается в два раза, а скорость обмена соответственно удваивается. При работе в синхронном режиме этот бит должен быть сброшен. Скорость обмена данными модуля USART в асинхронном режиме составляет CLK/(16*(UBRR+1)), если бит U2X=0, и CLK/(8*(UBRR+1)), если U2X=1. В синхронном режиме скорость равна CLK/(2*(UBRR+1)).

При работе в синхронном режиме в качестве ведомого скорость приема и передачи определяется частотой сигнала, поступающего на вывод XCK. Частота этого сигнала должна быть меньше частоты тактирования микроконтроллера в 4 раза. Это ограничение связано с тем, что сигнал с вывода XCK сначала синхронизируется с тактовой частотой микроконтроллера, а затем проходит через детектор фронтов. Задержка сигнала при прохождении этих узлов равна двум тактам.

Передаваемый кадр начинается со старт-бита, за которым следует младший бит слова данных. После старшего бита слова данных следует один или два стоп-бита. Если включена схема формирования бита четности, он включается между старшим битом слова данных и первым стоп-битом (рис. 7).



St – старт-бит, всегда 0
(n) – биты данных
P – бит четности
Sp1 – стоп-бит, всегда 1
IDLE – нет обмена по линии RxD или TxD, должна быть 1

Рисунок 7 – Структура кадра данных интерфейса USART

Формат кадра определяется различными битами регистров UCSRB и UCSRC. В частности, размер слова данных определяется битами UCSZ2...UCSZ0.

Выбор количества стоп-битов осуществляется с помощью бита USBS регистра UCSRC. Если этот бит сброшен в 0, блок передатчика формирует 1 стоп-бит в конце посылки. В противном случае, если бит установлен в 1, блок передатчика формирует 2 стоп-бита. Следует отметить, что приемником второй стоп-бит игнорируется, и соответственно ошибки кадрирования выявляются только для первого стоп-бита.

Биты UPM1:UPM0 регистра UCSRC определяют функционирование схемы контроля четности модулей USART. Значение бита четности получается путем выполнения операции «Исключающее ИЛИ» над всеми битами передаваемого слова данных. Если используется проверка на нечетность, полученный результат инвертируется.

Работа передатчика разрешается установкой в 1 бита TXEN регистра UCSRB. При установке бита вывод TXD подключается к передатчику USART и начинает функционировать как выход независимо от установок регистров управления портом. Если используется синхронный режим работы, то переопределяется также функционирование вывода XCK.

Передача инициируется записью передаваемых данных в буферный регистр передатчика − регистр данных UDR. После этого данные пересылаются из регистра UDR в сдвиговый регистр передатчика. Одновременно, если используются 9-битные данные, значение бита TXB8 регистра UCSRB копируется в 9-й бит сдвигового регистра, при этом возможны два варианта: в первом, запись в регистр UDR осуществляется в тот момент, когда передатчик находится в состоянии ожидания (предыдущие данные уже переданы). В этом случае данные пересылаются в сдвиговый регистр сразу же после записи в регистр UDR. Во втором, запись в регистр UDR осуществляется во время передачи. В этом случае данные пересылаются в сдвиговый регистр после передачи последнего стоп-бита текущего кадра.

9-й бит данных должен быть загружен в бит TXB8 до записи младшего байта слова в регистр данных. После пересылки слова данных в сдвиговый регистр флаг UDRE регистра UCSRA устанавливается в 1, что означает готовность передатчика к получению нового слова данных. В этом состоянии флаг остается до следующей записи в буфер. Одновременно с пересылкой в регистре формируется служебная информация — старт-бит, возможный бит четности, а также один или два стоп-бита.

После загрузки сдвигового регистра его содержимое начинает сдвигаться вправо и поступать на вывод TXD. Скорость сдвига определяется настройками контроллера тактовых сигналов. В синхронном режиме изменение состояния вывода TXD происходит по одному из фронтов сигнала XCK. Если бит UCPOL регистра UCSRC сброшен в 0, изменение состояния

вывода происходит по нарастающему фронту сигнала, если же установлен в 1 — по спадающему фронту.

Если во время передачи в регистр UDR было записано новое слово данных, то после передачи последнего стоп-бита оно автоматически пересылается в сдвиговый регистр. Если же к моменту окончания передачи кадра буфер передатчика будет пуст, устанавливается флаг прерывания «передача завершена» TXC регистра UCSRA.

Сброс флага осуществляется аппаратно при входе в подпрограмму обработки соответствующего прерывания или программно, записью в этот бит логической 1.

Выключение передатчика осуществляется сбросом бита TXEN регистра UCSRB. Если в момент выполнения этой команды осуществлялась передача, сброс бита произойдет только после завершения текущей и отложенной передач. При выключенном передатчике вывод TXD может использоваться как контакт ввода/вывода общего назначения.

Работа приемника разрешается установкой бита RXEN регистра UCSRB. При установке бита вывод RXD подключается к приемнику USART и начинает функционировать как вход независимо от установок регистров управления портом. Если используется синхронный режим работы, переопределяется также функционирование вывода XCK.

Прием данных начинается сразу же после обнаружения приемником корректного старт-бита. Каждый бит содержимого кадра, затем считывается с частотой, определяемой установками контроллера скорости передачи или тактовым сигналом XCK. Считанные биты данных последовательно помещаются в сдвиговый регистр приемника до обнаружения первого стоп-бита кадра. После этого содержимое сдвигового регистра пересылается в буфер приемника, из которого принятое значение может быть получено путем чтения регистра данных модуля. При использовании 9-битных слов данных значение старшего бита может быть определено по состоянию флага RX8 регистра UCSRB. Причем содержимое старшего бита данных должно быть считано до обращения к регистру данных. Это связано с тем, что флаг RX8 отображает значение старшего бита слова данных кадра, находящегося на верхнем уровне буфера приемника, состояние которого при чтении регистра данных изменится.

Если во время приема кадра была включена схема контроля четности, она вычисляет бит четности для всех битов принятого слова данных и сравнивает его с принятым битом четности. Результат проверки запоминается в буфере приемника вместе с принятым словом данных и стоп-битами. Наличие или отсутствие ошибки контроля четности может быть затем определено по состоянию флага UPE. Этот флаг устанавливается в 1, если следующее слово, которое может быть прочитано из буфера, имеет

ошибку контроля четности. При выключенном контроле четности флаг UPE всегда читается как 0.

Блок приемника модулей USART имеет еще два флага, показывающих состояние обмена: флаг ошибки кадрирования FE и флаг переполнения DOR . Флаг FE устанавливается в 1, если значение первого стоп-бита принятого кадра не соответствует требуемому, т. е. равно 0. Флаг DOR индицирует потерю данных из-за переполнения буфера приемника. Этот флаг устанавливается в 1 в случае приема старт-бита нового кадра при заполненных буфере и сдвиговом регистре приемника. Установленный флаг DOR означает, что между прошлым байтом, считанным из регистра UDR, и байтом, считанным в данный момент, произошла потеря одного или нескольких кадров.

Все флаги ошибок буферизуются вместе со словом данных, т.е. соответствующие биты регистра UCSRA относятся к кадру, слово данных которого будет прочитано при следующем обращении к регистру данных UDR. Поэтому состояние этих флагов должно быть считано перед обращением к регистру данных. Кроме того, для совместимости с будущими устройствами рекомендуется при записи в регистр UCSRA сбрасывать соответствующие этим флагам биты записываемого значения в 0.

Для индикации состояния приемника в модулях USART используется флаг прерывания «прием завершен» RXC регистра UCSRA. Этот флаг устанавливается в 1 при наличии в буфере приемника непрочитанных данных и сбрасывается в 0 при опустошении буфера (после считывания всех находящихся в нем данных).

Выключение приемника осуществляется сбросом бита RXEN регистра UCSRB. В отличие от передатчика, приемник выключается сразу же после сброса бита, а значит, кадр, принимаемый в этот момент, теряется. Кроме того, при выключении приемника очищается его буфер – теряются также все непрочитанные данные. При выключенном приемнике вывод RXD может использоваться как контакт ввода/вывода общего назначения.

Для подключения по протоколу USART будем использовать разъем RS-232 (англ. Recommended Standard 232) – стандарт последовательной асинхронной передачи двоичных данных между терминалом (англ. Data Terminal Equipment, DTE) и коммуникационным устройством (англ. Data Communications Equipment, DCE). Максимальное расстояние между двумя устройствами - 15 метров. Информация передается по проводам цифровым сигналом с двумя уровнями напряжения. Логическому "0" соответствует положительное напряжение (от +5 до +15 В для передатчика), а логической "1" отрицательное (от -5 до -15 В для передатчика). Асинхронная передача данных осуществляется с фиксированной скоростью при само-

синхронизации фронтом стартового бита. В конце байта, перед стоп битом, может передаваться бит чётности P для контроля качества передачи. Он позволяет выявить ошибку в нечетное число бит (используется, так как наиболее вероятна ошибка в 1 бит).

Пример программы, реализующей прием/передачу данных по интерфейсу USART:

```
#include <avr/io.h> //подключаем библиотеку ATtiny2313
#include <Util/delay.h> //подключаем библиотеку функции задержки
#define F_CPU 8000000UL //кварц 8 MHz
void USART_Transmit (unsigned int data); //объявляем функцию передачи
unsigned int USART_Receive(void); //объявляем функцию приема
int unsigned resl; //объявляем беззнаковую целочисленную переменную resl
int main (void) //главная функция программы
  {
   DDRB=0XFF; //порт B как выход (диоды)
   DDRD=0X00; //порт D как вход (кнопки)
   PORTB=0X00; //обнуляем порт B
   PORTD=0XFF; //осуществляем подтяжку внутреннего резистора
   uint32_t baud=9600; //установка скорости передачи 9600 бит/с
   int baudrate=(F_CPU/(16*baud))-1; //вычисление значения для регистра скорости передачи UBRR
   UBRRH = (unsigned char)(baudrate>>8); //присваивание значения старшему байту регистра UBRR
   UBRRL = (unsigned char) baudrate; //присваивание значения младшему байту регистра UBRR
   UCSRA=0x00; //все флаги регистра  сброшены в ноль, удвоение скорости нет, режим мультипроцессорного обмена выключен
   UCSRB=0x18; //включение приемника и передатчика USART
   UCSRC=0x06; //асинхронный режим,нет проверки на четность,1 стоповый бит,8 бит данных
   void USART_Transmit (unsigned int data) //функция отправки данных по интерфейсу USART
     {
      while (!(UCSRA&(1<<UDRE))){}; //бесконечный цикл, если регистр данных UDR полон
      UDR=data; //если регистр данных пуст запись в регистр данных UDR данных для отправки по USART
     }
   unsigned int USART_Receive(void) //функция приема данных по интерфейсу USART
     {
      while (!(UCSRA&(1<<RXC))){}; //бесконечный цикл, если в регистре UDR нет непрочитанных данных
      resl=UDR; //если в регистре UDR есть непрочитанные данные записывает в переменную значение регистра
```

```
            if(UCSRB&((1<<FE)|(1<<DOR)|(1<<UPE))){return -1;} //при ошибке кадрирования, переполнении или ошибке в принятых данных, функция приема вернет значение -1
             return resl; //в случае успешного приема данных возвращает 1 байт данных
            }
          while (1) //задаем бесконечный цикл
            {
             USART_Receive(); //прослушиваем линию на прием данных
             PORTB=resl; //если данные приняты запишем их в регистр PORTB
             USART_Transmit(resl); //отправим принятые данные обратно отправителю
             _delay_ms(500); //подождем 500 мс и вернемся в начало бесконечного цикла
            }
        }
```

Задание: на основе теоретических знаний и примеров программ, изложенных в данной лабораторной работе написать программы, осуществляющие:

1) Прием символов микроконтроллером, передаваемых компьютером через программу «Терминал» по интерфейсу RS-232 (USART со стороны микроконтроллера), обработка данных микроконтроллером и отправка этих же символов обратно компьютеру.

2) Эмуляцию работы начального меню роутера: эмуляция входа в привилегированный режим, эмуляция входа в конфигурационный режим, эмуляция при присваивании имени роутеру, а также обратный выход в вышестоящие режимы. Обмен информацией осуществляется через программу «Терминал». Микроконтроллер изначально передает компьютеру по интерфейсу RS-232 команду Router> которую можно увидеть в программе «Терминал». Для входа в привилегированный режим необходимо в «Терминале» ввести команду enable. В ответ на команду микроконтроллер отобразит в окне «Терминала» на новой строке Router#. Далее отправив команду configure terminal, микроконтроллер перейдет в конфигурационный режим Router(config)#. Если в конфигурационном режиме прописать команду hostname <имя>, то в ответ микроконтроллер отобразит имя(config)#, то есть произойдет присвоение имени «роутеру». Выход обратно в привилегированный режим и режим пользователя осуществляется последовательно по команде exit.

Контрольные вопросы

1. Назовите основные части структуры модуля приемо-передатчика USART и их функциональные назначения?
2. Какие виды прерываний используются для передачи данных по интерфейсу USART?

3. Какие регистры используются для управления параметрами передачи модуля приемо-передатчика USART?

4. Какой формат кадра используется при приеме/передачи данных по интерфейсу USART?

5. Опишите алгоритм передачи данных по интерфейсу USART.

6. Опишите алгоритм приема данных по интерфейсу USART.

7. Какие режимы работы модуля приемо-передатчика USART вы знаете?

Рекомендуемая литература

1 Гребнев В. В. Микроконтроллеры семейства AVR фирмы Atmel / / В. В. Гребнев. – М.: ИП РадиоСофт, 2002 – 176 с: ил. С. 47−51.

2 Евстифеев А. В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя / А. В. Евстифеев. – М: Издательский дом «Додека-XXI», 2007. − 432 с.: ил. С. 317−335.

3 ATTINY2313 – 8-bit AVR Microcontroller with 2K Bytes In-System Programmable Flash – ATMEL Corporation. – Сайт документации на электронные компоненты. – (Англ.). – URL: http://www.alldatasheet.com/ /datasheet-pdf/pdf/80317/ATMEL/ATTINY2313.html

4 Белов А. В. Микроконтроллеры AVR в радиолюбительской практике. – СПб.: Наука и Техника, 2007. – 352 с.: ил.

5 Белов А. В. Самоучитель по микропроцессорной технике. – СПб.: Наука и Техника, 2003. – 224 с.: ил.

6 Белов А. В. Создаем устройства на микроконтроллерах. – СПб.: Наука и Техника, 2007. – 304 с.: ил.

Лабораторная работа №6 «Управление шпиндельным двигателем, подключенным к микроконтроллеру»

***Цель работы*:** научиться управлять шпиндельным двигателем, подключенным к микроконтроллеру через устройство сопряжения.

***Задачи*:** изучить принцип работы шагового двигателя, условия его разгона, разобрать примеры программ по управлению шаговым двигателем, позволяющих управлять скоростью вращения.

Приборы и принадлежности: плата на основе микроконтроллера ATtiny2313, Datasheet к микроконтроллеру ATtiny2313, LPT- программатор, ПК с установленными PonyProg и пакетом программ WinAVR, шпин-

дельный двигатель, устройство сопряжения двигателя с микроконтроллером.

Краткие теоретические сведения

Бесколлекторный двигатель (шпиндельный двигатель) используют в приводах НГМД и НЖМД, в лазерном принтере он применяется для перемещения лазерного луча и для механизма протяжки. Этот двигатель используют там, где требуется постоянная, высокая и стабильная скорость вращения.

Этот тип двигателя характеризуется следующими преимуществами: малая неравномерность мгновенной скорости вращения; низкий уровень акустических шумов небольшие габариты, масса, потребляемая мощность; высокая надежность; низкая стоимость.

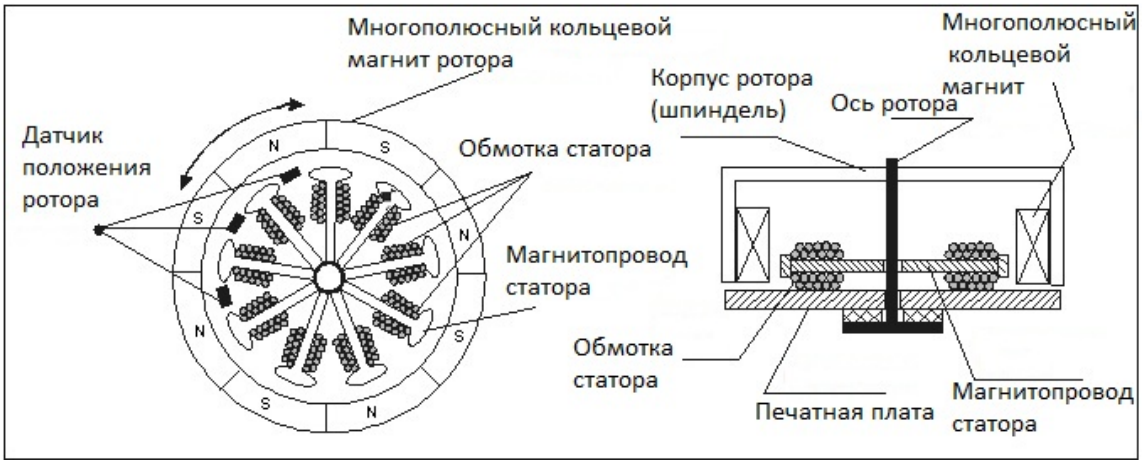


Рисунок 8 – Конструкция шпиндельного двигателя

В бесколлекторном двигателе на роторе расположены постоянные магниты, создающие магнитный поток. Эти магниты выполнены чаще всего в виде многополюсного кольцевого магнита. Обмотки статора являются неподвижными, т.е. получается обращенная конструкция (рис. 8).

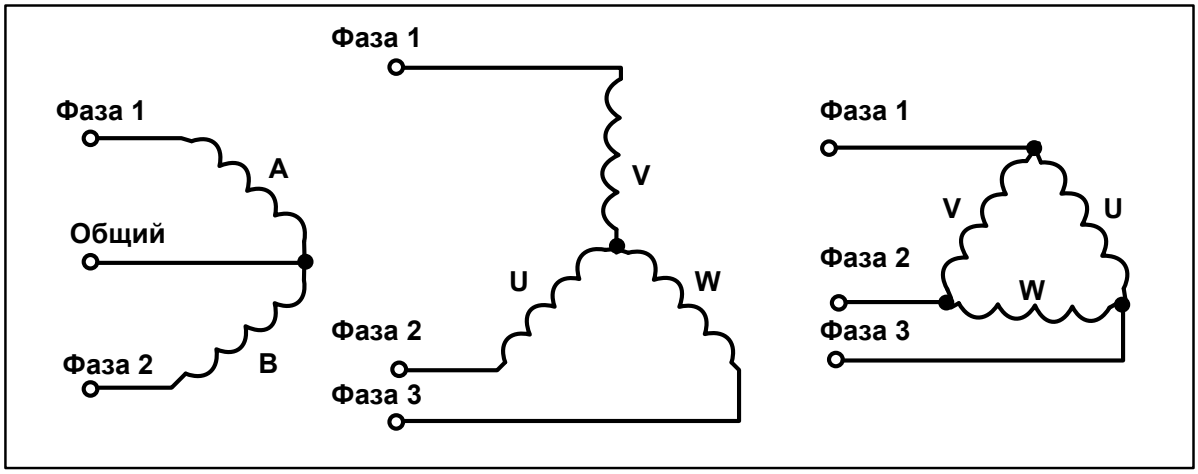


Рисунок 9 – Схемы включения обмоток шпиндельных двигателей

Вращающий момент в двигателе создается в результате взаимодействия магнитного потока в промежутке между полюсами магнита ротора и основанием статора с проводниками обмотки, по которым протекает электрический ток. Управление коммутацией катушек обмотки статора в зависимости от положения полюсов магнита ротора осуществляется специальной схемой (драйвером) по сигналам датчиков положения ротора. На практике нашли применение двухфазные и трехфазные двигатели, возможные схемы включения обмоток приводятся на рисунке 9.

В вентильных бесколлекторных двигателях магнит ротора имеет, как правило, 6-9 полюсов. Магнит изготавливают из магнитотвердых материалов на основе порошка феррита различных металлов.

Для того, чтобы двигатель вращался с постоянной скоростью, применяется датчик частоты вращения, обеспечивающий контроль за скоростью вращения. В зависимости от конструкции двигателя и числа фаз количество датчиков положения ротора меняется от 1 до 3.

Для управления бесколлекторными двигателями применяются специальные микросхемы – драйверы двигателя. Эти микросхемы выполняют следующие функции: усиление и обработка сигналов с датчиков положения ротора; усиление и обработка сигнала от датчика частоты вращения; формирование сигналов коммутации обмоток статора; стабилизация частоты вращения.

Условно микросхемы драйверов можно разделить на мощные и маломощные. У мощных – обмотки статора подключаются непосредственно к выводам микросхемы. У маломощных – двигатель подключается через транзисторные усилительные ключи.

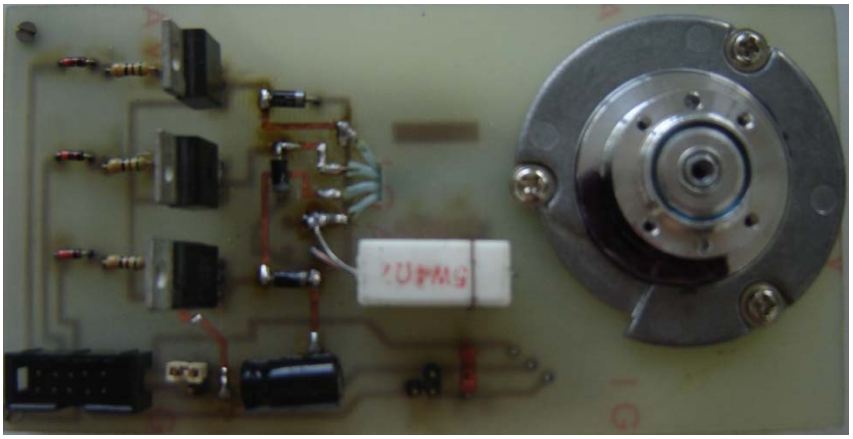


Рисунок 10 – устройство сопряжения шпиндельного двигателя с микроконтроллером

На вход микросхемы подаются сигналы от датчиков положения ротора и от датчика частоты вращения. В большинстве микросхем имеется входной сигнал START/STOP для включения и выключения двигателя. Так как микросхема поддерживает скорость вращения стабильной, то сигнал от датчика скорости вращения сравнивается с сигналом опорной частоты. Сигнал опорной частоты представляет собой синусоидальное напряжение, формируемое либо кварцевым (емкостным) резонатором, либо ведущей микросхемой (например микроконтроллером). Сигнал частоты вращения обычно обозначается FG.

Для проведения лабораторной работы используется устройство сопряжения шпиндельного двигателя Nidec DLH-3814A с микроконтроллером (рис. 10).

На рисунке 11 приведена электрическая схема сопряжения двигателя с микроконтроллером. При подаче положительного потенциала на входы №1, №2, №3 транзисторы T1,T2,T3, работающие в качестве ключей, открываются, и через катушки L1, L2, L3 протекает электрический ток. Диоды D1, D2, D3 служат для защиты микроконтроллера от возможных обратных токов. Сопротивления R1, R2, R3 ограничивают ток, поступающий с выводов микроконтроллера, так как максимально допустимый ток с одного вывода равен 20 мА. Диоды D4, D5, D6 гасят ток самоиндукции катушек, при выключении питания, тем самым оберегая транзистор от высоких напряжений.

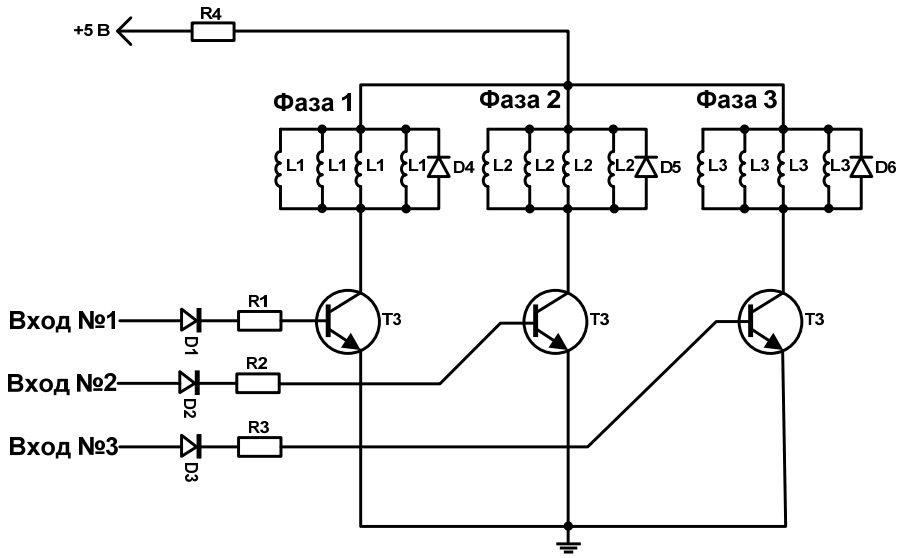


Рисунок 11 – электрическая схема устройства сопряжения шпиндельного двигателя с микроконтроллером

На статоре трехфазного двигателя DLH-3814A упорядоченно расположены 12 катушек индуктивности, по 4 на каждую фазу (рис. 12). Поэтому в шаговом режиме ротор двигателя принимает одно из 12 фиксированных положений, полушаговом – одно из 24. Для работы двигателя в шаговом режиме необходимо последовательно подавать напряжение на каждую из трех фаз. Тогда шпиндель будет вращаться таким образом, что каждый его магнитный полюс будет останавливаться над одной из катушек. В полушаговом режиме напряжение подается сначала на одну фазу, затем на две. При этом каждый магнитный полюс шпинделя останавливается сначала над катушкой, затем – между катушками.

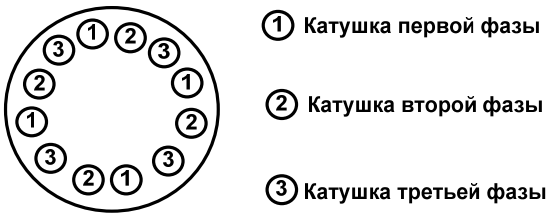


Рисунок 12 – Расположение катушек индуктивности в статоре

Рассмотрим пример программы для управления шпиндельным двигателем, позволяющей раскручивать и останавливать шпиндельный двигатель. Первая, вторая и третья фазы двигателя f1, f2, f3 будут подключаться к выводам микроконтроллера PB0, PB1, PB2 соответственно. Для рас-

крутки двигателя в шаговом режиме необходимо последовательно подавать на эти выводы 001, 010,100, в полушаговом – 001,011,010,110,100,101. Так как полушаговый режим более стабильный, то будем использовать его. Для мгновенной остановки двигателя необходимо подать напряжение сразу на три фазы, то есть код 111 на выводы PB0, PB1, PB2:

```
#include <avr/io.h> //подключаем библиотеку для микроконтроллера
#include <util/delay.h> //подключаем библиотеку для использования функции delay в программе
int k,m,i; //объявляем целочисленные переменные k, m, i
float n=5001; //объявляем переменную с плавающей точкой n и присваиваем ей значение 5001
float m=2000; //объявляем переменную с плавающей точкой m и присваиваем ей значение 2000
void MKS(int n); //объявляем функцию расчета задержки MKS
void rot(int n, int m); //объявляем функцию раскрутки двигателя rot
void MKS(int n) //определяем функцию расчета задержки MKS
  {
   for (k=0; k<n; k++) //цикл выполняется пока значение переменной к меньше значения переменной n
     {
      _delay_us(5); //задержка 5 мс
     }
  }
void rot(int n, int m) //n - ms, m - кол. оборотов (1/4). Определяем функцию раскрутки двигателя rot
  {
   for(i=0; i<m; i++)
     {
      PORTB=0b00000001; //напряжение на фазу f1
      MKS(n);
      PORTB=0b00000011; //напряжение на фазы f1,f2
      MKS(n);
      PORTB=0b00000010; //напряжение на фазу f2
      MKS(n);
      PORTB=0b00000110;  //напряжение на фазы f2,f3
      MKS(n);
      PORTB=0b00000100; //напряжение на фазу f3
      MKS(n);
      PORTB=0b00000101; //напряжение на фазы f1,f3
      MKS(n);
     }
  }
int main (void) //главная функция программы
  {
   DDRB=0xff; //порт B как выход
   DDRD=0x00; //порт D как вход
   PORTB=0x00; //обнуление порта B
```

```
        PORTD=0xff; //подтяжка внутренних резисторов против дрожания
        if (~PIND&(1<<PD0)) //функция мгновенной остановки двигателя при нажатии  на кнопку D0
            {
             PORTB=0b00000111;
            }
             rot (n, m); //Вызываем функцию раскрутки двигателя rot в главной программе
      }
```

Задание: написать программу, осуществляющую разгон шагового двигателя до 4200 оборотов в минуту и поддержание этой достигнутой скорости. Измерение скорости провести с помощью оптического метода.

Контрольные вопросы

1. Что такое шпиндельные двигатели и где они применяются?
2. Какие преимущества имеют бесколлекторные двигатели?
3. Опишите конструкцию и принцип работы бесколлекторных двигателей.
4. Какую функцию выполняют драйверы бесколлекторных двигателей?
5. В каких режимах могут работать шпиндельные двигатели?

Рекомендуемая литература

1 Гребнев В. В. Микроконтроллеры семейства AVR фирмы Atmel / / В. В. Гребнев. – М.: ИП РадиоСофт, 2002 – 176 с: ил. С. 85−90.

2 Евстифеев А. В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя / А. В. Евстифеев. – М: Издательский дом «Додека-XXI», 2007. − 432 с.: ил. С. 207−225.

3 ATTINY2313 – 8-bit AVR Microcontroller with 2K Bytes In-System Programmable Flash – ATMEL Corporation. – Сайт документации на электронные компоненты. – (Англ.). – URL: http://www.alldatasheet.com/ /datasheet-pdf/pdf/80317/ATMEL/ATTINY2313.html

4 Бесколлекторные двигатели. – Сайт учебно-практического центра «Эксперт». – URL: http://xprt.ru/index.php?option=com_content&task=view &id= 79&Itemid=46

5 Белов А. В. Микроконтроллеры AVR в радиолюбительской практике. – СПб.: Наука и Техника, 2007. – 352 с.: ил.

6 Белов А. В. Самоучитель по микропроцессорной технике. – СПб.: Наука и Техника, 2003. – 224 с.: ил.

7 Белов А. В. Создаем устройства на микроконтроллерах. – СПб.: Наука и Техника, 2007. – 304 с.: ил.

Лабораторная работа №7 «Программный отклик в среде HiAsm»

Цель: изучить программу HiAsm, с помощью которой возможно написание приложений программ (под ОС Windows) и научиться обеспечивать ее взаимодействие (прием, обработку, индикацию и отправку данных) с микроконтроллером, подключенным к ПК.

Задачи: освоить программную среду HiAsm, структуру интерфейса USART микроконтроллера, назначение функциональных блоков, принцип приема/передачи данных по прерыванию с интерфейса USART, разобрать примеры программ в среде HiAsm для управления передачей/приемом команд по интерфейсу RS-232.

Приборы и принадлежности: плата на основе микроконтроллера ATtiny2313, Datasheet к микроконтроллеру ATtiny2313, LPT- программатор, ПК с установленными PonyProg и пакетом программ WinAVR, HiAsm.

Краткие теоретические сведения

HiAsm – система визуального проектирования и разработки приложений, которая позволяет писать программы, обладая минимальными знаниями в области функционирования ОС. Весь процесс проектирования состоит в размещении элементов на рабочем столе программы и их связывания друг с другом, что в первом приближении можно сравнить с построением обычного алгоритма на основе стандартизированных функциональных блоков (циклов, условных блоков, переходов, операторов и прочее).

С помощью HiAsm можно написать приложение для передачи данных по интерфейсу RS-232. Согласно принятым по интерфейсу USART данным, микроконтроллер может управлять периферийными устройствами, в частности состояниями портов ввода/вывода.

Используя HiAsm, напишем приложение, которое будет отправлять по интерфейсу RS-232 числа от 0 до 8. Для этого выполним следующий алгоритм действий:

1) Проходим по вкладкам «файл» → «новый». Появится окно «создать новый проект», в котором выбираем «приложение Windows». Появится рабочее окно проекта.
2) На панели инструментов кликаем на значок «редактор формы». Появится окно «Form».
3) Кликаем на вкладку «элементы», которая находится слева от окна «Form».

4) Выбираем элемент «кнопка(button)» и помещаем его в окно «Form».

5) Кликаем на элемент «кнопка(button)», затем на вкладку «свойства», расположенную слева от окна «Form». Вместо «push» прописываем «отправить сигнал».

6) Аналогично выбираем элемент «надпись(label)», помещаем его в окно «Form» и вместо «label» прописываем «номер светодиода».

7) Выбираем элемент «поле ввода(edit)», помещаем его в окно «Form» и удаляем надпись «Edit» в свойствах.

8) Проходим по вкладкам «Вид» → «Редактор формы». Откроется окно с тремя ранее добавленными элементами в виде блоков с выводами.

9) Кликаем на вкладки «элементы» → «инструменты», выбираем элементы «разветвитель» и «поток-данные» и устанавливаем их в окно рядом с тремя ранее добавленными. В свойствах разветвителя выбираем разветвление 1:3.

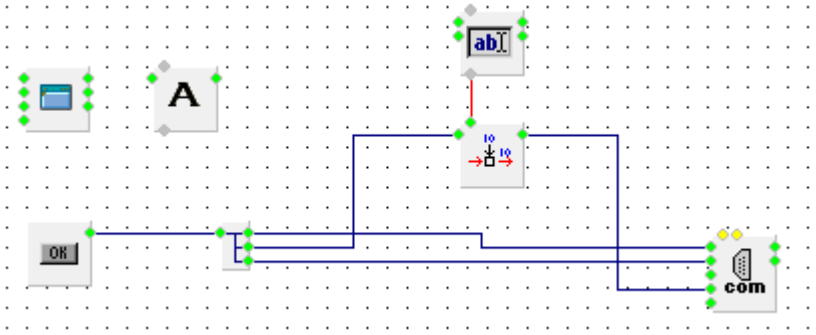


Рисунок 13 – Правильное соединение элементов для программы

10) Аналогично выбираем и устанавливаем элемент «COM-порт». В его свойствах выбираем физический COM1 и скорость подключения, к примеру, 9600 бит/с.

11) Соединяем выводы элементов, как показано на рисунке 13. При соединениях, когда указатель наводится на подключаемые точки, всплывает подсказка об их назначении. Таким образом, при нажатии на кнопку «ОК» сигнал попадет на разветвитель, который последовательно повторит его на своих выходах. Сначала с первого выхода сигнал активирует COM порт, затем сигнал со второго выхода активирует передачу данных из поля ввода, таким образом, данные передаются по протоколу USART, а затем с третьего выхода освобождается COM порт и его можно использовать для других задач.

12) В панеле инструментов нажимаем на кнопку «компилировать». Созданное приложение для Windows сохранится в папке с проектом.

13) Откроем приложение, получено в результате компиляции (рис. 14).

Если ввести в поле «Номер светодиода» число и нажать на кнопку, то это число программа отправит в COM-порт, к которому будет подключен микроконтроллер через интерфейс USART.

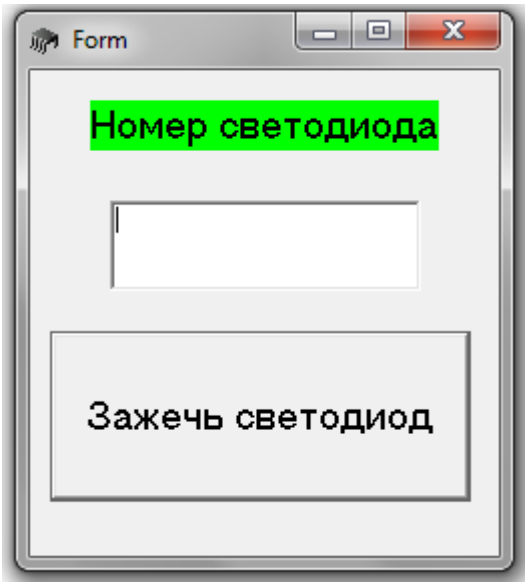


Рисунок 14 – Созданное с помощью HiAsm приложение

Программа для микроконтроллера, обрабатывающая полученные по интерфейсу USART данные, управляющая светодиодами, подключенными к выводам порта B, к примеру, выглядит следующим образом:

```
#include <avr/io.h> //подключаем библиотеку ATtiny2313
#include <Util/delay.h> //подключаем библиотеку функции задержки
#define F_CPU 8000000UL //кварц 8 MHz
void USART_Transmit (unsigned int data); //объявляем функцию передачи
unsigned int USART_Receive(void); //объявляем функцию приема
int unsigned resl; //объявляем беззнаковую целочисленную переменную resl
int main (void) //главная функция программы
  {
   DDRB=0XFF; //порт B как выход (диоды)
   DDRD=0X00; //порт D как вход (кнопки)
   PORTB=0X00; //обнуляем порт B
   PORTD=0XFF; //осуществляем подтяжку внутреннего резистора
   uint32_t baud=9600; //установка скорости передачи 9600 бит/с
   int baudrate=(F_CPU/(16*baud))-1; //вычисление значения для регистра скорости передачи UBRR
   UBRRH = (unsigned char)(baudrate>>8); //присваивание значения старшему байту регистра UBRR
   UBRRL = (unsigned char) baudrate; //присваивание значения младшему байту регистра UBRR
   UCSRA=0x00; //все флаги регистра  сброшены в ноль, удвоение скорости нет, режим мультипроцессорного обмена выключен
```

```
        UCSRB=0x18; //включение приемника и передатчика USART
        UCSRC=0x06; //асинхронный режим,нет проверки на четность,1 стоповый бит,8 бит данных
        void USART_Transmit (unsigned int data) //функция отправки данных по интерфейсу USART
          {
           while (!(UCSRA&(1<<UDRE))){}; //бесконечный цикл, если регистр данных UDR полон
           UDR=data; //если регистр данных пуст запись в регистр данных UDR данных для отправки по USART
          }
        unsigned int USART_Receive(void) //функция приема данных по интерфейсу USART
          {
           unsigned int resl; //объявляет беззнаковую целочисленную переменную resl, размером 16 бит
           while (!(UCSRA&(1<<RXC))){}; //бесконечный цикл, если в регистре UDR нет непрочитанных данных
           resl=UDR; //если в регистре UDR есть непрочитанные данные записывает в переменную значение регистра
           if(UCSRB&((1<<FE)|(1<<DOR)|(1<<UPE))){return -1;} //при ошибке кадрирования, переполнении или ошибке в принятых данных, функция приема вернет значение -1
           return resl; //в случае успешного приема данных возвращает 1 байт полученных данных
          }
        while (1) //задаем бесконечный цикл
          {
           PORTB&=(1<<(USART_Receive()-48)); // загорается соответсвующий отправленным данным номер светодиода
          }
      }
```

Задание: на основе теоретических знаний и примеров программ, изложенных в данной лабораторной работе написать приложение в среде HiAsm и программу, и прошивки микроконтроллера. Приложение должно, реализовать передачу/прием команд по интерфейсу RS-232. Программа для микроконтроллера должна по командам из приложения изменять скорость вращения шпингалетного двигателя, подключенного к микроконтроллеру. Так же, скорость вращения изменится и по кнопке D6. Предусмотреть 5 различных скоростей вращения. При этом, то на какой скорости вращается двигатель должно отображаться в приложении.

Контрольные вопросы

1. Какие преимущества имеет система визуального проектирования HiAsm?

2. Какие виды прерываний используются для передачи данных по интерфейсу USART?

3. Какие регистры используются для управления параметрами передачи модуля приемо-передатчика USART?

4. Какой формат кадра используется при приеме/передачи данных по интерфейсу USART?

5. Опишите алгоритм передачи данных по интерфейсу USART.

6. Опишите алгоритм приема данных по интерфейсу USART.

Рекомендуемая литература

1 Гребнев В. В. Микроконтроллеры семейства AVR фирмы Atmel / / В. В. Гребнев. – М.: ИП РадиоСофт, 2002 – 176 с: ил. С. 9−36.

2 Евстифеев А. В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя / А. В. Евстифеев. – М: Издательский дом «Додека-XXI», 2007. − 432 с.: ил. С. 317−335.

3 ATTINY2313 – 8-bit AVR Microcontroller with 2K Bytes In-System Programmable Flash – ATMEL Corporation. – Сайт документации на электронные компоненты. – (Англ.). – URL: http://www.alldatasheet.com/ /datasheet-pdf/pdf/80317/ATMEL/ATTINY2313.html

4 Белов А. В. Микроконтроллеры AVR в радиолюбительской практике. – СПб.: Наука и Техника, 2007. – 352 с.: ил.

5 Белов А. В. Самоучитель по микропроцессорной технике. – СПб.: Наука и Техника, 2003. – 224 с.: ил.

6 Белов А. В. Создаем устройства на микроконтроллерах. – СПб.: Наука и Техника, 2007. – 304 с.: ил.

Лабораторная работа №8 «Реализация протоколов ARP, ICMP, IP на микроконтроллере»

Цель: научиться программно, с использованием микроконтроллера, принимать и передавать данные посредством Ethernet-интерфейса по протоколам ARP, ICMP, IP.

Задачи: изучить структуру и принцип работы микроконтроллера ENC28J60, его регистры, особенности чтения/записи физических регистров, SPI интерфейс и SPI команды (чтение/запись регистров, буфера, установка/снятие бит по маске, мягкий сброс) для управления ENC28J60, сопряжение микроконтроллеров ATmega168 и ENC28J60, инициализацию ENC28J60, модель OSI, протоколы ARP, ICMP, IP, разобрать программы для реализации протоколов ARP, ICMP, IP на плате.

Приборы и принадлежности: плата на основе микроконтроллеров ATmega168 и ENC28J60, Datasheet к микроконтроллерам ATmega168 и ENC28J60, LPT- программатор, ПК с установленными PonyProg и пакетом программ WinAVR.

Краткие теоретические сведения

Для проведения лабораторной работы используется плата на основе микроконтроллера AVR ATmega168 и ENC28J60 (рис. 15).

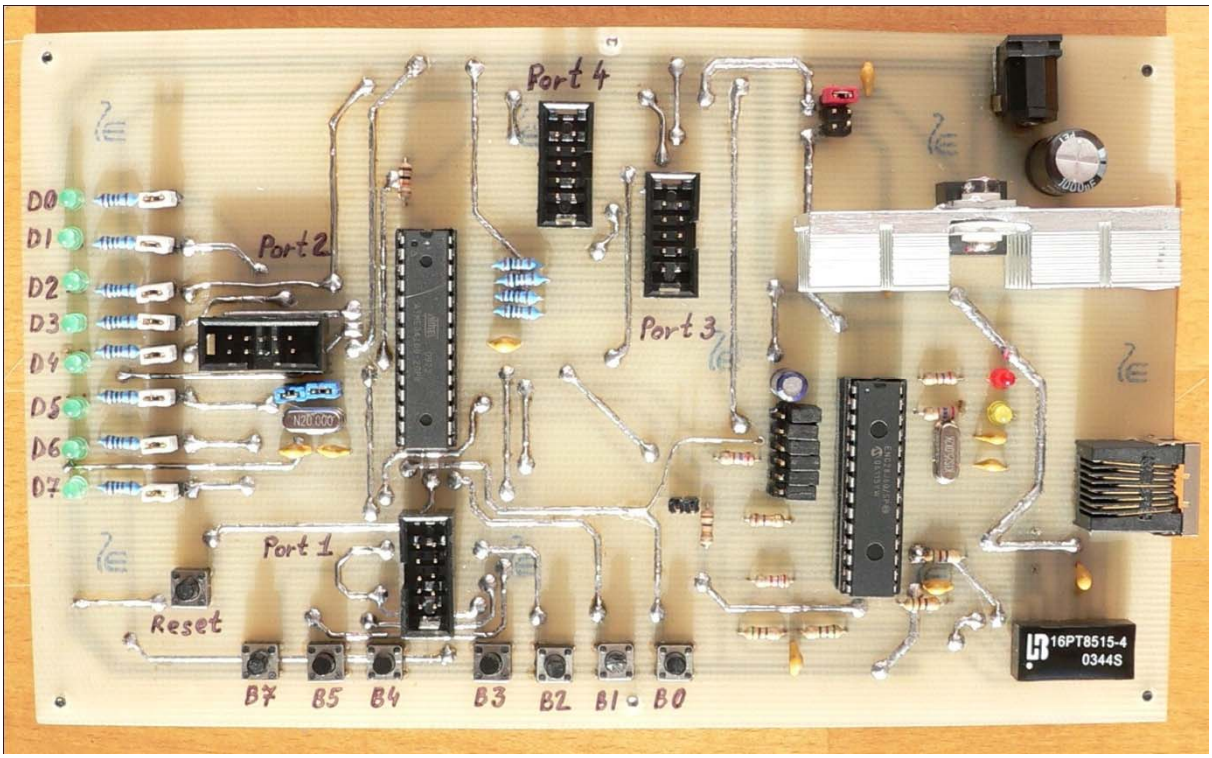


Рисунок 15 – Плата на основе микроконтроллеров ATmega168 и ENC28J60

Ethernet-чип ENC28J60 поддерживает интерфейс Ethernet стандарта 10BASE-T и позволяет передавать и принимать данные со скоростью 10 Мбит/с. Для подключения к ПК на плате предусмотрен разъем RJ-45. Основная функция ENC28J60 – осуществить передачу данных, полученных

от микроконтроллера ATmega168, по канальному и физическому уровню согласно OSI-модели, а также передать принятую по интерфейсу Ethernet информацию в микроконтроллер.

Упрощенная блок-схема ENC28J60 приведена на рисунке 16. Роль физического уровня (PHY) – кодирование двоичной последовательности фреймов, формирующихся на канальном уровне, ввод и прием сигналов через среду передачи данных. Доступ к PHY происходит через независимый от среды передачи данных интерфейс MII, который задуман так, чтобы канальный уровень мог «абстрагироваться» от типа среды передачи данных. PHY имеет свой набор 16-битных регистров, доступ к которым осуществляется через MII. Фактически, MII – это набор регистров, через которые управляется PHY.

На канальном уровне решаются задачи физической адресации, анализа сетевой топологии, уведомления об ошибках и управления потоками. Этот уровень делится на два подуровня: логическое управление каналом (LLC) и доступ к среде передачи данных (MAC).

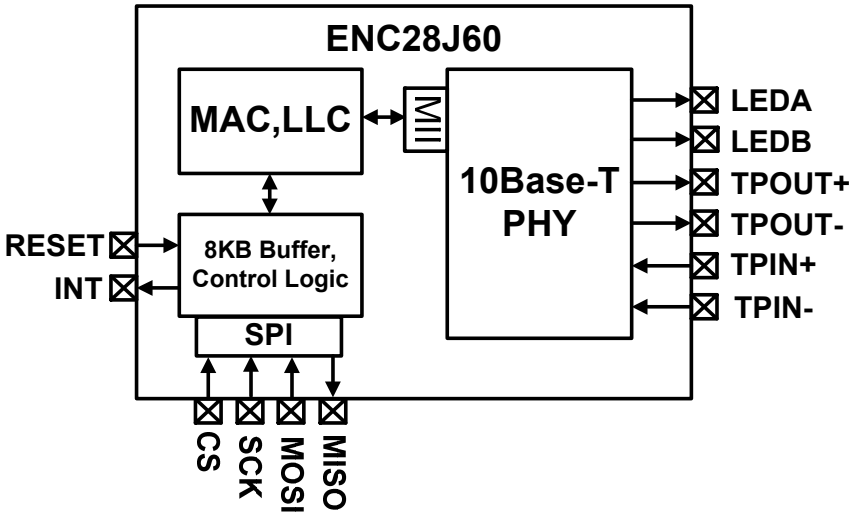


Рисунок 16 – Упрощенная блок-схема ENC28J60

Управляющая логика (Control Logic) обслуживает буфер, из которого MAC берёт отправляемые данные и складывает принятые, управляет режимами энергопотребления

Вся память в ENC28J60 делится на буфер для данных, управляющие регистры и регистры PHY. Управляющие регистры служат для конфигурации микрочипа и располагаются в 4-х адресных пространствах – банках. Каждый банк имеет размер 32 регистра, причём последние 5 регистров

одинаковы для всех банков (рис. 17). Назначение каждого регистра и его отдельных бит можно посмотреть в Datasheet.

	Банк 0	Банк 1	Банк 2	Банк 3
0x00	ERDPTL	EHT0	MACON1	MAADR1
0x01	ERDPTH	EHT1	MACON2	MAADR0
0x02	EWRPTL	EHT2	MACON3	MAADR3
0x03	EWRPTH	EHT3	MACON4	MAADR2
0x04	ETXSTL	EHT4	MABBIPG	MAADR5
0x05	ETXSTH	EHT5	–	MAADR4
0x06	ETXNDL	EHT6	MAIPGL	EBSTSD
0x07	ETXNDH	EHT7	MAIPGH	EBSTCON
0x08	ERXSTL	EPMM0	MACLCON1	EBSTCSL
0x09	ERXSTH	EPMM1	MACLCON2	EBSTCSH
0x0A	ERXNDL	EPMM2	MAMXFLL	MISTAT
0x0B	ERXNDH	EPMM3	MAMXFLH	–
0x0C	ERXRDPTL	EPMM4	Reserved	–
0x0D	ERXRDPTH	EPMM5	MAPHSUP	–
0x0E	ERXWRPTL	EPMM6	Reserved	–
0x0F	ERXWRPTH	EPMM7	–	–
0x10	EDMASTL	EPMCSL	Reserved	–
0x11	EDMASTH	EPMCSH	MICON	–
0x12	EDMANDL	–	MICMD	EREVID
0x13	EDMANDH	–	–	–
0x14	EDMADSTL	EPMOL	MIREGADR	–
0x15	EDMADSTH	EPMOH	Reserved	ECOCON
0x16	EDMACSL	EWOLIE	MIWRL	Reserved
0x17	EDMACSH	EWOLIR	MIWRH	EFLOCON
0x18	–	ERXFCON	MIRDL	EPAUSL
0x19	–	EPKTCNT	MIRDH	EPAUSH
0x1A	Reserved	Reserved	Reserved	Reserved
0x1B	EIE	EIE	EIE	EIE
0x1C	EIR	EIR	EIR	EIR
0x1D	ESTAT	ESTAT	ESTAT	ESTAT
0x1E	ECON2	ECON2	ECON2	ECON2
0x1F	ECON1	ECON1	ECON1	ECON1

Основное
Указатели буфера
DMA
Фильтрация пакетов
MAC-адрес
Регистры MAC
Регистры MII
Управление потоком
Прерывания
Идентификатор ревизии (e.g. 0x06 = ревизия 7)
Управление ножкой CLKOUT
«Встроенный тест»

Рисунок 17 – Управляющие регистры ENC28J60

Буфер данных, общий размер которого 8Кбайт, с помощью управляющих регистров обычно делят на две части: одну для хранения отправляемых данных, другую – для хранения принятых (рис. 18). Регистры ERXST и ERXND определяют границы буферного пространства, в которое будут складываться принятые данные, регистры ETXST и ETXND – границы для хранения отправляемых данных. Например, если записать ERXST=0, ERXND=0x1000, ETXST=0x1001, ETXND=0x1fff, то буфер будет разделен на две равные части для выполнения вышеуказанных функций. Чтобы прочитать или записать данные в буфер необходимо установить регистры ERDPT и EWRPT, которые называют указателем чтения и записи буфера.

Регистр ERXRDPT указывает на адрес ячейки буфера, с которой нужно прочитать данные, а регистр ERXWRPT – на адрес ячейки, в которую

их нужно записать. Для того, чтобы каждый раз не устанавливать значения указателей при операциях чтения/записи буфера, предусмотрена функция их автоматического инкрементирования.

После того, как значения указателей чтения/записи достигнут верхней границы части буфера, определенной для принятых пакетов (ERXND), указатель перейдет на нижнюю (ERXST) и будет инкрементироваться дальше. Поэтому место для хранения принятых пакетов называют кольцевым буфером.

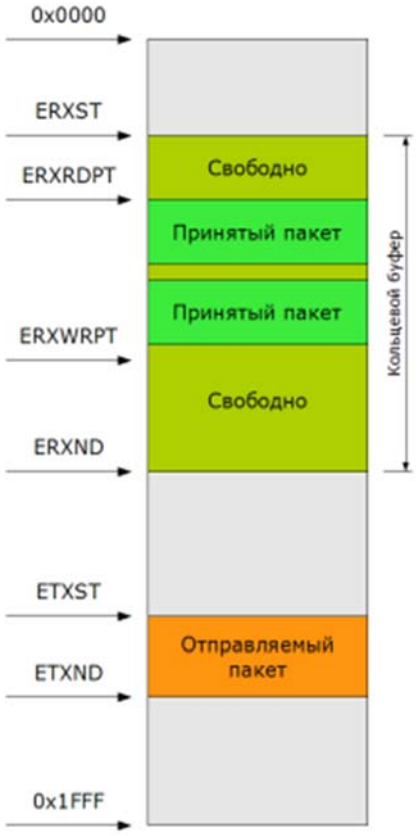


Рисунок 18 – Буфер ENC28J60

Микроконтроллеры экспериментальной платы соединены между собой шестью выводами. Со стороны ATmega168 это выводы: вход тактового сигнала (XTAL1), вход для прерываний (INT0), вывод приема данных последовательного периферийного интерфейса SPI (MISO), вывод передачи данных SPI (MOSI), вывод тактового сигнала SPI (SCK), вывод выбора активного ведомого SPI (SS). Со стороны ENC28J60 это выводы: вывод прерываний (INT), вывод тактового сигнала (CLKout), вывод передачи данных SPI (SO), вывод приема данных SPI (SI), вход тактового сигнала SPI (SCK), вывод выбора активного ведомого SPI (CS). Микроконтроллер ENC28J60 тактирует ATmega168 через вывод XTAL1, а через вывод INT0

может генерировать прерывания, подав на него напряжение высокого уровня.

Микроконтроллеры ATmega168 и ENC28J60 передают и принимают данные друг от друга посредством SPI-интерфейса. Для управления модулем SPI микроконтроллера ATmega168 предназначен регистр управления (SPCR), содержащий восемь управляющих бит: бит разрешения прерывания от SPI (SPIE), бит включения/выключение SPI (SPE), бит порядка передачи данных (DORD), бит выбора режима работы (MSTR), бит полярности тактового сигнала (CPOL), бит фазы тактового сигнала (CPHA), биты выбора скорости передачи (SPR1, SPR0). Контроль состояния модуля, а также дополнительное управление скоростью обмена осуществляется с помощью регистра состояния (SPSR), в котором используют три бита: бит флага прерывания от SPI (SPIF), бит флага конфликта записи (WCOL), бит удвоения скорости обмена (SPI2X). Для последующего описания значений бит введем обозначения: H – высокий уровень напряжения, L – низкий уровень напряжения. Модуль SPI имеет регистр данных (SPDR). Запись в этот регистр инициирует начало передачи.

Последовательный периферийный интерфейс SPI является полнодуплексным, то есть, передача и прием данных между микроконтроллерами может осуществляться одновременно. ATmega168 работает в режиме ведущего (Master) и через вывод SCK задает тактовый сигнал микроконтроллеру ENC28J60, который работает в режиме ведомого (Slave). При записи в регистр данных SPI ведущего ATmega168 запускается генератор тактового сигнала модуля SPI, и данные побитно поступают на вывод MOSI и соответственно на вывод SI ведомого ENC28J60. Порядок передачи бит данных определяется состоянием бита DORD регистра SPCR. Если бит установлен в H, первым передается младший бит байта, если же в L – старший бит. После выдачи последнего бита текущего байта генератор тактового сигнала останавливается с одновременной установкой в H флага «Конец передачи» (SPIF). Если прерывания от модуля SPI разрешены (флаг SPIE регистра SPCR установлен в H), генерируется запрос на прерывание. После этого ведущий микроконтроллер может начинать передачу следующего байта либо, подав на вход CS ведомого Ethernet-микроконтроллера напряжение высокого уровня, перевести его в состояние ожидания. Одновременно с передачей данных от ведущего ATmega168 к ведомому ENC28J60 происходит передача данных в обратном направлении, при условии, что на входе CS ведомого присутствует напряжение низкого уровня. Таким образом, в каждом цикле сдвига происходит обмен данными между микроконтроллерами.

Команды и данные посылают Ethernet-чипу на ножку SI с частотой сигнала SCK. Считывание происходит по фронту импульса. Вывод данных

происходит через SO с частотой сигнала SCK, но уже по спаду импульса. Пока выполняется обмен данными на CS должно присутствовать напряжение L, и переводиться в состояние H только по завершению обмена. Описанные ситуации представлены на рисунках 19, 20.

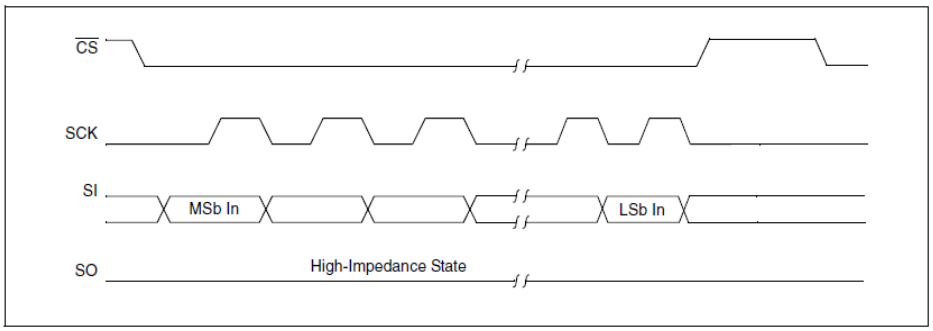


Рисунок 19 – Состояние выводов SPI ENC28J60 при приеме данных

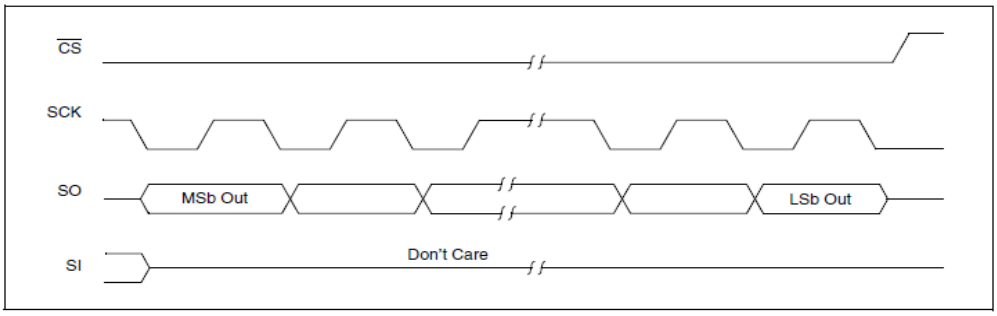


Рисунок 20 – Состояние выводов SPI ENC28J60 при передаче данных

Работа ENC28J60 полностью зависит от команд, посылаемых ATmega168 через SPI интерфейс. Эти команды принимают форму инструкций, одного или нескольких байтов, которые используются для доступа к управлению памятью и пространства Ethernet буфера. Инструкции состоят из трехбитного кода операции, пятибитного аргумента, который указывает или адрес регистра, или постоянные данные, а некоторые содержат еще и сами данные. Все семь инструкций для ENC28J60 приведены в таблице 14.

Таблица 14 – SPI-инструкции для ENC28J60
(a – биты регистра управления, d – биты данных)

Имя инструкции	Байт 0		Байт 1 и последующие
	Код	Аргумент	Данные
Чтение регистров управления (RCR)	0 0 0	a a a a a	Нет
Чтение буферной памяти (RBM)	0 0 1	1 1 0 1 0	Нет

Продолжение таблицы 14

Запись регистра управления(WCR)	0 1 0	a a a a a	d d d d d d d d
Запись буферной памяти (WBM)	0 1 1	1 1 0 1 0	d d d d d d d d
Установка битового поля (BFS)	1 0 0	a a a a a	d d d d d d d d
Очистка битового поля (BFC)	1 0 1	a a a a a	d d d d d d d d
Команда системного сброса (SRC)	1 1 1	1 1 1 1 1	Нет

Для чтения управляющих регистров Ethernet-чипа ATmega168 отправляет ENC28J60 команду чтения регистра и забирает значение. При чтении регистров MAC или MII, микроконтроллер должен пропустить 1 «ложный» байт, затем прочитать значение. Для записи регистров ATmega168 сначала отправляет команду записи регистра, затем – значение, которое требуется записать в регистр. Функции для чтения/записи регистров приведены ниже и включены в библиотеку enc28j60.c. Для их использования необходимо в главном файле программы main.c с помощью операции #include <enc28j60.c> подключить вышеприведенную библиотеку и вызвать эти функции.

```
uint8_t enc28j60_read_op(uint8_t cmd, uint8_t adr) // Операция чтения
 {
  uint8_t data;    enc28j60_select();   // Низкий уровень на CS
  enc28j60_tx(cmd | (adr & 0x1f)); // Отправляем команду
  if(adr & 0x80) // При необходимости, пропускаем "ложный" байт
  enc28j60_rx();
  data = enc28j60_rx();    // Читаем данные
  enc28j60_release();// Высокий уровень на ножке CS
  return data;
 }
void enc28j60_write_op(uint8_t cmd, uint8_t adr, uint8_t data) // Операция записи
 {
  enc28j60_select();
  enc28j60_tx(cmd | (adr & 0x1f));     // Отправляем команду
  enc28j60_tx(data); // Отправляем значение
  enc28j60_release();
}
```

Аргумент uint8_t cmd – код операции, uint8_t adr – адрес регистра, uint8_t data – значение, записываемое в регистр. Работать с номерами адресов регистров и кодов команд неудобно, поэтому с помощью макроподстановки #define приводят соответствие названий регистров их адресам и названий команд их кодам:

```
#define ERDPTL     0x00
#define ERDPTH     0x01
……….
#define ENC28J60_SPI_RCR    0x00
#define ENC28J60_SPI_WCR   0x40
#define ENC28J60_SPI_BFS     0x80
#define ENC28J60_SPI_BFC     0xA0
……..
```

Вышеприведенные макроподстановки помещены в библиотеку enc28j60.h, которую нужно подключать в файле программы, где макроподстановки будут использоваться. Теперь в функциях чтения/записи регистров вместо аргументов uint8_t cmd и uint8_t adr можно записывать соответственно названия регистров и названия операций.

Перед тем, как осуществлять доступ к определённому регистру, нужно выбрать банк. Чтобы не отправлять команду переключения банка при каждой операции с регистром, можно кэшировать текущий банк и переключать его только при необходимости. Функции выбора банка, чтения/записи регистров с учетом выбора банка приведены ниже и включены в библиотеку enc28j60.c.

```
uint8_t enc28j60_current_bank = 0; // Выбор банка регистров
void enc28j60_set_bank(uint8_t adr)
{
 uint8_t bank;
 if( (adr & ENC28J60_ADDR_MASK) < ENC28J60_COMMON_CR ) // Регистр относится к определённому банку?
  {
   bank = (adr >> 5) & 0x03; //BSEL1|BSEL0=0x03 // Получаем номер банка
   if(bank != enc28j60_current_bank)  // Если выбран "не тот" банк
    {
     enc28j60_write_op(ENC28J60_SPI_BFC, ECON1, 0x03); // Выбираем банк
     enc28j60_write_op(ENC28J60_SPI_BFS, ECON1, bank);
     enc28j60_current_bank = bank;
    }
   }
  }
uint8_t enc28j60_rcr(uint8_t adr) // Чтение регистра
 {
```

```
   enc28j60_set_bank(adr);
   return enc28j60_read_op(ENC28J60_SPI_RCR, adr);
  }
uint16_t enc28j60_rcr16(uint8_t adr) // Чтение пары регистров (L и H)
 {
   enc28j60_set_bank(adr);
   return enc28j60_read_op(ENC28J60_SPI_RCR, adr) |
   (enc28j60_read_op(ENC28J60_SPI_RCR, adr+1) << 8);
  }
void enc28j60_wcr(uint8_t adr, uint8_t arg) // Запись регистра
 {
   enc28j60_set_bank(adr);
   enc28j60_write_op(ENC28J60_SPI_WCR, adr, arg);
  }
void enc28j60_wcr16(uint8_t adr, uint16_t arg) // Запись пары регистров (L и H)
 {
   enc28j60_set_bank(adr);
   enc28j60_write_op(ENC28J60_SPI_WCR, adr, arg);
   enc28j60_write_op(ENC28J60_SPI_WCR, adr+1, arg>>8);
 }
void enc28j60_bfc(uint8_t adr, uint8_t mask) // Очистка битов в регистре (reg[adr] &=
~mask)
 {
  enc28j60_set_bank(adr);
  enc28j60_write_op(ENC28J60_SPI_BFC, adr, mask);
 }
void enc28j60_bfs(uint8_t adr, uint8_t mask) // Установка битов в регистре (reg[adr] |=
mask)
{
  enc28j60_set_bank(adr);
  enc28j60_write_op(ENC28J60_SPI_BFS, adr, mask);
 }
```

Для чтения буфера ATmega168 отправляет команду чтения, затем считывает данные, поступающие на вывод MISO. Завершается операция поднятием линии CS. Функция чтения буфера приведена ниже. Аргумент функции uint8_t *buf – массив, в который помещаются принятые данные, uint16_t len – длина принятых пакетов в байтах.

```
#define ENC28J60_SPI_RBM    0x3A
void enc28j60_read_buffer(uint8_t *buf, uint16_t len) // Чтение данных из буфера (по
адресу в регистрах ERDPT)
 {
   enc28j60_select();
   enc28j60_tx(ENC28J60_SPI_RBM);
   while(len--)
```

```
  *(buf++) = enc28j60_rx();
   enc28j60_release();
 }
```

Для записи в буфер ATmega168 отправляет команду записи, затем данные. Завершается операция поднятием линии CS. Функция записи в буфер приведена ниже. Аргумент функции uint8_t *buf – массив, в который помещаются записываемые данные, uint16_t len – длина данных в байтах.

```
#define ENC28J60_SPI_WBM    0x7A// Запись данных в буфер (по адресу в регистрах EWRPT)
void enc28j60_write_buffer(uint8_t *buf, uint16_t len)
 {
   enc28j60_select();
   enc28j60_tx(ENC28J60_SPI_WBM);
   while(len--)
   enc28j60_tx(*(buf++));
   enc28j60_release();
  }
```

Для выполнения перезагрузки Ethernet-чипа служит команда системного сброса. После сброса необходимо подождать 1 мс, чтобы ENC28J60 мог выполнить внутреннюю инициализацию. Функция сброса приведена ниже и включена в библиотеку enc28j60.c.

```
#define ENC28J60_SPI_SC       0xFF
void enc28j60_soft_reset()
 {
   enc28j60_select(); // Отправляем команду
   enc28j60_tx(ENC28J60_SPI_SC);
   enc28j60_release();    _delay_ms(1);   // Ждём, пока ENC28J60 инициализируется
   enc28j60_current_bank = 0; // Не забываем про банк
  }
```

Так же как и с LCD, начало работы с Ethernet-чипом начинается с его инициализации – начальной настройке. Типичная последовательность инициализации ENC28J60 выглядит следующим образом:

1) Настраиваем размер буфера для приёма данных с помощью регистров ERXST, ERXND; инициализируем указатель для чтения данных ERXRDPT.
2) Настраиваем фильтрацию входящих пакетов. По умолчанию, ENC28J60 пропускает пакеты, приходящие на наш MAC-адрес и широковещательные пакеты. В принципе, можно так и оставить.

3) Настройка MAC:
   - очищаем бит MACON2 регистра MARST чтобы снять сброс MAC;
   - устанавливаем бит MACON1 регистра MARXEN чтобы разрешить приём данных MAC;
   - устанавливаем бит MACON1 регистра RXPAUS и бит MACON1 регистра TXPAUS для включения аппаратного управления потоком;
   - настраиваем биты PADCFG, TXCRCEN в регистре MACON3 для выравнивания пакета до 60 байт и автоматического добавления контрольной суммы;
   - устанавливаем максимальный размер фрейма в регистре MAMXF;
   - устанавливаем размер промежутка между фреймами в регистрах MABBIPG, MAIPGL и MAIPGH;
   - устанавливаем MAC-адрес в регистрах MAADR.

4) Настройка PHY:
   - включаем бит PHCON2 регистра HDLDIS, если не хотим получать свои пакеты обратно в полудуплексном режиме;
   - выбираем, как на различные события будут реагировать светодиоды LEDA и LEDB в регистре PHLCON.

5) Настраиваем дуплексный режим, если хотим переопределить значение, определяемое полярностью светодиода LEDB. Для включения полного дуплекса устанавливаем бит PHCON1 регистра PDPXMD и бит MACON3 регистра FULDPX.

6) Разрешаем приём пакетов, установив бит RXEN регистра ECON1.

```
#define ENC28J60_SPI_DDR    DDRB
#define ENC28J60_SPI_PORT    PORTB
#define ENC28J60_SPI_CS       (1<<PB4)
#define ENC28J60_SPI_MOSI    (1<<PB5)
#define ENC28J60_SPI_MISO    (1<<PB6)
#define ENC28J60_SPI_SCK    (1<<PB7)
#define ENC28J60_BUFSIZE    0x2000
#define ENC28J60_RXSIZE       0x1A00
#define ENC28J60_MAXFRAME    1500
#define ENC28J60_RXSTART    0
#define ENC28J60_RXEND       (ENC28J60_RXSIZE-1)
#define ENC28J60_TXSTART    ENC28J60_RXSIZE
#define ENC28J60_BUFEND       (ENC28J60_BUFSIZE-1)
uint16_t enc28j60_rxrdpt = 0;
void enc28j60_init(uint8_t *macadr)
  {
   ENC28J60_SPI_DDR |=
ENC28J60_SPI_CS|ENC28J60_SPI_MOSI|ENC28J60_SPI_SCK; // Настраиваем SPI
```

```
   ENC28J60_SPI_DDR &= ~ENC28J60_SPI_MISO;
   enc28j60_release();
   SPCR = (1<<SPE)|(1<<MSTR);
   SPSR |= (1<<SPI2X);
   enc28j60_soft_reset();    // Выполняем сброс
// Настраиваем размер буфера для приёма пакетов
enc28j60_wcr16(ERXST, ENC28J60_RXSTART);
enc28j60_wcr16(ERXND, ENC28J60_RXEND);
// Указатель для чтения принятых пакетов
enc28j60_wcr16(ERXRDPT, ENC28J60_RXSTART);
enc28j60_rxrdpt = ENC28J60_RXSTART;
// Настраиваем MAC
 enc28j60_wcr(MACON2, 0); // очищаем сброс
 enc28j60_wcr(MACON1, MACON1_TXPAUS|MACON1_RXPAUS| // включаем
управление потоком
 MACON1_MARXEN); // разрешаем приём данных
 enc28j60_wcr(MACON3, MACON3_PADCFG0| // разрешаем паддинг
 MACON3_TXCRCEN| // разрешаем рассчёт контрольной суммы
 MACON3_FRMLNEN| //разрешаем контроль длины фреймов
 MACON3_FULDPX); // включаем полный дуплекс
 enc28j60_wcr16(MAMXFL, ENC28J60_MAXFRAME); // устанавливаем максималь-
ный размер фрейма
 enc28j60_wcr(MABBIPG, 0x15); // устанавливаем промежуток между фреймами
 enc28j60_wcr(MAIPGL, 0x12);
 enc28j60_wcr(MAIPGH, 0x0c);
 enc28j60_wcr(MAADR5, macadr[0]); // устанавливаем MAC-адрес
 enc28j60_wcr(MAADR4, macadr[1]);
 enc28j60_wcr(MAADR3, macadr[2]);
 enc28j60_wcr(MAADR2, macadr[3]);
 enc28j60_wcr(MAADR1, macadr[4]);
 enc28j60_wcr(MAADR0, macadr[5]);
  // Настраиваем PHY
 enc28j60_write_phy(PHCON1, PHCON1_PDPXMD); // включаем полный дуплекс
 enc28j60_write_phy(PHCON2, PHCON2_HDLDIS); // отключаем loopback
 enc28j60_write_phy(PHLCON, PHLCON_LACFG2| // настраиваем светодиодики
 PHLCON_LBCFG2|PHLCON_LBCFG1|PHLCON_LBCFG0|
 PHLCON_LFRQ0|PHLCON_STRCH);
// разрешаем приём пакетов
 enc28j60_bfs(ECON1, ECON1_RXEN);
}
```

Для отправки пакета, записывают указатели на его начало и конец в регистры ETXST и ETXND. Перед пакетом должен находиться управляющий байт, в котором можно переопределить некоторые настройки MAC для отправки этого пакета (рис. 21).

После отправки пакета, за ним на аппаратном уровне запишется 6-байтовый блок, содержащий статус передачи. Функция отправки пакета

приведена ниже и включена в библиотеку enc28j60.c. Аргумент uint8_t *data – массив, в который помещаются данные для передачи, uint16_t len – размер данных в байтах.

Рисунок 21 – Расположение передаваемых данных в буфере

```
void enc28j60_send_packet(uint8_t *data, uint16_t len)
 {
  while(enc28j60_rcr(ECON1) & ECON1_TXRTS)    // Ждём готовности передатчика
    {
     if(enc28j60_rcr(EIR) & EIR_TXERIF) // При ошибке, сбрасываем передатчик
      {
       enc28j60_bfs(ECON1, ECON1_TXRST);
       enc28j60_bfc(ECON1, ECON1_TXRST);
      }
    }
  // Записываем пакет в буфер
  enc28j60_wcr16(EWRPT, ENC28J60_TXSTART);
  enc28j60_write_buffer((uint8_t*)"\x00", 1);
  enc28j60_write_buffer(data, len);
  // Устанавливаем указатели ETXST и ETXND
  enc28j60_wcr16(ETXST, ENC28J60_TXSTART);
  enc28j60_wcr16(ETXND, ENC28J60_TXSTART + len);
  // Разрешаем отправку
  enc28j60_bfs(ECON1, ECON1_TXRTS);
 }
```

Принятые пакеты записываются в кольцевой буфер ENC28J60 в виде связанного списка (рис. 22). Адрес первого непрочитанного пакета хранится в регистре ERXRDPT. Забрав пакет, микроконтроллер записывает в

ERXRDPT адрес следующего пакета. После этого место, которое занимал пакет, считается свободным и ENC28J60 может использовать его для приёма новых пакетов. Перед пакетом на аппаратном уровне записывается 2-байтовый блок указателя на следующий пакет и 4-байтовый блок статуса приёма. Все принятые пакеты ENC28J60 записывает в буфер с выравниванием на 2 байта. Таким образом, адрес пакета всегда чётный. Для того чтобы забрать принятые данные, необходимо сделать следующее:

– проверить сколько принято пакетов (в регистре EPKTCNT);
– прочитать очередной пакет из буфера (по адресу ERXRDPT);
– записать в ERXRDPT адрес следующего пакета;
– уменьшить значение счётчика пакетов установкой бита ECON2 регистра PKTDEC.

Каждый прочитанный пакет помещается в массив данных.

Рисунок 22 – Расположение принимаемых данных в буфере

Функция приема пакетов приведена ниже и включена в библиотеку enc28j60.c. Аргумент uint8_t *data – массив, в который помещаются принятые данные, uint16_t len – размер данных в байтах.

```
// "Правильное" значение ERXRDPT
uint16_t enc28j60_rxrdpt = 0;
uint16_t enc28j60_recv_packet(uint8_t *buf, uint16_t buflen)
 {
  uint16_t len = 0, rxlen, status, temp;
```

```
  // Есть ли принятые пакеты?
  if(enc28j60_rcr(EPKTCNT))
   {
    // Считываем заголовок
    enc28j60_wcr16(ERDPT, enc28j60_rxrdpt);
    enc28j60_read_buffer((void*)&enc28j60_rxrdpt, sizeof(enc28j60_rxrdpt));
    enc28j60_read_buffer((void*)&rxlen, sizeof(rxlen));
    enc28j60_read_buffer((void*)&status, sizeof(status));
    // Пакет принят успешно?
    if(status & 0x80)
     {
      // Выбрасываем контрольную сумму
      len = rxlen - 4;
     // Читаем пакет в буфер (если буфера не хватает, пакет обрезается)
      if(len > buflen) len = buflen;
      enc28j60_read_buffer(buf, len);
      }
    // Устанавливаем ERXRDPT на адрес следующего пакета - 1
    temp = (enc28j60_rxrdpt - 1) & ENC28J60_BUFEND;
    enc28j60_wcr16(ERXRDPT, temp);
   // Уменьшаем счётчик пакетов
    enc28j60_bfs(ECON2, ECON2_PKTDEC);
   }
    return len;
}
```

Передачу данных от одного прикладного процесса к другому можно описать моделью взаимодействия открытых систем OSI, состоящей из семи уровней, определяющих функциональность протоколов обмена данными. На каждом из уровней описывается функция, выполняемая при передаче данных между приложениями, взаимодействующими по сети.

Таким образом с помощью модели OSI разделяют процессы, участвующие в сеансе связи, на четко различающиеся функциональные уровни: физический, канальный, сетевой, транспортный, сеансовый, представления, прикладной.

На физическом уровне определяются характеристики аппаратного обеспечения, необходимого для осуществления передачи данных: уровни напряжения, количество и расположение контактов интерфейсов.

На канальном уровне происходит проверка достоверности принятых по физическому каналу данных. На передающей стороне данные упаковывают в кадры. Кадр представляет собой структуру данных, специфическую для канального уровня; эта структура содержит информацию, доста-

точную для успешной передачи данных по физическому каналу к точке приема.

Как правило, функциональность физического и канального уровней реализуется сетевым адаптером, установленным внутри компьютера.

На сетевом уровне устанавливается маршрут между отправителем и получателем. Используются IP-протокол и IP -адреса. Логическая адресация происходит независимо от физической второго уровня.

На транспортном уровне, как и на канальном, проверяется достоверность принятых данных. Гарантированная доставка происходит при выполнении двух условий. Во-первых, узел-отправитель должен получить подтверждение того, что каждый пакет был принят узлом-получателем без изменений. Во-вторых, перед тем как подтверждать прием пакета, узел-получатель должен проверить целостность его содержимого.

Тем не менее, эта функция выполняется за пределами местного сегмента локальной сети. Здесь обнаруживаются пакеты, отвергнутые маршрутизатором, и автоматически генерируется запрос на повторную передачу. На транспортном уровне обеспечивается надежность сквозной передачи.

Другая важная функция транспортного уровня – упорядочивание сегментов поступивших данных. Для этого необходимо определить исходную последовательность сегментов и расположить их последовательно перед тем, как передавать содержимое на следующий уровень.

Основная функция, описываемая на сеансовом уровне OSI, – организация связи, то есть организация сеанса передачи и его управление во время связи двух систем. Сеанс определяет направленность передачи данных, а также гарантирует завершение обработки одного запроса до принятия следующего. На сеансовом уровне также могут поддерживаться некоторые из следующих дополнений: управление диалогом (выбирается двусторонний – дуплексный или односторонний – полудуплексный режим обмена данными), управление маркерами (определяется на какой стороне находится маркер и как он передается между двумя сторонами), управление операциями (создается контрольные точки – точки синхронизации для восстановления передачи с прерванного места).

Представительский уровень содержит службы преобразования данных для программного обеспечения более высокого уровня. Ранее функции на этом уровне обеспечивали перекодировку символов (например, из EBCDIC в ASCII), сейчас они обеспечивают также шифрование и дешифрирование данных. К этим службам, в частности, относятся Secure Socket Layer фирмы Netscape и PCT API фирмы Microsoft.

На прикладном уровне объединяются предназначенные для пользователей системы или приложений сетевые службы высокого уровня, а имен-

но: служба файлов и печати, службы Web, служба FTP, служба регистрации на сетевом сервере. Этот уровень может рассматриваться как сторона, непосредственно инициирующая сеанс связи.

Структура уровней соответствует естественной последовательности событий, происходящих во время сеанса связи. На рисунке 23 видно, что в процессе передачи, к данным на каждом уровне прибавляется свой заголовок, содержащий служебную информацию протоколов соответствующих уровней; этот процесс называется инкапсуляцией данных.

Инкапсуляция повторяется до тех пор, пока данные не передадутся физическому уровню, с которого они передаются в среду передачи. На получающем устройстве заголовки удаляются, по мере продвижения сообщения вверх по уровням OSI-модели. Таким образом, сообщение достигает получающего процесса или приложения.

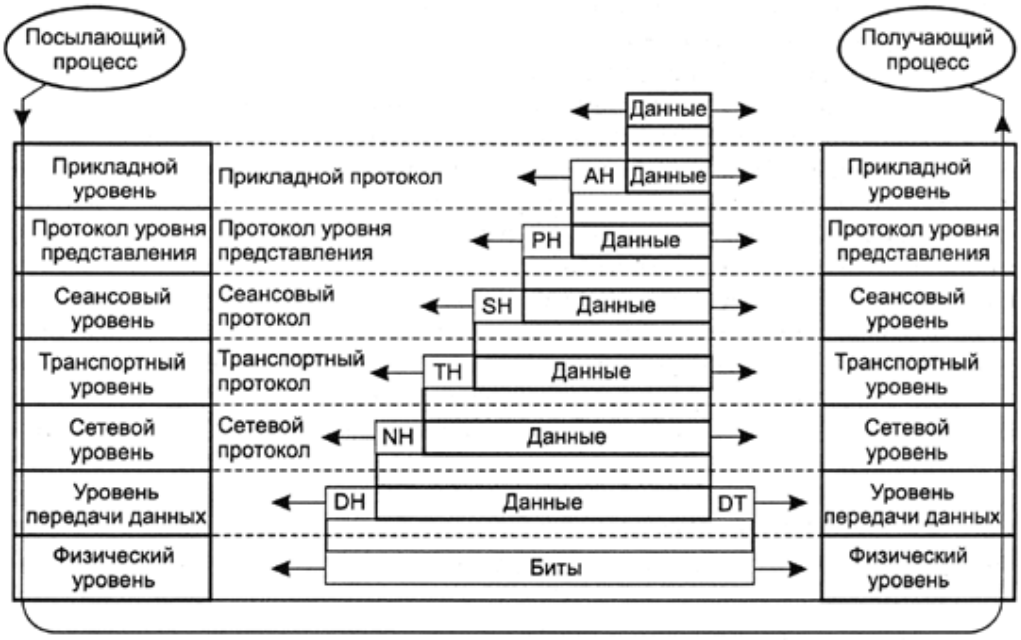


Рисунок 23 – Пример использования модели OSI при описании взаимодействия протоколов различных уровней

Основная идея данной модели заключается в том, что хотя в действительности продвижение данных (рис. 23) осуществляется по «вертикали», каждый уровень запрограммирован так, будто бы перемещение данных было «горизонтальным», то есть от транспортного уровня к транспортному, от сетевого – к сетевому и так далее.

Установлением соответствия между IP-адресом и физическим адресом занимается протокол канального уровня ARP. ARP-протокол ведет таблицу соответствия IP и физических адресов (таблица 15), которая

обычно строится динамически, хотя имеет место добавление в нее статических записей.

Таблица 15 – Таблица ARP или кэш ARP

	Физический порт	Физический адрес	IP-адрес	Тип
Запись 1				
Запись 2				
Запись n				

ARP также имеет кэш записей, называемый кэшем ARP. Обычно поиск начинается с кэша ARP и только в случае неудачи данные ищутся в таблице ARP. Динамически сгенерированные записи кэша ARP становятся недействительными при истечении интервала тайм-аута, тогда как на статические записи тайм-аут не распространяется. Каждая строка кэша ARP или таблицы соответствует одному устройству, и содержит 4 атрибута: физический порт, физический адрес, IP-адрес, тип.

Поле типа записи принимает четыре возможных значения: «2» – запись недействительна; «3» – соответствие устанавливается; «4» – статическое соответствие; «1» – ни один из перечисленных вариантов.

При получении IP-адреса, для которого нужно найти соответствующий физический адрес, ARP просматривает кэш и таблицу ARP в поисках совпадения. Если совпадение будет найдено, ARP возвращает физический адрес стороне, от которой был получен IP-адрес. Если информация не обнаружена, в сеть отправляется запрос ARP (таблица 16) – специальное широковещательное сообщение, получаемое всеми устройствами локальной сети.

Таблица 16 – Структура запросов и ответов ARP (RARP)

Тип оборудования (16 бит)	
Тип протокола (16 бит)	
Длина аппаратного адреса	Длина протокольного адреса
Код операции (16 бит)	
Аппаратный адрес отправителя	
IP- адрес отправителя	
Аппаратный адрес получателя	
IP-адрес получателя	

Запрос ARP содержит IP-адрес предполагаемого получателя. Если устройство опознает IP-адрес как принадлежащий ему, оно отправляет от-

ветное сообщение со своим физическим адресом тому компьютеру, который сгенерировал запрос ARP. Полученная информация сохраняется в таблице и кэше ARP для будущего использования. Таким образом, ARP может определить физический адрес любого компьютера на основании его IP-адреса. При отправке запроса используются все поля структуры, кроме поля аппаратного адреса получателя. В ответах ARP используются все поля. Поле «тип оборудования» определяет тип аппаратного интерфейса. Например, 1- Ethernet, 21 – ATM. Поле «тип протокола» определяет тип протокола, используемого устройством-отправителем. Например, 2048 – IP-протокол, 2054 – ARP-протокол, 32821 – RARP-протокол. Поле «код операции» определяет, является дейтаграмма запросом (если равно 1) или ответом ARP (если равно 2).

Интернет протокол IP относится к протоколам сетевого уровня. Заголовок IP (таблица 17) содержит маршрутную и управляющую информацию, связанную с доставкой дейтаграмм.

Таблица 17 – Структура заголовка IP

<table>
<tr><td colspan="6">32 бита</td></tr>
<tr><td colspan="2">4 бита</td><td>4 бита</td><td>8 бит</td><td colspan="2">6 бит</td></tr>
<tr><td colspan="2">Версия</td><td>Длина заголовка</td><td>Тип службы</td><td colspan="2">Общая длина</td></tr>
<tr><td colspan="4">Идентификатор</td><td>Флаги</td><td>Смещение фрагмента</td></tr>
<tr><td colspan="2">Срок жизни</td><td colspan="2">Протокол</td><td colspan="2">Контрольная сумма</td></tr>
<tr><td colspan="6">Адрес отправителя</td></tr>
<tr><td colspan="6">Адрес получателя</td></tr>
<tr><td colspan="6">Параметры и заполнители</td></tr>
</table>

Поле «версия» определяет версию протокола IP (4 или 6) и формат заголовка. По содержимому поля «длина заголовка» получатель определяет, когда нужно прекратить чтение заголовка и начать чтение данных. Длина заголовка измеряется в тридцатидвухбитовых словах. Минимальный размер заголовка составляет пять слов. Поле «тип службы» показывает желаемый уровень качества обслуживания. Сети могут обеспечивать различный уровень преимуществ при доставке, играющий важную роль при условиях высокой загрузки сети. Поддерживаются также три опции качества обслу-

живания – малая задержка, высокая надежность и высокая пропускная способность. Поле «общая длина» содержит значение размера дейтаграммы в байтах с учетом заголовка и данных. Поле «идентификатор» содержит значение идентификатора, которое отправитель задает для обеспечения корректного порядка сборки фрагментов дейтаграммы на приемной стороне. Поле «флаги» содержит трехбитовое поле флагов управления. Поле «смещение фрагмента» содержит тринадцатибитовое значение, задающее смещение фрагмента от начала целой дейтаграммы. Смещение фрагмента измеряется в восьмибайтовых словах. Первый фрагмент имеет нулевое смещение. Поле «срок жизни» показывает максимальное время существования дейтаграммы в сети. При нулевом значении этого поля дейтаграмма должна быть уничтожена. Время жизни дейтаграмм измеряется в секундах. Однако, поскольку каждый модуль, работающий с дейтаграммой, должен уменьшать значение поля по крайней мере на один (даже в тех случаях, когда обработка дейтаграммы занимает меньше секунды), значение этого поля должно быть не меньше желаемого времени жизни дейтаграммы. Дейтаграммы с истекшим в процессе доставки временем жизни не попадают к получателю. Поле «протокол» указывает протокол следующего уровня, содержащийся в поле данных дейтаграммы IP.

Протокол управления сообщениями ICMP используется для передачи сообщений об ошибках и других исключительных ситуациях, возникших при передаче данных, например, запрашиваемая услуга недоступна, или хост, или маршрутизатор не отвечают. Также на ICMP возлагаются некоторые сервисные функции. Формат ICMP пакета представлен таблицей 18.

Таблица 18 – Формат ICMP пакета

Бит	0-7	8-15	16-31
0	Тип	Код	Контрольная сумма
32	Содержание сообщения (зависит от значений полей «Код» и «Тип»)		

Утилита «Ping» использует Echo Request/Echo Reply (запрос/ответ) сообщения протокола ICMP, при этом в поле «тип» должно быть записано число 8 при запросе и число 0 при ответе. В поле «код» должна быть записано число 0.

Напишем программу для микроконтроллера, позволяющую устройству отвечать на ARP и ICMP-запросы. Для начала необходимо создать 2 файла-библиотеки для ENC28J60: enc28j60.h и enc28j60.c. В файл

enc28j60.c поместим функции инициализации ENC28J60, приема/передачи данных по SPI, записи/чтения регистров и буфера ENC28J60, выбора банка регистров, отправки и приема пакетов (приложение Б). В файле enc28j60.h должны быть объявлены все эти функции, размещены все регистры ENC28J60 и их адреса, команды управления ENC28J60 и соответствующие им значения.

При обмене информацией по протоколам ARP и ICMP, важно знать расположение полей заголовков каждого уровня в потоке данных. Для этих целей служит программа в файле net.h, куда и помещена данная информация:

```
#ifndef NET_H
#define NET_H
#define ETH_HEADER_LEN      14 //длина заголовка ethernet - 14 байт, начиная с 0 байта данных
#define ETHTYPE_ARP_H_V 0x08 // значение первого байта  поля "Тип" - 8
#define ETHTYPE_ARP_L_V 0x06 // и значение второго байта поля "Тип"-6 - ARP
#define ETHTYPE_IP_H_V  0x08 // значение первого байта  поля "Тип" - 8
#define ETHTYPE_IP_L_V  0x00 // и значение второго байта поля "Тип"-0 - IP
#define ETH_TYPE_H_P 12 //поле "тип" начинается с 12 байта данных
#define ETH_TYPE_L_P 13 // второй бит поля "тип" -13 байт данных
#define ETH_DST_MAC 0 // поле "мас-адрес получателя" начинается с 0 байта данных
#define ETH_SRC_MAC 6 // поле "мас-адрес источника" начинается с 6 байта данных
#define ETH_ARP_OPCODE_REPLY_H_V 0x00 //значение старшего байта поля "кода операции" 0 - arp
#define ETH_ARP_OPCODE_REPLY_L_V 0x02 //и значение младшего байта поля "кода операции" 2 - arp - ответ
#define ETHTYPE_ARP_L_V 0x06 // значение второго байта поля "Тип"-6 - ARP
#define ETH_ARP_DST_IP_P 0x26 //поле "IP-адрес получателя" начинается с 38 байта
#define ETH_ARP_OPCODE_H_P 0x14 //поле "код операции" начинается с 20 байта
#define ETH_ARP_OPCODE_L_P 0x15 // 2-й байт поля "код операции" - 21 байт
#define ETH_ARP_SRC_MAC_P 0x16 //поле "MAC-адрес источника" начинается с 22 байта
#define ETH_ARP_SRC_IP_P 0x1c //поле "IP-адрес источника" начинается с 28 байта
#define ETH_ARP_DST_MAC_P 0x20 //поле "MAC-адрес получателя" начинается с 32 байта
#define ETH_ARP_DST_IP_P 0x26 //поле "IP-адрес получателя" начинается с 38 байта
#define IP_HEADER_LEN 20
#define IP_SRC_P 0x1a //ip-адрес источника начинается с 26-го байта данных
#define IP_DST_P 0x1e //ip-адрес получателя начинается с  30-го байта данных
#define IP_HEADER_LEN_VER_P 0xe
#define IP_CHECKSUM_P 0x18 //поле "контрольная сумма" начинается с  24-го байта данных
#define IP_TTL_P 0x16 //22-й бит данных
#define IP_FLAGS_P 0x14 // поле "флаги" начинается с 20-го байта данных
#define IP_P 0xe
```

#define IP_TOTLEN_H_P 0x10 //поле "общая длина" начинается с 16-го байта данных
#define IP_TOTLEN_L_P 0x11 //поле "общая длина" - 17-й байт данных
#define IP_PROTO_P 0x17 //поле "протокол" начинается с 23-го байта данных
#define IP_PROTO_ICMP_V 1 // значение поля "протокол" равно 1 - icmp протокол
#define IP_PROTO_TCP_V 6 // значение поля "протокол" равно 6 - tcp протокол
#define IP_PROTO_UDP_V 17 // значение поля "протокол" равно 17 - udp протокол
#define ICMP_TYPE_ECHOREPLY_V 0 //значение поля "тип" равно 0 - пинг-ответ
#define ICMP_TYPE_ECHOREQUEST_V 8 //значение поля "тип" равно 8 - пинг-запрос
#define ICMP_TYPE_P 0x22 // поле "тип" начинается с 34 байта данных
#define ICMP_CHECKSUM_P 0x24 //поле "контрольная сумма" начинается с 36 байта данных
#endif

Для анализа принятых данных, а также генерации фреймов и пакетов в ответ на запросы ARP и ICMP, соответствующие функции помещены в файл программы ip_arp_icmp.c:

```
#include <avr/io.h>
#include "net.h"
#include "enc28j60.h"
static uint8_t macaddr[6];
static uint8_t ipaddr[4];
uint16_t checksum(uint8_t *buf, uint16_t len);
void init_ip_arp(uint8_t *mymac,uint8_t *myip);
uint8_t eth_type_is_arp_and_my_ip(uint8_t *buf,uint16_t len);
uint8_t eth_type_is_ip_and_my_ip(uint8_t *buf,uint16_t len);
void make_eth(uint8_t *buf);
void fill_ip_hdr_checksum(uint8_t *buf);
void make_ip(uint8_t *buf);
void make_arp_answer_from_request(uint8_t *buf);
void make_echo_reply_from_request(uint8_t *buf,uint16_t len);
uint16_t checksum(uint8_t *buf, uint16_t len) // вычисляем контрольную сумму для ip заголовка
 {
  uint32_t sum = 0;
  while(len >1)
    {
     sum += 0xFFFF & (*buf<<8|*(buf+1));
     buf+=2;
     len-=2;
    }
   if (len)
    {
     sum += (0xFF & *buf)<<8;
    }
  while (sum>>16)
```

```
    {
      sum = (sum & 0xFFFF)+(sum >> 16);
     }
   return( (uint16_t) sum ^ 0xFFFF);
  }
```
void init_ip_arp(uint8_t *mymac,uint8_t *myip) // функция генерации ip- и MAC-адресов, которые будут переданы в пакет и фрейм соответственно
```
 {
  uint8_t i=0;
  while(i<4)
   {
     ipaddr[i]=myip[i];
```
// присваиваем ip-адресу, который впоследствии будет отправлен в ip- пакете значение нашего ip
```
      i++;
    }
   i=0;
   while(i<6)
    {
     macaddr[i]=mymac[i];
```
// присваиваем MAC-адресу, который впоследствии будет отправлен в Ethernet-фрейме значение нашего MAC
```
      i++;
     }
   }
```
uint8_t eth_type_is_arp_and_my_ip(uint8_t *buf,uint16_t len) // функция проверки, является ли полученный пакет ARP-запросом и предназначен ли нам?
```
{
 uint8_t i=0;
 if (len<41)
  {
    return(0);
  }
 if(buf[ETH_TYPE_H_P] != ETHTYPE_ARP_H_V || buf[ETH_TYPE_L_P] != ETHTYPE_ARP_L_V)
  {
   return(0);
  }
 while(i<4)
  {
   if(buf[ETH_ARP_DST_IP_P+i] != ipaddr[i])
    {
     return(0);
    }
    i++;
  }
  return(1);
```
//в случае успеха возвращает значение 1
```
}
```
uint8_t eth_type_is_ip_and_my_ip(uint8_t *buf,uint16_t len) // функция проверки, адресован ли полученный IP пакет нам?

```
 {
uint8_t i=0;
if (len<42)
 {
  return(0);
 }
 if(buf[ETH_TYPE_H_P]!=ETHTYPE_IP_H_V                                    ||
buf[ETH_TYPE_L_P]!=ETHTYPE_IP_L_V)
  {
   return(0);
  }
  if (buf[IP_HEADER_LEN_VER_P]!=0x45)
   {
    return(0);
   }
 while(i<4)
   {
    if(buf[IP_DST_P+i]!=ipaddr[i])
     {
      return(0);
     }
    i++;
   }
 return(1); //если да, то возвращает 1
}
void make_eth(uint8_t *buf) //функция замены MAC-адреса источника на MAC-адрес
получателя
{
 uint8_t i=0;
 while(i<6)
   {
    buf[ETH_DST_MAC +i]=buf[ETH_SRC_MAC +i];
    buf[ETH_SRC_MAC +i]=macaddr[i];
    i++;
   }
}
void fill_ip_hdr_checksum(uint8_t *buf) // функция генерации контрольной суммы,
флагов, времени жизни для отправляемого IP пакета
 {
  uint16_t ck;
  buf[IP_CHECKSUM_P]=0;
  buf[IP_CHECKSUM_P+1]=0;
  buf[IP_FLAGS_P]=0x40;
  buf[IP_FLAGS_P+1]=0;
  buf[IP_TTL_P]=64;
  ck=checksum(&buf[IP_P], IP_HEADER_LEN);
  buf[IP_CHECKSUM_P]=ck>>8;
  buf[IP_CHECKSUM_P+1]=ck& 0xff;
```

```
 }
void make_ip(uint8_t *buf) //функция генерации ip пакета для отправления
 {
  uint8_t i=0;
  while(i<4)
    {
     buf[IP_DST_P+i]=buf[IP_SRC_P+i];
     buf[IP_SRC_P+i]=ipaddr[i];
     i++;
    }
  fill_ip_hdr_checksum(buf);
 }
void make_arp_answer_from_request(uint8_t *buf) // функция генерации ARP-ответа для отправления
 {
  uint8_t i=0;
  make_eth(buf);
  buf[ETH_ARP_OPCODE_H_P]=ETH_ARP_OPCODE_REPLY_H_V;
  buf[ETH_ARP_OPCODE_L_P]=ETH_ARP_OPCODE_REPLY_L_V;
  while(i<6)
    {
     buf[ETH_ARP_DST_MAC_P+i]=buf[ETH_ARP_SRC_MAC_P+i];
     buf[ETH_ARP_SRC_MAC_P+i]=macaddr[i];
     i++;
    }
  i=0;
  while(i<4)
    {
     buf[ETH_ARP_DST_IP_P+i]=buf[ETH_ARP_SRC_IP_P+i];
     buf[ETH_ARP_SRC_IP_P+i]=ipaddr[i];
     i++;
    }
  enc28j60PacketSend(42,buf);
  }
void make_echo_reply_from_request(uint8_t *buf,uint16_t len) // функция генерации понг-ответа для отправления
 {
  make_eth(buf);
  make_ip(buf);
  buf[ICMP_TYPE_P]=ICMP_TYPE_ECHOREPLY_V;
  if (buf[ICMP_CHECKSUM_P] > (0xff-0x08))
   {
    buf[ICMP_CHECKSUM_P+1]++;
   }
  buf[ICMP_CHECKSUM_P]+=0x08;
  enc28j60PacketSend(len,buf);
 }
```

Конечная программа с использованием вышеописанных функций размещается в файле main.c и выглядит следующим образом:

```
#include <util/delay.h>
#include <avr/io.h>
#include "ip_arp_icmp.c"
#include "enc28j60.h"
#include "enc28j60.c"
#include "net.h"
static uint8_t mymac[6] = {0x54,0x55,0x58,0x10,0x00,0x24}; // прописываем MAC-адрес устройства
static uint8_t myip[4] = {192,168,0,4}; // прописываем IP-адрес устройства
#define BUFFER_SIZE 400 // указываем размер буфера для приема данных
static uint8_t buf[BUFFER_SIZE+1]; //инициализируем массив данных для буфера
int main(void) // главная функция программы
 {
   PORTD=0x00;
   DDRD|=0xff;
   uint16_t plen;
   CLKPR=(1<<CLKPCE);
   CLKPR=0; // частота 8Мгц
   _delay_ms(10);
   enc28j60Init(mymac); //запускаем функцию инициализации ENC28J60
   _delay_ms(20);
   enc28j60PhyWrite(PHLCON,0x476);  //настраиваем режим работы светодиодов ENC28J60: один светодиод индицирует статус соединения,другой-передачу и прием
   _delay_ms(20);
   init_ip_arp(mymac,myip); //запускаем функцию присваивания MAC- адреса и IP-адреса для последующей передачи
   while(1)
    {
      plen = enc28j60PacketReceive(BUFFER_SIZE, buf); //запускаем функцию приема пакетов от ENC28J60
      if(plen==0) // если принятых данных нет
       {
         continue; //то перейти на начало цикла while
       }
      if(eth_type_is_arp_and_my_ip(buf,plen))  // если полученные данные ARP-запрос,адресованный нам
       {
         make_arp_answer_from_request(buf); //то сгенерировать и отправить ARP-ответ
         continue; // перейти на начало цикла while
       }
      if(eth_type_is_ip_and_my_ip(buf,plen)==0) // если полученные данные являются ip-пакетом и адресованы не нам
       {
         continue; // перейти на начало цикла while
```

```
        }
if(buf[IP_PROTO_P]==IP_PROTO_ICMP_V&&buf[ICMP_TYPE_P]==ICMP_TYPE
_ECHOREQUEST_V) //если в полученном ip-пакете содержится запрос "пинг"
        {
          make_echo_reply_from_request(buf,plen); // отправим ответ "понг"
          continue; //перейти на начало цикла while
        }
      }
    return (0);
   }
```

Если, после прошивки программы в микроконтроллер, подключить его к сети с номером 192.168.0.0 и в командной строке компьютера из этой сети запустить команду «ping 192.168.0.4», то получим ответ от устройства (рис. 24)

Проверить устройство на обмен данными с ПК по ARP протоколу можно, запустив команду «arp –a» . Отобразится таблица соответствия MAC-адресов, IP-адресам (рис. 25).

Задание: на основе теоретических знаний и примеров программ, изложенных в данной лабораторной работе написать программу для микроконтроллера, осуществляющую ответы посредством Ethernet интерфейса на ARP, ICMP(пинг)-запросы. В программе установить MAC адрес, равный 84:85:88:16:0:36 и IPv4 адрес 10.24.0.112, время жизни IP пакета задать равным 128.

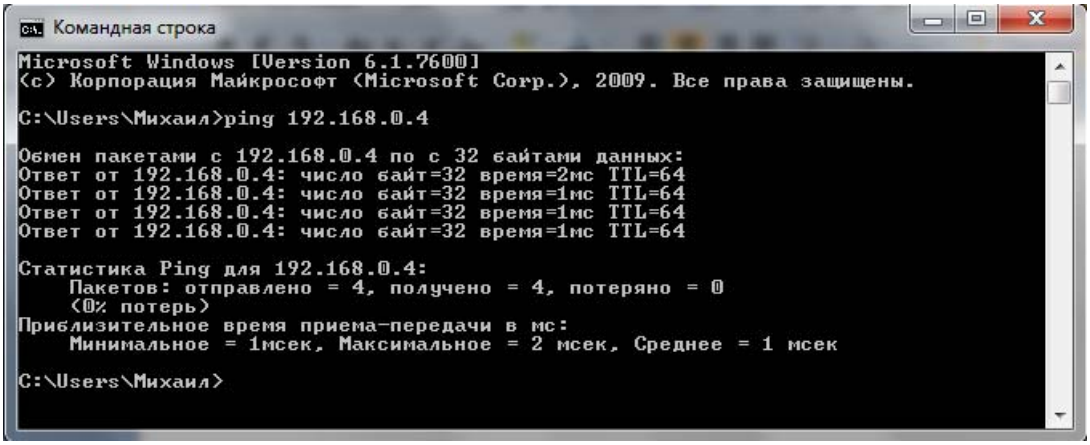


Рисунок 24 – Ответ от устройства при ping-запросе

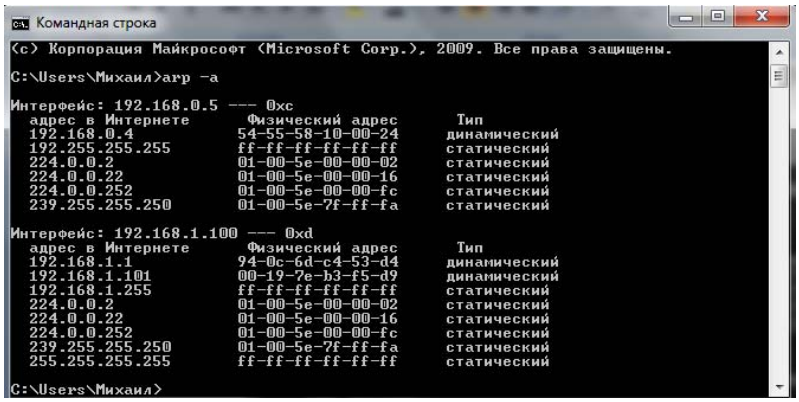


Рисунок 25 – Проверка реализации ARP-протокола на устройстве

Контрольные вопросы

1. Что такое модель взаимодействия открытых систем OSI? Какие функциональные уровни она содержит?
2. Как происходит передача/прием данных согласно модели OSI?
3. Для чего предназначен протокол ARP? Опишите структуру данных запросов и ответов протокола ARP.
4. Для чего предназначен протокол ICMP? Какой формат ICMP пакета?
5. Для чего предназначен микроконтроллер ENC28J60?
6. Какой интерфейс служит для сопряжения микроконтроллеров ATmega168 и ENC28J60?
7. Каким образом осуществляется управление микроконтроллером ENC28J60?
8. Как происходит передача данных от ENC28J60 по ethernet-интерфейсу?
9. Опишите структуру буфера данных ENC28J60. Почему буфер приема называют кольцевым?

Рекомендуемая литература

1 Гребнев В. В. Микроконтроллеры семейства AVR фирмы Atmel / / В. В. Гребнев. – М.: ИП РадиоСофт, 2002 – 176 с: ил. С. 43−47.

2 Евстифеев А. В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя / А. В. Евстифеев. – М: Издательский дом «Додека-XXI», 2007. − 432 с.: ил. С. 419−427.

3 Подключение микроконтроллера к локальной сети: работаем с ENC28J60. – 2011. – Сайт для разработчиков устройств. – URL:

http://we.easyelectronics.ru /electro-and-pc/podklyuchenie-mikrokontrollera-k-lokalnoy-seti-rabotaem-s-enc28j60.html

4 ATMEGA168 – 8-bit Microcontroller with 8K Bytes In-System Programmable Flash – ATMEL Corporation. – Сайт документации на электронные компоненты. – (Англ.). – URL: http://www.alldatasheet.com/ datasheetpdf/pdf/83753/ATMEL/ATMEGA168.html

5 ENC28J60 – Stand-Alone Ethernet Controller with SPI Product Brief – Microchip Technology. – Сайт документации на электронные компоненты. – (Англ.). – URL: http://www.alldatasheet.com/datasheet-pdf/ pdf/102687/ MICROCHIP/ENC28J60.html

6 Паркер Т. TCP/IP. Для профессионалов. 3-е изд. / Т. Паркер, К. Сиян. – СПб.: Питер. – 2004. – 859 с: ил.

Лабораторная работа №9 «Реализация протокола UDP на микроконтроллере»

Цель: научиться программно, с использованием микроконтроллера, принимать и передавать данные посредством Ethernet-интерфейса по UDP протоколу.

Задачи: изучить UDP протокол, структуру UDP сообщения, разобрать программы, реализующие управление выводами микроконтроллера согласно UDP запросу.

Приборы и принадлежности: плата на основе микроконтроллеров ATmega168 и ENC28J60 , Datasheet к микроконтроллерам ATmega168 и ENC28J60, LPT- программатор, ПК с установленными PonyProg и пакетом программ WinAVR.

Краткие теоретические сведения

Протокол пользовательских дейтаграмм UDP, также как и TCP, является протоколом транспортного уровня модели OSI. Если сравнивать протоколы UDP и TCP, то у каждого есть свои преимущества и недостатки. В отличие от TCP, UDP не пытается обеспечить упорядоченный прием данных, поэтому порядок принятых данных может отличаться от порядка, использовавшегося при отправке. Это вносит ограничения при использовании этого протокола в качестве протокола транспортного уровня: если нужна гарантия доставки данных к получателю в правильном порядке используют протокол TCP. Чаще всего UDP используют приложения, рабо-

тающие по схеме «команда/ответ», причем команды и ответы могут пересылаться в одной дейтаграмме. В этом случае приложение не тратит ресурсы на открытие и закрытие соединений для пересылки малого объема данных, как это делает TCP. Другое преимущество UDP проявляется при широковещательной и групповой рассылке данных, которая нецелесообразна при использовании протокола TCP. Таким образом, протокол UDP обеспечивает ненадежный и не ориентированный на соединение сервис доставки данных, то есть проигрывает TCP в надежности, но выигрывает в скорости передачи данных.

Несмотря на недостатки, на базе UDP построено много достаточно популярных приложений, таких как DNS, NFS, TFTP и других. Структура UDP-дейтаграммы приведена на рисунке 26.

Биты	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0-31	Порт отправителя (Source port)	Порт получателя (Destination port)
32-63	Длина дейтаграммы (Length)	Контрольная сумма (Checksum)
64-...	Данные (Data)	

Рисунок 26 – Структура UDP-дейтаграммы

Каждому прикладному процессу, использующему в качестве протокола транспортного уровня протокол UDP, присваивается уникальный 16-битный номер – номер порта. По этому номеру происходит адресация и доставка UDP пакета к приложениям. Поля «Порт отправителя» и «Порт получателя» как раз содержат этот номер.

Поле «Длина дейтаграммы» содержит длину дейтаграммы UDP в октетах, с учетом как длины заголовка UDP, так и длины данных. Минимальное значение поля длины равно 8 и указывает на то, что поле данных имеет нулевой размер.

Поле «Контрольная сумма» имеет длину 16 бит и вычисляется как с учетом самой дейтаграммы UDP, так и псевдозаголовка, который обеспечивает защиту от неправильной маршрутизации дейтаграмм (Рис 27).

Биты	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0-31	IP-адрес отправителя (Source address)	
32-63	IP-адрес получателя (Destination address)	
64-95	0 0 0 0 0 0 0 0 Протокол (Protocol)	Длина UDP-датаграммы (UDP length)

Рисунок 27 – Структура псевдозаголовка UDP

Псевдозаголовок не присутствует в потоке данных в явном виде, а служит только для расчета контрольной суммы UDP-дейтаграммы. Поля «IP-адрес отправителя», «IP-адрес получателя» и «Протокол» берутся у IP модуля, обрабатывающего данные на сетевом уровне.

В качестве примера рассмотрим программу, управляющую светодиодом, подключенным к выводу PD7 микроконтроллера через UDP-запрос, а так же передающую данные о состоянии светодиода в UDP-ответе.

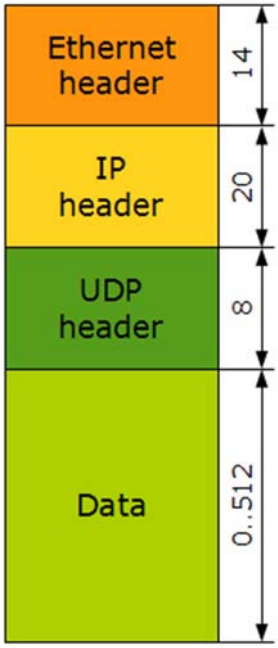


Рисунок 28 – Структура данных, принимаемых микроконтроллером ATmega168 от ENC28J60

Формат данных, принимаемых микроконтроллером ATmega168 по SPI-интерфейсу при реализации UDP-протокола, представлен на рисунке 28 и содержит Ethernet-заголовок, IP-заголовок и UDP-заголовок. Все, что делает ENC28J60, так это проверяет и отбрасывает контрольную сумму Ethernet-фрейма FCS, а также согласно рассмотренным ранее библиотекам enc28j60.h и enc28j60.c, передает данные по SPI интерфейсу микроконтроллеру. При обратной передаче данных от ATmega168 к ENC28J60 структура данных сохраняется, а ENC28J60 вычисляет и добавляет контрольную сумму FCS.

Так как принятые микроконтроллером ATmega168 данные сохраняются в специально созданном буфере, то необходимо знать местоположение в буфере каждого поля каждого заголовка – порядковый номер полей в потоке данных.

Для этого используется файл net.h следующего содержания:

#ifndef NET_H
#define NET_H
//Ethernet-заголовок
#define ETH_HEADER_LEN 14 //длина заголовка ethernet - 14 байт, начиная с 0 байта данных
#define ETHTYPE_ARP_H_V 0x08 // значение первого байта поля "Тип" - 8
#define ETHTYPE_ARP_L_V 0x06 // и значение второго байта поля "Тип"-6 - ARP
#define ETHTYPE_IP_H_V 0x08 // значение первого байта поля "Тип" - 8
#define ETHTYPE_IP_L_V 0x00 //и значение второго байта поля "Тип"-0 - IP
#define ETH_TYPE_H_P 12 //поле "тип" начинается с 12 байта данных
#define ETH_TYPE_L_P 13 // второй бит поля "тип" -13 байт данных
#define ETH_DST_MAC 0 // поле "мас-адрес получателя" начинается с 0 байта данных
#define ETH_SRC_MAC 6 // поле "мас-адрес источника" начинается с 6 байта данных
#define ETH_ARP_OPCODE_REPLY_H_V 0x00 // значение старшего байта поля "кода операции" 0 - arp
#define ETH_ARP_OPCODE_REPLY_L_V 0x02 //и значение младшего байта поля "кода операции" 2 - arp - ответ
#define ETHTYPE_ARP_L_V 0x06 // значение второго байта поля "Тип"-6 - ARP
#define ETH_ARP_DST_IP_P 0x26 //поле "IP-адрес получателя" начинается с 38 байта
#define ETH_ARP_OPCODE_H_P 0x14 //поле "код операции" начинается с 20 байта
#define ETH_ARP_OPCODE_L_P 0x15 // 2-й байт поля "код операции" - 21 байт
#define ETH_ARP_SRC_MAC_P 0x16 //поле "MAC-адрес источника" начинается с 22 байта
#define ETH_ARP_SRC_IP_P 0x1c //поле "IP-адрес источника" начинается с 28 байта
#define ETH_ARP_DST_MAC_P 0x20 //поле "MAC-адрес получателя" начинается с 32 байта
#define ETH_ARP_DST_IP_P 0x26 //поле "IP-адрес получателя" начинается с 38 байта
//IP-заголовок
#define IP_HEADER_LEN 20
#define IP_SRC_P 0x1a //ip-адрес источника начинается с 26-го байта данных
#define IP_DST_P 0x1e //ip-адрес получателя начинается с 30-го байта данных
#define IP_HEADER_LEN_VER_P 0xe
#define IP_CHECKSUM_P 0x18 //поле "контрольная сумма" начинается с 24-го байта данных
#define IP_TTL_P 0x16 //22-й бит данных
#define IP_FLAGS_P 0x14 // поле "флаги" начинается с 20-го байта данных
#define IP_P 0xe
#define IP_TOTLEN_H_P 0x10 //поле "общая длина" начинается с 16-го байта данных
#define IP_TOTLEN_L_P 0x11 //поле "общая длина" - 17-й байт данных
#define IP_PROTO_P 0x17 //поле "протокол" начинается с 23-го байта данных
#define IP_PROTO_ICMP_V 1 // значение поля "протокол" равно 1 - icmp протокол
#define IP_PROTO_TCP_V 6 // значение поля "протокол" равно 6 - tcp протокол
#define IP_PROTO_UDP_V 17 // значение поля "протокол" равно 17 - udp протокол
//ICMP-заголовок
#define ICMP_TYPE_ECHOREPLY_V 0 //значение поля "тип" равно 0 - пинг-ответ
#define ICMP_TYPE_ECHOREQUEST_V 8 // значение поля "тип" равно 8 - пинг-запрос

```
#define ICMP_TYPE_P 0x22 // поле "тип" начинается с 34 байта данных
#define ICMP_CHECKSUM_P 0x24 // поле "контрольная сумма" начинается с 36 байта данных
//UDP-заголовок
#define UDP_HEADER_LEN       8 // Длина заголовка UDP равна 8 байтам
#define UDP_SRC_PORT_H_P 0x22 // Старший байт порта источника- 34 байт данных
#define UDP_SRC_PORT_L_P 0x23 //Младший байт порта источника- 35 байт данных
#define UDP_DST_PORT_H_P 0x24 //Старший байт порта получателя - 36 байт данных
#define UDP_DST_PORT_L_P 0x25 //Младший байт порта источника - 37 байт данных
#define UDP_LEN_H_P 0x26 // Старший байт поля "длина дейтаграммы"-38 байт данных
#define UDP_LEN_L_P 0x27 // Младший байт поля "длина дейтаграммы"-39 байт данных
#define UDP_CHECKSUM_H_P 0x28 // Старший байт поля "контрольная сумма" - 40 байт данных
#define UDP_CHECKSUM_L_P 0x29 // Младший байт поля "контрольная сумма" - 41 байт данных
#define UDP_DATA_P 0x2a // Данные начинаются с 42 байта данных
#endif
```

Этот файл отличается от файла net.h, используемого в предыдущей лабораторной работе только дополнением местоположения полей для UDP-протокола.

Функция расчета контрольной суммы для UDP заголовка получается в результате дополнения некоторыми операциями функции расчета контрольной суммы для ip:

```
uint16_t checksum(uint8_t *buf, uint16_t len,uint8_t type)
 {
  uint32_t sum = 0;
  if(type==1)
   {
    sum+=IP_PROTO_UDP_V;
    sum+=len-8;
   }
  // переведем сумму в 16-битные слова
  while(len >1)
   {
    sum += 0xFFFF & (*buf<<8|*(buf+1));
    buf+=2;
    len-=2;
   }
  // если остался байт слева, дополним его нулями
  if (len)
   {
    sum += (0xFF & *buf)<<8;
```

```
    }
  while (sum>>16)
   {
     sum = (sum & 0xFFFF)+(sum >> 16);
    }
  return( (uint16_t) sum ^ 0xFFFF);
 }
```

Вышеприведенная функция универсальна при применении расчета контрольной суммы IP или UDP: если переменная type равна 1, то идет расчет суммы для UDP заголовка, при любом другом значении – для IP.

Для ответа на UDP-запрос функция, размещенная в файле ip_arp_icmp.c, будет иметь вид:

```
void make_udp_reply_from_request(uint8_t *buf,char *data,uint8_t datalen,uint16_t port) //функция ответа на UDP-запрос
 {
  uint8_t i=0; //объявляем переменную i и приравниваем ее к нулю
  uint16_t ck; //объявляем переменную ck
  make_eth(buf); //заполним ethernet-заголовок
  if (datalen>220) //определим максимальный размер данных для ответа (220 байт)
  {
   datalen=220;
  }
  buf[IP_TOTLEN_H_P]=0; //обнуляем старший байт поля "общая длина" заголовка IP
 buf[IP_TOTLEN_L_P]=IP_HEADER_LEN+UDP_HEADER_LEN+datalen;//устанавливаем младший байт поля "общая длина" заголовка IP
  make_ip(buf); //заполним ip-заголовок
  buf[UDP_DST_PORT_H_P]=port>>8; //установим старший байт поля "порт назначения" заголовка UDP
  buf[UDP_DST_PORT_L_P]=port & 0xff; //установим младший байт поля "порт назначения" заголовка UDP
  buf[UDP_LEN_H_P]=0; //обнуляем старший байт поля "длина дейтаграммы" заголовка UDP
  buf[UDP_LEN_L_P]=UDP_HEADER_LEN+datalen; //устанавливаем младший байт поля "длина дейтаграммы" заголовка UDP
  buf[UDP_CHECKSUM_H_P]=0; //обнуляем поле "контрольная сумма" заголовка UDP
  buf[UDP_CHECKSUM_L_P]=0;
  while(i<datalen) //записываем данные в буфер
   {
    buf[UDP_DATA_P+i]=data[i];
    i++;
   }
  ck=checksum(&buf[IP_SRC_P], 16 + datalen,1); //вычисляем контрольную сумму для UDP-заголовка
```

```
  buf[UDP_CHECKSUM_H_P]=ck>>8; //записываем вычисленную контрольную сумму в буфер
  buf[UDP_CHECKSUM_L_P]=ck& 0xff;

enc28j60PacketSend(UDP_HEADER_LEN+IP_HEADER_LEN+ETH_HEADER_LEN
+datalen,buf); //отправляем ответ UDP
 }
```

Главная функция программы будет выглядеть следующим образом:

```
#include <util/delay.h>
#include <avr/io.h>
#include "ip_arp_icmp.c"
#include "enc28j60.h"
#include "enc28j60.c"
#include "net.h"
static uint8_t mymac[6] = {0x54,0x55,0x58,0x10,0x00,0x24}; // прописываем MAC-адрес устройства
static uint8_t myip[4] = {192,168,1,25}; // прописываем IP-адрес устройства
#define BUFFER_SIZE 400 // указываем размер буфера для приема данных
static uint8_t buf[BUFFER_SIZE+1]; //инициализируем массив данных для буфера
static uint16_t myport=1200; //порт для UDP
int main(void) // главная функция программы
 {
  PORTD=0x00;
  DDRD|=0xff;
  uint16_t plen;
  CLKPR=(1<<CLKPCE);
  CLKPR=0; // частота 8Мгц
  _delay_ms(10);
  enc28j60Init(mymac); //запускаем функцию инициализации ENC28J60
  _delay_ms(20);
  enc28j60PhyWrite(PHLCON,0x476); //настраиваем режим работы светодиодов ENC28J60: один светодиод индицирует статус соединения, другой - передачу и прием
  _delay_ms(20);
  init_ip_arp(mymac,myip); //запускаем функцию присваивания MAC- адреса и IP-адреса для последующей передачи
  while(1)
   {
    plen = enc28j60PacketReceive(BUFFER_SIZE, buf); //запускаем функцию приема пакетов от ENC28J60
    if(plen==0) // если принятых данных нет
    {
     continue; //то перейти на начало цикла while
    }
    if(eth_type_is_arp_and_my_ip(buf,plen)) // если полученные данные являются ARP-запросом, адресованным нам
    {
```

```
      make_arp_answer_from_request(buf); //то сгенерировать и отправить ARP-ответ
      continue; // перейти на начало цикла while
     }
    if(eth_type_is_ip_and_my_ip(buf,plen)==0) // если полученные данные являются ip-пакетом и адресованы не нам
    {
     continue; // перейти на начало цикла while
     }

if(buf[IP_PROTO_P]==IP_PROTO_ICMP_V&&buf[ICMP_TYPE_P]==ICMP_TYPE_ECHOREQUEST_V) //если в полученном ip-пакете содержится запрос "пинг"
    {
     make_echo_reply_from_request(buf,plen); // отправим ответ "понг"
     continue;//перейти на начало цикла while
     }
if(buf[IP_PROTO_P]==IP_PROTO_UDP_V&&buf[UDP_DST_PORT_H_P]==4&&buf[UDP_DST_PORT_L_P]==0xb0) //если в принятых данных тип протокола в заголовке IP UDP и предназначен нам
    {
     char otvet [11]; // объявляем массив символов
     if(buf[UDP_DATA_P]==49) //если получили число 1
      {
       PORTD=0x80; //зажгли светодиод D7
       char otvet[11]="Diod_is_on"; // массив символов ответ равен Diod_is_on
      }
     if(buf[UDP_DATA_P]==48) // если получили число 0
      {
       PORTD=PORTD&(~(1<<7)); // потушили светодиод D7
       char otvet[11]="Diod_is_off"; //массив символов ответ равен Diod_is_off
      }
     make_udp_reply_from_request(buf,otvet,11,myport); // отправили ответ UDP
    }
return (0);
  }
}
```

Таким образом, отправив по UDP протоколу символ «1», включим диод D7, а отправив «0» – выключим его. При этом после каждого запроса будет приходить UDP ответ с подтверждением выполнения данных команд (Diod_is_on или Diod_is_off). Для приема/передачи данных по протоколу UDP, в среде HiAsm написана программа (Рис. 29). В верхней строке программы вводится ip-адрес, на который мы хотим отправить UDP дейтаграмму, в средней – данные для отправки (данные UDP запроса), в нижней – принятые данные (данные UDP ответа).

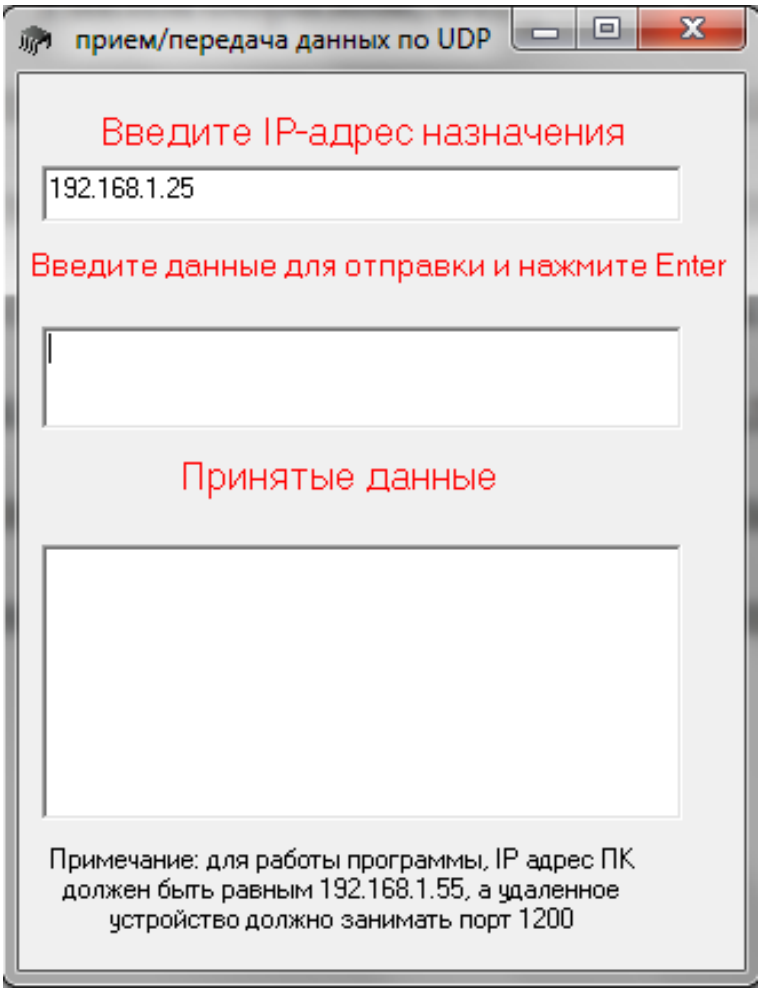


Рисунок 29 – Программа приема/передачи данных по UDP

Задание: на основе теоретических знаний и примеров программ, изложенных в данной лабораторной работе и лабораторной работе №8 написать программу, реализующую управление выводами микроконтроллера согласно UDP-запросу. В качестве MAC адреса использовать 44:37:88:11:44:24, в качестве IP адреса – 192.168.1.27, UDP порт взять равным 1200, в качестве пароля взять kubsu. При команде «kubsu,switch_on_n», вводимой в поле передачи данных программы «UDP», микроконтроллер должен включить n-светодиод, подключенный к выводу PORTDn, при команде «kubsu,switch_off_n» – выключить его. При этом, после каждого включения/выключения светодиода микроконтроллер должен посылать команду подтверждения «Diodn_is_switch_on» и «Diodn_is_switch_off» соответственно.

Контрольные вопросы

1. Для чего предназначен протокол UDP?

2. Опишите структуру UDP-дейтаграммы и предназначение каждого поля.

3. Каким образом происходит идентификация процессов при использовании UDP протокола?

4. Для чего служит псевдозаголовок UDP? Опишите его структуру.

5. Какой формат имеют данные, полученные микроконтроллером ATmega168 от ethernet-чипа ENC28J60 при передаче данных по протоколу UDP?

Рекомендуемая литература

1 Гребнев В. В. Микроконтроллеры семейства AVR фирмы Atmel / / В. В. Гребнев. – М.: ИП РадиоСофт, 2002 – 176 с: ил. С. 43−47.

2 Евстифеев А. В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя / А. В. Евстифеев. – М: Издательский дом «Додека-XXI», 2007. − 432 с.: ил. С. 419−427.

3 Подключение микроконтроллера к локальной сети: работаем с ENC28J60. – 2011. – Сайт для разработчиков устройств. – URL: http://we.easyelectronics.ru /electro-and-pc/podklyuchenie-mikrokontrollera-k-lokalnoy-seti-rabotaem-s-enc28j60.html

4 ATMEGA168 – 8-bit Microcontroller with 8K Bytes In-System Programmable Flash – ATMEL Corporation. – Сайт документации на электронные компоненты. – (Англ.). – URL: http://www.alldatasheet.com/ datasheetpdf/pdf/83753/ATMEL/ATMEGA168.html

5 ENC28J60 – Stand-Alone Ethernet Controller with SPI Product Brief – Microchip Technology. – Сайт документации на электронные компоненты. – (Англ.). – URL: http://www.alldatasheet.com/datasheet-pdf/ pdf/102687/ MICROCHIP/ENC28J60.html

6 Паркер Т. TCP/IP. Для профессионалов. 3-е изд. / Т. Паркер, К. Сиян. – СПб.: Питер. – 2004. – 859 с: ил.

Лабораторная работа №10 «Реализация протоколов TCP, HTTP на микроконтроллере»

Цель: научиться программно, с использованием микроконтроллера, принимать и передавать данные посредством Ethernet-интерфейса по протоколам TCP и HTTP, организовать Web-сервер на основе микроконтроллера, отвечающий на запросы по протоколу HTTP.

Задачи: изучить TCP и HTTP протоколы, структуру TCP сообщения и HTTP запроса, разобрать программы, реализующие Web сервер на микроконтроллере.

Приборы и принадлежности: плата на основе микроконтроллеров ATmega168 и ENC28J60 , Datasheet к микроконтроллерам ATmega168 и ENC28J60, LPT- программатор, ПК с установленными PonyProg и пакетом программ WinAVR.

Краткие теоретические сведения

Протокол управления передачей TCP – один из основных транспортных протоколов, используемых в сети Интернет. В отличие от UDP, он гарантирует надежную доставку данных от отправителя к получателю и имеет ряд полезных функций: управление потоком передачи данных, управление приоритетностью передаваемых данных, мультиплексирование данных. На рисунке 30 представлена структура сегмента TCP.

Как и протокол UDP, TCP идентифицирует процессы по номерам портов. Номера портов, используемые для конкретных специфических целей, выделяет и регистрирует организация IANA . Все порты разделяются на 3 категории: общеизвестные, с номерами от 0 до 1023; зарегистрированные, с номерами от 1024 до 49151; динамически используемые порты для закрытых сетей, с номерами от 49152 до 65535. Например, протокол HTTP использует порт с номером 80. Сегмент TCP начинается с полей: «Порт источника», «Порт назначения», в которые помещаются номера соответствующих портов.

Бит	0 — 3	4 — 9	10 — 15	16 — 31
0	Порт источника			Порт назначения
32	Номер последовательности			
64	Номер подтверждения			
96	Смещение данных	Зарезервировано	Флаги	Размер Окна
128	Контрольная сумма			Указатель важности
160	Опции (необязательное, но используется практически всегда)			
160/192+	Данные			

Рисунок 30 – Структура сегмента TCP

Надежность передачи данных в TCP обеспечивается полями «Номер последовательности» и «Номер подтверждения». Каждому октету данных в сегментах TCP присваивается порядковый номер. В поле «Номер последовательности» записывается порядковый номер первого октета данных в сегменте. Поле «Номер подтверждения» содержит порядковый номер сле-

дующего ожидаемого октета данных и считывается получателем, если установлен флаг ACK.

Поле «Смещение данных» определяет размер заголовка TCP в 32-битных словах. Минимальный размер заголовка 5 слов, максимальный – 15.

Поле «Зарезервировано» предназначено для использования в будущем, а пока должно быть заполнено нулями.

Поле «Флаги» содержит 6 флагов соответственно: URG, ACK, PSH, RST, SYN, FIN. Если установлен флаг URG, то поле «Указатель важности» задействовано, если ACK – поле «Номер подтверждения». Установленный флаг PSH инструктирует получателя протолкнуть данные, накопившиеся в приемном буфере, в приложение пользователя. Флаг RST устанавливается для обрыва соединения и очистки буфера, флаг SYN – для процесса синхронизации номеров последовательности, FIN – для завершения соединения.

Поле «Размер окна» управляет потоком передачи данных и указывает размер данных в байтах, которые отправитель готов принять.

Поле «Контрольная сумма», как и в случае UDP, рассчитывается с учетом псевдозаголовка, который в явном виде не присутствует в потоке данных и служит только для проверки контрольной суммы, исключая ошибки маршрутизации (Рис. 31).

Биты	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0-31	IP-адрес отправителя (Source address)																															
32-63	IP-адрес получателя (Destination address)																															
64-95	0	0	0	0	0	0	0	0	Протокол (Protocol)								Длина TCP-сегмента (TCP length)															

Рисунок 31 – Псевдозаголовок TCP

В поле «Указатель важности» находится порядковый номер октета данных, на котором заканчиваются важные данные и принимается во внимание, только если установлен флаг URG.

Поле «Опции» используется для настройки дополнительных параметров и не обязательно. Длина поля должна быть кратна 8 битам для того, чтобы можно было определить его длину из поля «Смещение данных». Если кратности нет, то поле выравнивается нулями. Существует 2 формата поля «Опции»: тип параметра (1 октет); тип параметра (1 октет), длина параметра (1 октет), данные параметра (2 октета). Параметры типа 0 и 1 относятся к первому формату, содержат один октет и означают соответственно конец списка параметров и нет операции. Параметр с типом 2 относится ко второму формату, содержит 4 октета и оповещает о максимальной

ном размере сегмента TCP (MSS). Значение MSS согласовывается только в начале соединения при синхронизации порядковых номеров. Если значение MSS не используется, то разрешаются сегменты любого размера.

Прежде чем передавать данные по протоколу TCP, необходимо открыть соединение. Для этого используется процедура согласования, основанная на обмене тремя сегментами со следующими значениями флагов SYN и ACK:

1) SYN=1, ACK=0 – открытие соединения;
2) SYN=0, ACK=1 – подтверждение открытия;
3) SYN=0, ACK=1 – пакет данных или сегмент ACK.

Следует отметить, что каждый раз после отправки сегмента принимающей стороне, последняя должна отправить сегмент подтверждения принятых данных с установленным флагом ACK и номером следующего ожидаемого октета данных (Поле «Номер подтверждения»), что будет говорить о корректном приеме сегмента. Если подтверждение не пришло, то по тайм-ауту, сегмент будет отправлен заново. Принимающая сторона может инициировать повторную отправку сегмента, полученного, например, с ошибкой, указав номер подтверждения, равный номеру предыдущего верно полученного сегмента.

Если клиент запрашивает открытие соединения у сервера, отправив сегмент со следующими параметрами: SYN=1, ACK=0, SEQ (номер последовательности) = X, Z октетов данных. Тогда сервер подтвердит открытие соединения, ответив: SYN=0, ACK=1, SEQ=Y, NACK (номер подтверждения) =X+Z+1. После этого клиент может отправить запрос на получение интересующей его информации: SYN=0, ACK=1, SEQ=X+Z+1, NACK=Y+1, Q октетов данных запроса.

Протокол HTTP обеспечивает обмен данными между web-серверами и web-клиентами. Он строится на схеме «запрос/ответ». Сервер ожидает запросы и, при получении последних, отвечает на них. HTTP не поддерживает собственных соединений с клиентом и использует надежные соединения TCP, чаще всего на порту TCP с номером 80. Операция обмена данными между клиентом и сервером делится на четыре основных этапа: браузер подключается к серверу, браузер запрашивает документ с сервера, сервер передает браузеру запрошенные данные, связь разрывается. Протокол HTTP не хранит информацию о состоянии текущего подключения. В таблице 19 представлена структура запросов HTTP.

Таблица 19 – Структура пакета запросов http

Метод	Запрашиваемый URI	Версия HTTP

Поле «метод» содержит действие, которое нужно выполнить для ресурса: GET – запрашивает указанный документ; HEAD – запрашивает только заголовок документа; POST – сервер должен интерпретировать указанный документ как исполняемую программу и передать ему некоторые данные; PUT – заменяет содержимое указанного документа данными, полученными от клиента; DELETE – запрашивает удаление указанной страницы.

Поле «запрашиваемый URI» содержит универсальный идентификатор ресурса, к которому адресуется запрос, то есть сетевого ресурса.

Поле «версия HTTP» содержит информацию об используемой версии протокола HTTP.

Пакет откликов представлен в таблице 20.

Таблица 20 – Структура пакета откликов HTTP

Версия HTTP	Код состояния	Причина

Поле «код состояния» – содержит трехзначное целое число, указывающее код результата, например, успешный код – 200–206, ошибки клиента – 400–415, ошибки сервера – 500–505).

Поле «причина» содержит текстовое описание результата запроса.

Рассмотрим пример программы, реализующий Web-сервер на микроконтроллере с использованием TCP в качестве транспортного протокола. Структура TCP сегмента, полученного микроконтроллером ATmega168 от ENC28J60 представлена на рисунке 32.

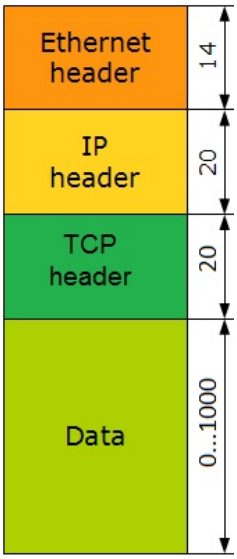


Рисунок 32 – Формат TCP-сегмента полученного микроконтроллером ATmega168

Как и в предыдущих работах, необходимо знать местоположение каждого поля заголовка TCP в буфере данных. Для этого в файл net.h добавим следующее:

#define TCP_SRC_PORT_H_P 0x22 //старший байт поля "порт источника" - 34 байт данных
#define TCP_SRC_PORT_L_P 0x23 //младший байт поля "порт источника" - 35 байт данных
#define TCP_DST_PORT_H_P 0x24 //старший байт поля "порт назначения"- 36 байт данных
#define TCP_DST_PORT_L_P 0x25 //младший байт поля "порт назначения"- 37 байт данных
#define TCP_SEQ_H_P 0x26 //поле "номер последовательности" начинается с 38 байта данных
#define TCP_SEQACK_H_P 0x2a //поле"номер подтверждения" начинается с 42 байта данных
#define TCP_FLAGS_P 0x2f //поле "флаги" содержится в 47 байте данных
#define TCP_FLAGS_SYN_V 2 //присвоение флагам значений, соответствующих номерам разрядов,которые они занимают в 47 байте данных
#define TCP_FLAGS_FIN_V 1
#define TCP_FLAGS_PUSH_V 8
#define TCP_FLAGS_SYNACK_V 0x12
#define TCP_FLAGS_ACK_V 0x10
#define TCP_FLAGS_PSHACK_V 0x18
#define TCP_HEADER_LEN_PLAIN 20 //длина TCP заголовка без опций
#define TCP_HEADER_LEN_P 0x2e // поле "смещение данных" начинается с 46 байта данных

#define TCP_CHECKSUM_H_P 0x32 //старший байт поля "контрольная сумма"- 50 байт данных
#define TCP_CHECKSUM_L_P 0x33 //младший байт поля "контрольная сумма"- 51 байт данных
#define TCP_OPTIONS_P 0x36 // поле "опции" начинается с 54 байта данных

Для того, чтобы принимать, обрабатывать и передавать данные по протоколу TCP, написаны следующие функции, которые необходимо поместить в файл ip_arp_icmp.c из предыдущей работы:

```
int16_t checksum(uint8_t *buf, uint16_t len, uint8_t type) // вычисляем контрольную сумму для ip/udp/tcp заголовков
{
 uint32_t sum = 0;
 if(type==1)
 {
  sum+=IP_PROTO_UDP_V;
  sum+=len-8;
 }
 if(type==2)
 {
  sum+=IP_PROTO_TCP_V;
  sum+=len-8;
 }
 while(len >1)
 {
  sum += 0xFFFF & (*buf<<8|*(buf+1));
  buf+=2;
  len-=2;
 }
 if (len)
 {
  sum += (0xFF & *buf)<<8;
 }
 while (sum>>16)
 {
  sum = (sum & 0xFFFF)+(sum >> 16);
 }
 return( (uint16_t) sum ^ 0xFFFF);
}

void make_tcphead(uint8_t *buf,uint16_t rel_ack_num,uint8_t mss,uint8_t cp_seq) //функция генерации заголовка TCP для последующей отправки
 {
  uint8_t i=0;
  uint8_t tseq;
  while(i<2)
```

```
 {
  buf[TCP_DST_PORT_H_P+i]=buf[TCP_SRC_PORT_H_P+i];
  buf[TCP_SRC_PORT_H_P+i]=0; // clear source port
  i++;
 }
buf[TCP_SRC_PORT_L_P]=80;
i=4;
while(i>0)
 {
  rel_ack_num=buf[TCP_SEQ_H_P+i-1]+rel_ack_num;
  tseq=buf[TCP_SEQACK_H_P+i-1];
  buf[TCP_SEQACK_H_P+i-1]=0xff&rel_ack_num;
  if (cp_seq)
   {
    buf[TCP_SEQ_H_P+i-1]=tseq;
   }
  else
   {
    buf[TCP_SEQ_H_P+i-1]= 0;
   }
  rel_ack_num=rel_ack_num>>8;
  i--;
 }
if (cp_seq==0)
 {
  buf[TCP_SEQ_H_P+0]= 0;
  buf[TCP_SEQ_H_P+1]= 0;
  buf[TCP_SEQ_H_P+2]= seqnum;
  buf[TCP_SEQ_H_P+3]= 0;
  seqnum+=2;
 }
buf[TCP_CHECKSUM_H_P]=0;
buf[TCP_CHECKSUM_L_P]=0;
if (mss)
 {
  buf[TCP_OPTIONS_P]=2;
  buf[TCP_OPTIONS_P+1]=4;
  buf[TCP_OPTIONS_P+2]=0x05;
  buf[TCP_OPTIONS_P+3]=0x80;
  buf[TCP_HEADER_LEN_P]=0x60;
 }
else
 {
  buf[TCP_HEADER_LEN_P]=0x50;
 }
}
```

```
void make_tcp_synack_from_syn(uint8_t *buf) //функция отправки подтверждения открытия соединения
 {
  uint16_t ck;
  make_eth(buf);
  buf[IP_TOTLEN_H_P]=0;
  buf[IP_TOTLEN_L_P]=IP_HEADER_LEN+TCP_HEADER_LEN_PLAIN+4;
  make_ip(buf);
  buf[TCP_FLAGS_P]=TCP_FLAGS_SYNACK_V;
  make_tcphead(buf,1,1,0);
  ck=checksum(&buf[IP_SRC_P], 8+TCP_HEADER_LEN_PLAIN+4,2);
  buf[TCP_CHECKSUM_H_P]=ck>>8;
  buf[TCP_CHECKSUM_L_P]=ck& 0xff;

enc28j60PacketSend(IP_HEADER_LEN+TCP_HEADER_LEN_PLAIN+4+ETH_HEADER_LEN,buf);
 }
uint16_t get_tcp_data_pointer(void) //функция расчета  положения первого октета данных в буфере
 {
  if (info_data_len)
  {
   return((uint16_t)TCP_SRC_PORT_H_P+info_hdr_len);
  }
  else
  {
   return(0);
  }
 }
void init_len_info(uint8_t *buf) //функция определения размера TCP заголовка и принятых данных
 {
  info_data_len=(buf[IP_TOTLEN_H_P]<<8)|(buf[IP_TOTLEN_L_P]&0xff);
  info_data_len-=IP_HEADER_LEN;
  info_hdr_len=(buf[TCP_HEADER_LEN_P]>>4)*4;
  info_data_len-=info_hdr_len;
  if (info_data_len<=0)
   {
    info_data_len=0;
   }
 }
uint16_t fill_tcp_data_p(uint8_t *buf,uint16_t pos, const prog_char *progmem_s) //функция записи строковых данных в буфер
 {
  char c;
  while ((c = pgm_read_byte(progmem_s++)))
   {
    buf[TCP_CHECKSUM_L_P+3+pos]=c;
```

```
      pos++;
     }
    return(pos);
  }
uint16_t fill_tcp_data(uint8_t *buf,uint16_t pos, const char *s) //функция записи строковых данных в буфер
 {
   while (*s)
    {
     buf[TCP_CHECKSUM_L_P+3+pos]=*s;
     pos++;
     s++;
    }
   return(pos);
  }
void make_tcp_ack_from_any(uint8_t *buf) // функция генерации подтверждения любого запроса
 {
   uint16_t j;
   make_eth(buf);
   buf[TCP_FLAGS_P]=TCP_FLAGS_ACK_V;
   if (info_data_len==0)
    {
     make_tcphead(buf,1,0,1);
    }
   else
    {
     make_tcphead(buf,info_data_len,0,1);
    }
    j=IP_HEADER_LEN+TCP_HEADER_LEN_PLAIN;
    buf[IP_TOTLEN_H_P]=j>>8;
    buf[IP_TOTLEN_L_P]=j& 0xff;
    make_ip(buf);
     j=checksum(&buf[IP_SRC_P], 8+TCP_HEADER_LEN_PLAIN,2);
     buf[TCP_CHECKSUM_H_P]=j>>8;
     buf[TCP_CHECKSUM_L_P]=j& 0xff;

enc28j60PacketSend(IP_HEADER_LEN+TCP_HEADER_LEN_PLAIN+ETH_HEADER_LEN,buf);
  }
void make_tcp_ack_with_data(uint8_t *buf,uint16_t dlen)
 {
   uint16_t j;

buf[TCP_FLAGS_P]=TCP_FLAGS_ACK_V|TCP_FLAGS_PUSH_V|TCP_FLAGS_FIN_V;
   j=IP_HEADER_LEN+TCP_HEADER_LEN_PLAIN+dlen;
   buf[IP_TOTLEN_H_P]=j>>8;
```

```
   buf[IP_TOTLEN_L_P]=j& 0xff;
   fill_ip_hdr_checksum(buf);
   buf[TCP_CHECKSUM_H_P]=0;
   buf[TCP_CHECKSUM_L_P]=0;
   j=checksum(&buf[IP_SRC_P], 8+TCP_HEADER_LEN_PLAIN+dlen,2);
   buf[TCP_CHECKSUM_H_P]=j>>8;
   buf[TCP_CHECKSUM_L_P]=j& 0xff;

enc28j60PacketSend(IP_HEADER_LEN+TCP_HEADER_LEN_PLAIN+dlen+ETH_H
EADER_LEN,buf);
  }
```

Теперь, используя все вышеприведенные функции и функции, изложенные в лабораторных работах 8 и 9, можно составить главную программу, реализующую Web-сервер на микроконтроллере с использованием протокола TCP. При этом web-страница будет интерактивной и позволит не только управлять светодиодом, но и производить контроль его состояния в режиме реального времени:

```
#include <avr/io.h>
#include <stdlib.h>
#include <string.h>
#include <avr/pgmspace.h>
#include "ip_arp_udp_tcp.h"
#include "enc28j60.h"
#include "timeout.h"
#include "avr_compat.h"
#include "net.h"
static uint8_t mymac[6] = {0x66,0x32,0x54,0x10,0x45,0x31}; //MAC-адрес
static uint8_t myip[4] = {192,168,1,25}; //IP-адрес
static char baseurl[]="http://192.168.1.25/"; //значение URL
#define BUFFER_SIZE 700 // размер буфера для приема данных
static uint8_t buf[BUFFER_SIZE+1];
static char password[]="secret"; //пароль(не длиннее 9 символов)
uint8_t verify_password(char *str) //функция проверки пароля (первых 5 символов)
 {
  if (strncmp(password,str,5)==0)
  {
   return(1); //при совпадении возвращает 1
  }
  return(0); //при не совпадении возвращает 0
 }
int8_t analyse_get_url(char *str) // функция анализа получаемого URL
 {
  uint8_t i=0;
  if (verify_password(str)==0)
```

```
   {
    return(-1); //возвращает -1, если пароль не правильный
   }
   while(*str && i<10 && *str >',' && *str<'{') //если пароль правильный, найдем положение символа '/'после пароля
   {
    if (*str=='/')
    {
     str++;
     break;
    }
    i++;
    str++;
   }
   if (*str < 0x3a && *str > 0x2f) // если после '/' находится число от 0 до 9 (код ASCII)
   {
    return(*str-0x30); //возвратим значение этого числа числа в прямом виде
   }
   return(-2); // если пароль верный, но нет команды после него, то возвращает -2
  }
uint16_t print_webpage(uint8_t *buf,uint8_t on_off) // записывает в буфер страницу для отправки
  {
   uint16_t plen;
   plen=fill_tcp_data_p(buf,0,PSTR("HTTP/1.0 200 OK\r\nContent-Type: text/html\r\n\r\n"));
   plen=fill_tcp_data_p(buf,plen,PSTR("<center><font size=25>УПРАВЛЕНИЕ И МОНИТОРИНГ СОСТОЯНИЯ СВЕТОДИОДА"));
   plen=fill_tcp_data_p(buf,plen,PSTR("<center><p><font color=\"#ff0000\">Диод: </font> "));
   if (on_off)
   {
    plen=fill_tcp_data_p(buf,plen,PSTR("<font color=\"#00FF00\"> Включен </font>"));
   }
   else
   {
    plen=fill_tcp_data_p(buf,plen,PSTR("Выключен"));
   }
   plen=fill_tcp_data_p(buf,plen,PSTR(" <small><a href=\""));
   plen=fill_tcp_data(buf,plen,baseurl);
   plen=fill_tcp_data(buf,plen,password);
   plen=fill_tcp_data_p(buf,plen,PSTR("\">[Проверить состояние]</a></small></p>\n<p><a href=\""));
   plen=fill_tcp_data(buf,plen,baseurl);
   plen=fill_tcp_data(buf,plen,password);
   if (on_off)
   {
```

```
     plen=fill_tcp_data_p(buf,plen,PSTR("/0\">Выключить</a><p>"));
    }
   else
   {
    plen=fill_tcp_data_p(buf,plen,PSTR("/1\">Включить</a><p>"));
    }
   plen=fill_tcp_data_p(buf,plen,PSTR("</center><br><hr>Демонстрация работы
платы "));
   plen=fill_tcp_data_p(buf,plen,PSTR(" как Web-сервера. "));
   plen=fill_tcp_data_p(buf,plen,PSTR(" КубГУ.ФТФ. 2012 "));
   return(plen);
  }
int main(void)
 {
   uint16_t plen;
   uint16_t dat_p;
   int8_t cmd;
   PORTD=0x00;
   DDRD=0xff;
   CLKPR=(1<<CLKPCE); //разрешаем установку частоты тактирования
   CLKPR=0; // устанавливаем частоту тактирования 8 Мгц
   delay_ms(10);
   enc28j60Init(mymac); //инициализация ENC28J60
   delay_ms(20);
   enc28j60PhyWrite(PHLCON,0x476); //устанавливаем один диод для статуса соеди-
нения,другой для статуса приема/передачи
   delay_ms(20);
   init_ip_arp_udp(mymac,myip); //инициализация ip- и MAC-адресов
   while(1)
    {
     plen = enc28j60PacketReceive(BUFFER_SIZE, buf); //запускаем функцию приема
пакетов от ENC28J60
     if(plen==0) // если принятых данных нет
     {
      continue; //то перейти на начало цикла while
      }
     if(eth_type_is_arp_and_my_ip(buf,plen)) // если полученные данные являются
ARP-запросом, адресованным нам
     {
       make_arp_answer_from_request(buf); //то сгенерировать и отправить ARP-ответ
       continue; // перейти на начало цикла while
      }
    if(eth_type_is_ip_and_my_ip(buf,plen)==0) // если полученные данные являются ip-
пакетом и адресованы не нам
    {
     continue; // перейти на начало цикла while
     }
```

```
if(buf[IP_PROTO_P]==IP_PROTO_ICMP_V&&buf[ICMP_TYPE_P]==ICMP_TYPE
_ECHOREQUEST_V) //если в полученном ip-пакете содержится запрос "пинг"
    {
     make_echo_reply_from_request(buf,plen); // отправим ответ "понг"
     continue; //перейти на начало цикла while
    }
  if (buf[IP_PROTO_P]==IP_PROTO_TCP_V&&buf[TCP_DST_PORT_H_P]==0
&&buf[TCP_DST_PORT_L_P]==80) // если полученные данные-TCP сегменты и ад-
ресованы нам (на порт 80)
    {
     if (buf[TCP_FLAGS_P] & TCP_FLAGS_SYN_V) //если идет запрос на синхрони-
зацию
     {
      make_tcp_synack_from_syn(buf); //отправим подтверждение синхронизации
      continue; //выйдем из цикла и начнем принимать данные заново
     }
     if (buf[TCP_FLAGS_P] & TCP_FLAGS_ACK_V) //если установлен флаг под-
тверждения
     {
      init_len_info(buf); //определяем длину данных в TCP сегменте
      dat_p=get_tcp_data_pointer(); //устанавливаем указатель на начало данных в TCP
сегменте
      if (dat_p==0) //прислан TCP сегмент без указания размера заголовка
      {
       if (buf[TCP_FLAGS_P] & TCP_FLAGS_FIN_V) //если флаг завершения соеди-
нения FIN установлен
       {
        make_tcp_ack_from_any(buf); //подтвердить завершение
       }
       continue; //и выйти из цикла
      }
      if (strncmp("GET ",(char *)&(buf[dat_p]),4)!=0) // если получен запрос "Get"
      {
       plen=fill_tcp_data_p(buf,0,PSTR("HTTP/1.0 200 OK\r\nContent-Type:
text/html\r\n\r\n<h1>200 OK</h1>")); //отправить команду 200 OK
       goto SENDTCP; //перейти к отправке
      }
      if (strncmp("/ ",(char *)&(buf[dat_p+4]),2)==0) // если получен '/ '
      {
       plen=fill_tcp_data_p(buf,0,PSTR("HTTP/1.0 200 OK\r\nContent-Type:
text/html\r\n\r\n")); //отправить команду 200 OK
       plen=fill_tcp_data_p(buf,plen,PSTR("<p>Usage: ")); // отправить подсказку ввес-
ти пароль: "Usage:http://192.168.1.25/password
       plen=fill_tcp_data(buf,plen,baseurl);
       plen=fill_tcp_data_p(buf,plen,PSTR("password</p>"));
       goto SENDTCP; // перейти к отправке
      }
```

```
      cmd=analyse_get_url((char  *)&(buf[dat_p+5]));  //запускаем функцию анализа
URL и записываем возвращаемое ею значение в cmd
      if (cmd==-1) // если пароль неверный
      {
       plen=fill_tcp_data_p(buf,0,PSTR("HTTP/1.0 401 Unauthorized\r\nContent-Type:
text/html\r\n\r\n<h1>401 Unauthorized</h1>")); //оправить команду отказано в автори-
зации
       goto SENDTCP; //перейти к отправке
       }
      if (cmd==1) //если команда равна 1
      {
        PORTD|= (1<<PD7); //включить светодиод PD7
       }
      if (cmd==0) //если команда равна 0
      {
        PORTD &= ~(1<<PD7); //выключить светодиод
       }
      plen=print_webpage(buf,(PORTD & (1<<PD7))); //создать страницу
     //SENDTCP:
      make_tcp_ack_from_any(buf); // подтвердить запрос get
      make_tcp_ack_with_data(buf,plen); // отправить данные
      continue; //выйти из цикла и перейти к началу приема данных
     }
    }
  }
 return (0);
}
```

Работает программа следующим образом: при вводе в адресной строке браузера «http://192.168.1.25/secret», микроконтроллер высылает интерактивную страничку (Рис. 33), с помощью которой можно включать, выключать светодиод D7, а также проводить мониторинг его состояния в режиме реального времени. «Secret» – это пароль, который прописан в главном файле программы.

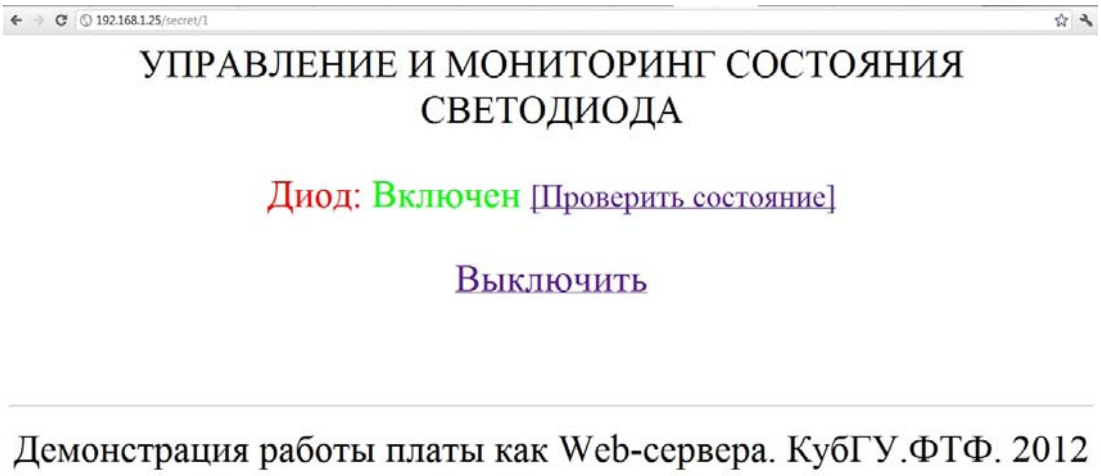


Рисунок 33 – Интерактивная Web-страничка управления светодиодом

Задание: на основе теоретических знаний и примеров программ, изложенных в данной лабораторной работе и лабораторных работах 8 и 9, написать программу, реализующую Web сервер на микроконтроллере. В качестве MAC адреса использовать 44:37:83:10:0:24, в качестве IP адреса – 10.24.0.112, TCP порт взять равным 80, в качестве пароля взять kubsu. На команду http://10.24.0.112/kubsu (HTTP-запрос), вводимую в адресной строке браузера, микроконтроллер (Web-сервер) должен ответить html страницей, содержание которой «Я люблю физико-технический факультет».

Контрольные вопросы

1. Для чего предназначен протокол TCP?
2. Опишите структуру сегмента TCP и предназначение каждого поля.
3. Каким образом происходит идентификация процессов при использовании TCP протокола?
4. Для чего служит псевдозаголовок TCP? Опишите его структуру.
5. Чем отличается TCP от UDP протокола? Какие преимущества и недостатки они имеют?
6. Для чего предназначен протокол HTTP?
7. Опишите структуру пакета запросов и откликов протокола HTTP и предназначение каждого поля.
8. Какой формат имеют данные, полученные микроконтроллером ATmega168 от ethernet-чипа ENC28J60 при передаче данных по протоколу TCP?

Рекомендуемая литература

1 Гребнев В. В. Микроконтроллеры семейства AVR фирмы Atmel / / В. В. Гребнев. – М.: ИП РадиоСофт, 2002 – 176 с: ил. С. 43−47.

2 Евстифеев А. В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя / А. В. Евстифеев. – М: Издательский дом «Додека-XXI», 2007. − 432 с.: ил. С. 419−427.

3 Подключение микроконтроллера к локальной сети: работаем с ENC28J60. – 2011. – Сайт для разработчиков устройств. – URL: http://we.easyelectronics.ru /electro-and-pc/podklyuchenie-mikrokontrollera-k-lokalnoy-seti-rabotaem-s-enc28j60.html

4 ATMEGA168 – 8-bit Microcontroller with 8K Bytes In-System Programmable Flash – ATMEL Corporation. – Сайт документации на электронные компоненты. – (Англ.). – URL: http://www.alldatasheet.com/ datasheetpdf/pdf/83753/ATMEL/ATMEGA168.html

5 ENC28J60 – Stand-Alone Ethernet Controller with SPI Product Brief – Microchip Technology. – Сайт документации на электронные компоненты. – (Англ.). – URL: http://www.alldatasheet.com/datasheet-pdf/ pdf/102687/ MICROCHIP/ENC28J60.html

6 Паркер Т. TCP/IP. Для профессионалов. 3-е изд. / Т. Паркер, К. Сиян. – СПб.: Питер. – 2004. – 859 с: ил.