

## СОЗДАНИЕ НАДЕЖНОГО HTTP-СЕРВЕРА ДЛЯ УДАЛЕННОГО УПРАВЛЕНИЯ ПО TCP/IP СЕТИ НА ОСНОВЕ ATMEGA1280 И WIZNET W5100

*В работе рассматривается создание http-сервера на микроконтроллере ATmega1280 и контроллере сети Wiznet W5100. Сервер пересылает браузеру клиента HTML текст, изображения, показания температурного датчика DS18B20. Управляет мобильным телефоном. Возможно простое расширение его функций. Программа для сервера написана на Си (WinAVR). Особенностью сервера является его устойчивая работа через сеть Интернет, возможность использования всей памяти программ микроконтроллера (>64Кбайт) для размещения HTML данных. Программная среда Arduino на это не способна.*

*Ключевые слова: микроконтроллер, ATmega1280, Wiznet W5100, Arduino, TCP/IP, ethernet, SPI, регистры, протокол.*

**Введение.** Создание систем управления по TCP/IP сети требует построения на микроконтроллерах устойчиво работающего web-сервера[1]. Возможно построение сервера на базе контроллера Wiznet W5100 с встроенными в него Ethernet и TCP/IP протоколами на аппаратном уровне. Это значительно упрощает программу сервера и уменьшает требуемый размер оперативной памяти микроконтроллера, так как нет необходимости писать программу для реализации протокола TCP/IP. Необходимо лишь реализовать на микроконтроллере протокол HTTP.

**Постановка задачи.** Предлагаемый здесь web-сервер должен обладать следующими особенностями по сравнению с представленным HTTP-сервером в статье [1]:

1. Аппаратная часть реализована на AVR микроконтроллере ATmega1280 и микросхеме Wiznet W5100. Указанные схемы установлены в модулях Arduino Mega и Ethernet Shield W5100[2].

2. К серверу подключен температурный датчик DS18B20, показания которого должны отображаться браузером клиента.

3. Сервер может пересылать клиенту кроме текста также и изображения. Эти данные должны размещаться в flash памяти микроконтроллера в виде массивов байт.

4. Размер данных, размещенных в flash памяти, может превышать 64Кбайт. Поэтому для размещения данных и программы можно использовать всю flash память ATmega1280 и Atmega2560. Программная среда Ардуино этого не делает для микроконтроллеров AVR[1].

5. Сервер должен работать устойчиво, не «зависать» при подключении к нему с сети Интернет, что характерно для программной среды Ардуино и программы в статье [1]. Это очень важно при создании надежных систем удаленного управления.

6. К серверу должен подключаться мобильный телефон, который выполняет дозвон при срабатывании концевого выключателя (например, при открытии входной двери). Дозвон также может быть инициирован удаленно от сервера, если в браузере набрать заданный адрес. Если абонент примет вызов, то он сможет прослушивать удаленно район расположения сервера.

**Полученные результаты.** *Схема подключения сервера к устройствам.*

На рис. 1 показана схема подключения Arduino (ATmega1280) к температурному датчику через 6-й выход (PORTH3), к реле дозвона через базу транзистора - 5-й выход (PORTE3) и к датчику двери через 2-й выход (PORTE4).

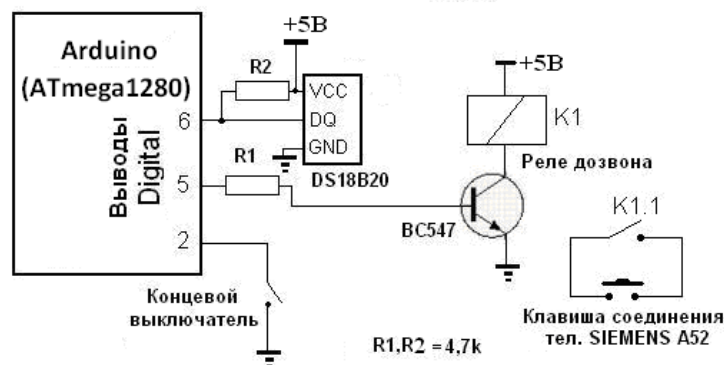


Рис. 1. Схема подключения Arduino к исполнительным устройствам

Согласно документации на мобильный телефон SIEMENS A52 при двукратном нажатии на клавишу соединения выполняется вызов последнего набранного номера.

*Основные сведения о TCP/IP контроллере Wiznet W5100.*

В Wiznet W5100 реализован полнофункциональный стандарт IEEE 802.3 (физический и канальный уровень протокола Ethernet), стек TCP/IP протоколов. Поэтому программирование Wiznet W5100 состоит в чтении и записи данных, которые находятся во внутренних регистрах, значения которых изменяются при работе стека TCP/IP. Работа с регистрами выполняется или параллельно с использованием шин адреса и данных или последовательно с помощью шины SPI. В работе управление чипом W5100 выполняется через шину SPI. На рисунке 2 показана связь между ATmega1280 и Ethernet Shield W5100 по шине SPI.

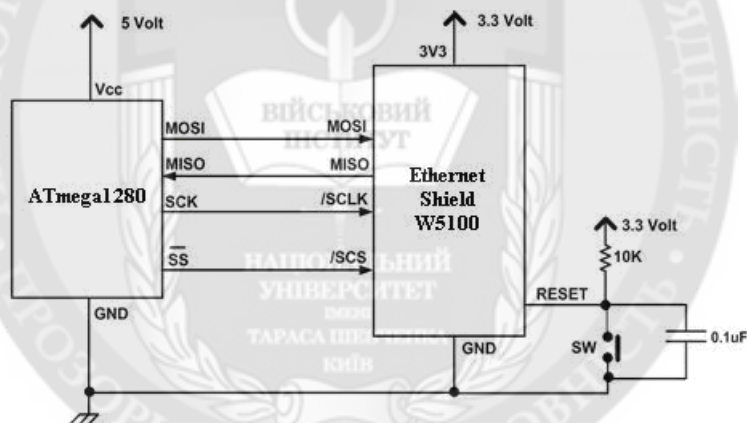


Рис. 2. Связь ATmega1280 и W5100 по SPI

Здесь Wiznet W5100 выступает в качестве ведомого устройства SPI, а микроконтроллер ATmega1280 в качестве ведущего. Для реализации протокола SPI нужно как минимум четыре сигнала, т.е. MOSI, MISO, SCK и CS. Микроконтроллер поддерживает все режимы SPI (т.е. 0,1,2 и 3), Wiznet W5100 чип – только режим 0 и режим 3. В работе используется режим 0 шины SPI. Не используется режим прерывания для W5100, поэтому вывод прерывания не задействован.

Построения сервера на Wiznet W5100 выполняется по этапам.

#### 1. Инициализация Wiznet W5100 по шине SPI.

Для инициализации необходимо выполнить соответствующие записи в регистры режима работы MR, маски сети SUBR, MAC – адреса SAR, IP – адреса SIPR, регистра памяти приема – RMSR и регистра памяти передачи данных - TMSR. На рисунке 3 показана карта памяти Wiznet W5100.

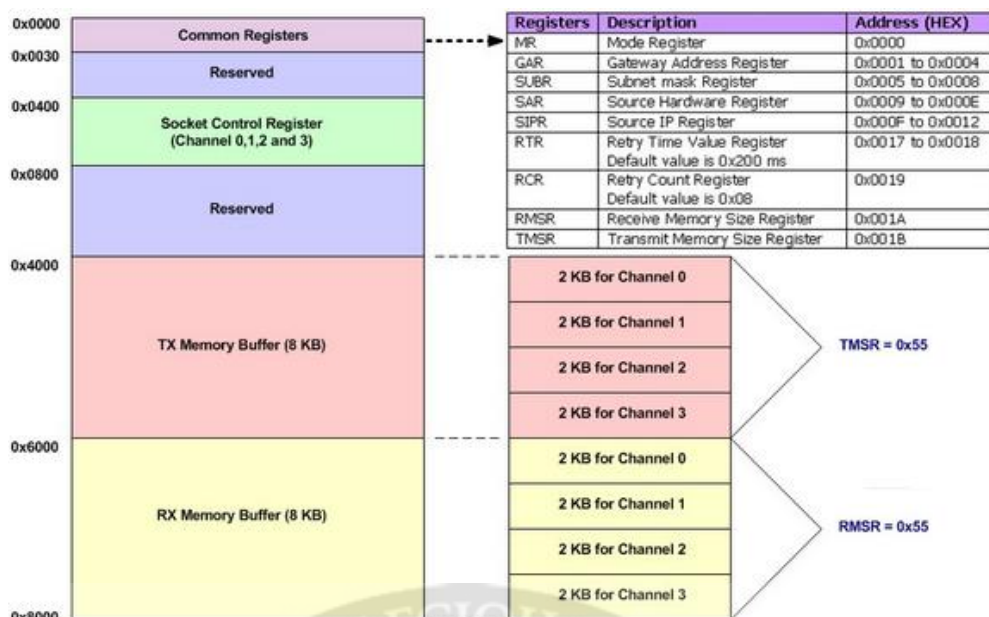


Рис. 3. Карта памяти Wiznet W5100

Все регистры адреса контроллера W5100 являются 16-разрядными, а его собственные регистры – 8 разрядными. Поэтому, используя 8 – разрядный микроконтроллер ATmega1280, по шине SPI записываем или читаем сначала первый 8-и битный старший байт, а потом следующий 8-и битный младший байт регистра адреса W5100. Для записи данных в Wiznet W5100 необходимо по SPI вначале передать команду 0xF0, а при чтении данных - команду 0x0F. Для записи и чтения по шине SPI в программе используются функции SPI\_Write() и SPI\_Read().

Wiznet W5100 поддерживает до 4-х одновременных каналов или sockets, причем каждый из каналов имеет свой собственный адресный регистр, контролирующей операции. Все эти каналы поддерживают 8КБайт буфер для передачи и 8КБайт буфер для приема. При инициализации Wiznet W5100 необходимо на каждый канал распределить конкретную величину памяти путем записи необходимых значений в регистры RMSR и TMSR. Например, для того, чтобы распределить на каждый канал по 2 Кбайт памяти, выполняется код:

```
SPI_Write(RMSR,0x55);
SPI_Write(TMSR,0x55);
```

За инициализацию W5100 отвечает функция W5100\_Init() программы на Си. В этой функции:

- Записывается 0x80 в W5100 в регистр режима MR по адресу 0x0000, что означает программный сброс W5100;
- Записываются по адресам с 0x0001 по 0x0004 в регистр GAR (регистр шлюза) четыре байта, представляющие собой адрес основного шлюза;
- Записываются по адресам с 0x0005 по 0x0008 четыре байта в регистр SUBR (регистр маски), представляющие собой маску сети;
- Записываются шесть байт MAC адреса сервера в SUBR по адресам с 0x0009 по 0x000E;
- По адресам с 0x00F по 0x0012 записывается 4-е байта IP – адреса сервера.

## 2. Программная реализация web – сервера

Web - сервер использует простой текст под названием язык гипертекстовой разметки (HTML) для взаимодействия с браузером через протокол TCP/IP. Поскольку протокол TCP/IP реализован в W5100, необходимо написать программу для микроконтроллера для реализации протокола HTTP. Сервер прослушивает запрос от браузера клиента через

стандартный TCP/IP порт номер 80 (в рассматриваемой работе порт 8080). После отправки запроса от клиента, согласно стандарту сервер ответит ему HTTP заголовком

HTTP /1.0 200 OK

Content-Type: text/html

(пустая строка)

После этого сервер перешлет весь HTML текст клиенту, а затем автоматически прервет соединение (см. рис. 4).

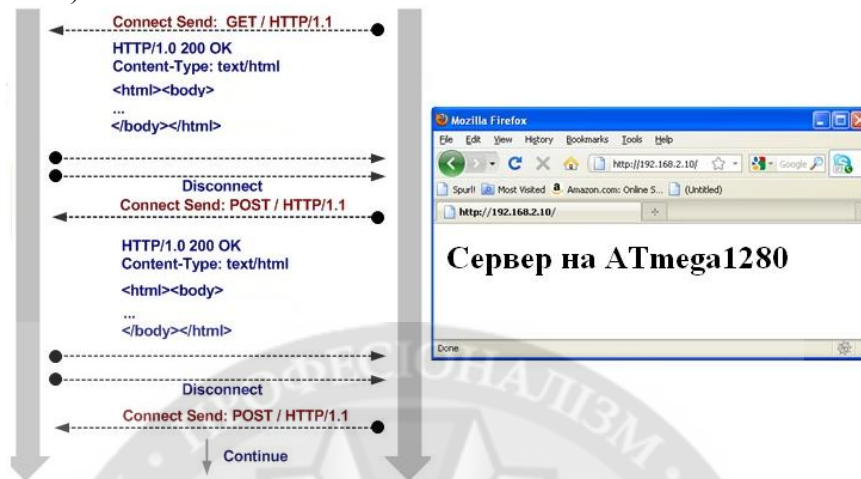


Рис. 4. Схема взаимодействия клиента и сервера

Ниже в качестве примера представлены запрос клиента и ответ сервера на него:

Client Request:

GET / HTTP/1.1

Host: 192.168.2.101

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Keep-Alive: 115

Connection: keep-alive

HTTP Server Response:

HTTP/1.0 200 OK

Content-Type: text/html

(Здесь пустая строка!)

<html> <body>

<h1>Сервер на ATmega1280</h1>

</body> </html>

Получив ответ от сервера, клиент представит HTML текст и изображения на экране браузера. Функции микроконтроллера состоят в том, чтобы правильно обрабатывать полученный запрос от клиента и отправлять необходимую в соответствии с запросом информацию клиенту. В связи с этим рассмотрим последовательность работы микроконтроллера ATmega1280 и Wiznet W5100.

Для настройки и управления Wiznet W5100 в режиме web – сервера необходимо работать с его регистрами управления и буферной памятью, которая получает и передает данные в сеть. Для простоты используется только один канал (сокет 0) из четырех, поддерживаемых контроллером W5100. Регистры управления для сокета 0 Wiznet W5100 начинаются с адреса 0x400 и заканчиваются адресом 0x4FF. В таблице 1 представлен список управляющих регистров для сокета 0 при условии использования буферной памяти приема RX и передачи TX данных размером по 2Кбайт для этого сокета.



## Регистры управления

Registers	Description	Address	Usage
S0_MR	Socket 0 Mode Register	0x0400	Select the protocol to be used i.e. TCP, UDP, PPP
S0_CR	Socket 0 Command Register	0x0401	Used for controlling the socket operation i.e. OPEN, LISTEN, CONNECT, DISCONNECT, CLOSE, SEND, RECEIVE
S0_SR	Socket 0 Status Register	0x0403	Contain the socket operation status value i.e. CLOSE,INITIAL, LISTEN, ESTABLISHED, CLOSED
S0_PORT	Socket 0 Source Port	0x0404 0x0405	Hold the TCP/IP Port value e.g. 80 for the standard HTTP server
S0_TX_FSR	Socket 0 TX Free Size Register	0x0420 0x0421	Maximum TX Free Memory buffer available for sending data
S0_TX_RD	Socket 0 TX Read Pointer	0x0422 0x0423	The start pointer at the available TX memory buffer for sending data
S0_TX_WR	Socket 0 TX Write Pointer	0x0424 0x0425	The end pointer mark at the TX memory buffer, this value will always the same as the S0_TX_RD after the "SEND" command
S0_RX_RSR	Socket 0 RX Received Size	0x0426 0x0427	Contain the receiving data size at the RX memory buffer
S0_RX_RD	Socket 0 RX Read Pointer	0x0428 0x0429	The start pointer of the received data at the RX memory buffer for reading the data
2 KB Socket 0 TX Memory Buffer 0x4000 to 0x47FF			
2 KB Socket 0 RX Memory Buffer 0x6000 to 0x67FF			

Последовательность работы web – сервера по программе на Си, можно посмотреть в Интернет - источнике[3].

Основой для работы сервера является использование командного регистра S0\_CR, а также выполнение чтения и записи данных в буферы W5100. Диаграмма на рисунке 5 показывает, в какой последовательности происходит работа с регистрами и буферами памяти TX, RX.

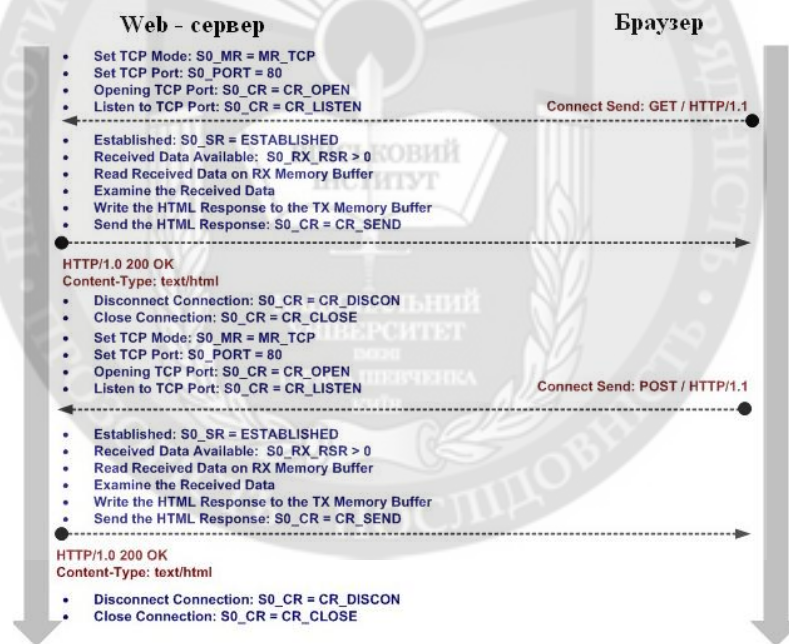


Рис. 5. Диаграмма работы с регистрами Wiznet W5100

Из диаграммы видно, что после инициализации W5100 открывается TCP/IP порт 80 и выполняется прослушивание этого порта для любых клиентских запросов. Эти действия выполняются функцией socket():

```

...
// Выбор протокола
SPI_Write(S0_MR,eth_protocol);
// Запись номера порта. Для этого сначала выделяем старший байт и
// смещаем его на 8 разрядов, записывая по адресу S0_PORT(0x0404)
SPI_Write(S0_PORT,((tcp_port & 0xFF00) >> 8));
// Далее выделяем младший байт и записываем его по адресу 0x0405

```

```

SPI_Write(S0_PORT + 1,(tcp_port & 0x00FF));
// Выполняем открытие сокета
SPI_Write(S0_CR,CR_OPEN);
// Ожидаем открытие сокета
while(SPI_Read(S0_CR));
ша(SPI_Read(S0_SR) == SOCK_INIT)

```

...

После записи команды CR\_OPEN (0x01) в регистр S0\_CR сокета 0, W5100 автоматически очистит этот регистр. Поэтому с помощью цикла while(SPI\_Read(S0\_CR)) необходимо подождать его очистки. Регистр статуса S0\_SR должен получить значение SOCK\_INIT(0x13).

После открытия сокета необходимо выполнить прослушивание 80 порта. Для этого в регистр команд S0\_CR необходимо отправить команду CR\_LISTEN (0x02). Как и в предыдущем случае с помощью цикла while(SPI\_Read(S0\_CR)) необходимо подождать очистки регистра S0\_CR, а регистр статуса S0\_SR должен получить значение SOCK\_LISTEN (0x14):

...

```

SPI_Write(S0_CR,CR_LISTEN);
while(SPI_Read(S0_CR));
if(SPI_Read(S0_SR) == SOCK_LISTEN)

```

...

Эти команды выполняются в функции listen(). После этих действий Wiznet W5100 готов принимать запросы от браузера клиента.

Согласно программе микроконтроллера, организован бесконечный цикл, в котором постоянно проверяются запросы со стороны клиента. Проверяется состояние регистра S0\_SR. Если его значение соответствует SOCK\_ESTABLISHED (0x17), то связь с клиентом установлена. Следующим действием является проверка размера считанных данных RX. Для этого выполняется чтение регистра S0\_RX\_RSR в функции recv\_size():

```

uint16_t recv_size(void)
{
// Читается старший и младший байт, которые представляются
// 16-и битным числом
return ((SPI_Read(S0_RX_RSR) & 0x00FF) << 8) + SPI_Read(S0_RX_RSR + 1);
}

```

Если полученные данные в буферной памяти RX существуют (recv\_size()>0) то выполняется чтение содержимого буфера RX памяти, которая реализуется в функции recv(). Однако замечено, что при установлении соединения данные в буферной памяти могут отсутствовать и это приводит к «зависанию» сервера. В связи с этим в программе вводится цикл ожидания, и если после его выполнения значение recv\_size() продолжает оставаться равным нулю, после небольшой задержки (0.5 – 1.0 секунды) инициируется принудительный разрыв соединения.

Перед чтением полученных буфером данных вначале необходимо рассчитать адрес памяти буфера, с которого необходимо выполнить чтение данных. Далее, полученные от клиента данные читаются с этого места. Предполагается, что буфер чтения для сокета 0 имеет размер 2048Байт (0x800). На рис. 6 поясняется, как рассчитывается физический адрес, с которого необходимо выполнить чтение полученных данных.

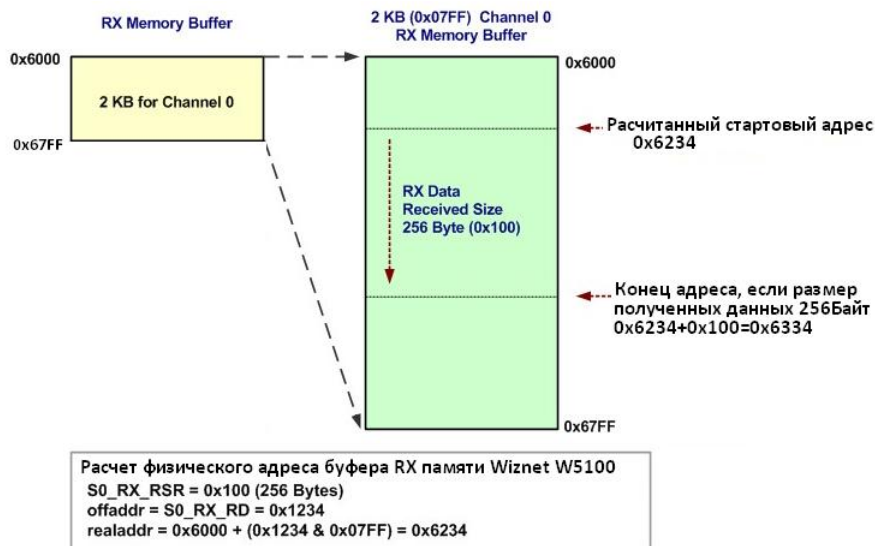


Рис. 6. Расчет физического адреса буфера RX памяти

Для определения физического адреса в буфере приема RX необходимо считать значение регистра `S0_RX_RD` и с помощью битовой операции `&` наложить на него значение `0x7FF` (размер буфера приема – 2048Байт нулевого сокета). К полученному значению добавить `0x6000` – начальный адрес буфера приема. В функции `recv()` это делается следующим образом:

```

...
// Чтение данных с регистра S0_RX_RD старшего и младшего байтов
ptr = SPI_Read(S0_RX_RD);
offaddr = (((ptr & 0x00FF) << 8) + SPI_Read(S0_RX_RD + 1));
// Чтение в цикле полученных данных из буфера RX с размещением их в массиве *buf
while(buflen) { buflen--;
// Расчет физического адреса
realaddr=RXBUFADDR + (offaddr & RX_BUF_MASK);
*buf = SPI_Read(realaddr); offaddr++; buflen++; }
*buf='\0'; // Формирование конца строки

```

Окончательно в регистр команд `S0_CR` пересылается команда `CR_RECV` (`0x40`) для того, чтобы Wiznet W5100 завершил процесс приема:

```

...
// Пересылка команды CR_RECV
SPI_Write(S0_CR,CR_RECV);
_delay_us(5); // Ожидание конца приема

```

После изучения запроса клиента, HTTP – сервер пошлет ответ. Ответ рассматриваемого тут сервера состоит из HTML – текстовых страниц, изображений и показаний температурного датчика DS18B20. HTML - текст и изображения представлены в виде массивов, которые размещены в flash памяти микроконтроллера. Часть текстовых данных размещена в SRAM – памяти микроконтроллера и копируется в текстовый буфер, как показано ниже:

```

...
// Создание HTTP ответа
strcpy((char *)buf,("HTTP/1.0 200 OK\nContent-Type: text/html; charset=windows-1251\n\n"));

```

```

strcat((char *)buf,("<!DOCTYPE HTML>\n<html><title>Web-server
ATmega1280+W5100</title><body>\n"));
strcat((char *)buf,("<h1>Сервер на ATmega1280+W5100</h1>\r\n"));
strcat((char *)buf,("<A href='t.htm'>Температура в помещении</a>"));

```

После копирования HTML ответа в буфер (buf), необходимо его содержимое передать клиенту. Это выполняется с помощью функции send(). До передачи данных необходимо проверить размер буфера передачи Wiznet W5100 посредством чтения регистра S0\_TX\_FSR. Его значение должно быть равно 2Кбайт (0x07FF):

```

// Определение размера буферного регистра передачи
txsize=SPI_Read(SO_TX_FSR);
txsize=(((txsize & 0x00FF) << 8) + SPI_Read(SO_TX_FSR + 1));
// Если размер его меньше, чем количество поступивших данных(buflen)
// то необходимо обождать хотя бы 1000мс, пока он не освободиться
timeout=0; while (txsize < buflen) {
    _delay_ms(1); txsize=SPI_Read(SO_TX_FSR);
txsize=(((txsize & 0x00FF) << 8) + SPI_Read(SO_TX_FSR + 1));
    if (timeout++ > 1000) {
// Если буфер не освободиться, то принудительно прервать соединение
        disconnect(sock); } }

```

Как в случае приема данных, необходимо рассчитать физический адрес в буфере TX, начиная с которого будут записываться передаваемые данные (см. рис. 7):

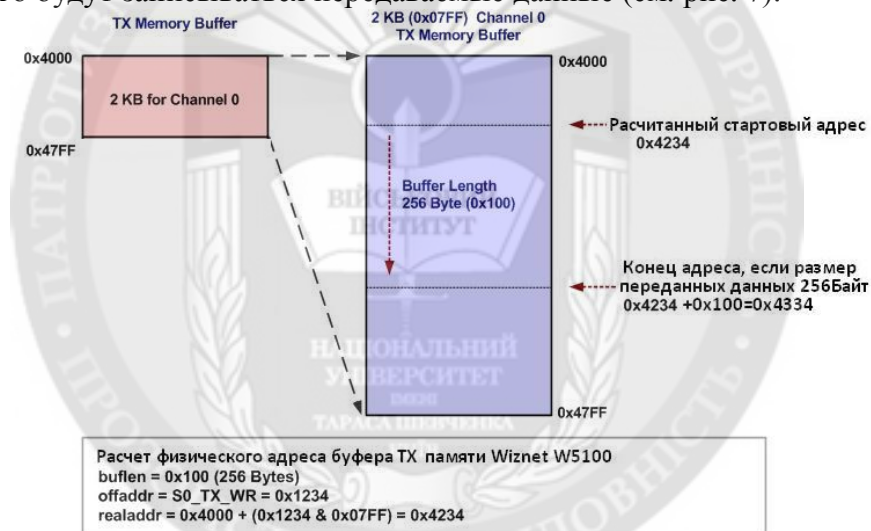


Рис. 7. Расчет физического адреса буфера TX

Для определения физического адреса в буфере передачи TX необходимо считать значение регистра S0\_TX\_TD и битовой операцией & наложить на него 0x7FF (размер буфера передачи – 2048 Байт для нулевого сокета). К полученному значению добавить 0x4000 – начальный адрес буфера передачи. В функции send() это делается так:

```

...
// Чтение данных с регистра S0_TX_TD старшего и младшего байтов
ptr = SPI_Read(S0_TX_WR);
offaddr = (((ptr & 0x00FF) << 8) + SPI_Read(S0_TX_WR + 1));
// Запись в цикле переданных данных в буфер TX из массива *buf
while(buflen) { buflen--;
    // Расчет физического адреса
    realaddr = TXBUFADDR + (offaddr & TX_BUF_MASK);
    // Копирование данных в TX буфер W5100
    SPI_Write(realaddr, *buf);
}

```



```
offaddr++; buf++; }
```

После получения всех данных буфером TX памяти необходимо последний указатель записать обратно в регистр S0\_TX\_WR, чтобы с этого места помещать следующую порцию данных:

```
SPI_Write(S0_TX_WR,(offaddr & 0xFF00) >> 8 );  
SPI_Write(S0_TX_WR + 1,(offaddr & 0x00FF));
```

...

Далее необходимо записать в регистр S0\_CR команду CR\_SEND (0x20), которая отправит данные в сеть из буфера.

```
SPI_Write(S0_CR,CR_SEND);
```

...

В соответствии с требованием протокола HTTP, после отправки HTML ответа клиенту, необходимо отключить и закрыть соединение с клиентом. Это выполняется с помощью функций disconnect() и close() соответственно:

...

```
// Посылка команды Disconnect  
SPI_Write(S0_CR,CR_DISCON);
```

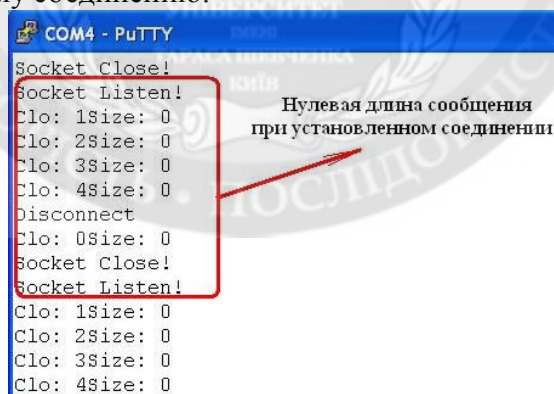
...

```
// Посылка команды Close  
SPI_Write(S0_CR,CR_CLOSE);
```

...

В программе main() выполняется бесконечный цикл открытия и прослушивания нового запроса, который идет от клиента к серверу.

Ранее отмечалось, что рассмотренная программа, за основу которой взят код из источника [1], обладает недостатком - при подключении к серверу через Интернет сервер через короткий промежуток времени «виснет». Для выявления проблемы в разных участках программы выполнялась распечатка проходящих данных. Было обнаружено, что некоторые браузеры при работе с сервером через Интернет выполняют установление соединения при посылке нулевых запросов. Это приводит к заикливанию сервера. На рис. 8 показана распечатка нулевых длин запросов со стороны клиента в части программы, которая соответствует установленному соединению.



```
COM4 - PuTTY  
Socket Close!  
Socket Listen!  
Clo: 1size: 0  
Clo: 2size: 0  
Clo: 3size: 0  
Clo: 4size: 0  
Disconnect  
Clo: 0size: 0  
Socket Close!  
Socket Listen!  
Clo: 1size: 0  
Clo: 2size: 0  
Clo: 3size: 0  
Clo: 4size: 0
```

Рис. 8. Распечатка нулевой длины запроса от браузера

Для выхода из заикливания в программе main() был установлен счетчик clo(см. текст программы). Если значение clo при нулевой длине сообщения, вычисленной функцией recv\_size(), устанавливалось более 5, то вызывалась функция disconnect() для разрыва соединения через экспериментально подобранную задержку.

### 3.Использование Flash памяти микроконтроллера для размещения текста и изображений.

Рассмотрим возможность размещения html – страниц во всей flash памяти микроконтроллера ATmega1280, используя контроллер Arduino Mega и среду разработки AVR Studio 4 с компилятором WinAVR. Особенностью ATmega1280, ATmega2560 является

то, что они 8-ми разрядные, поэтому при их программировании возникают сложности адресации к памяти за пределами 64Кбайт. Тем более не все компиляторы поддерживают обращение к памяти за область 64Кбайт. Для микроконтроллеров ATmega32, ATmega644 и др. аналогичных таких проблем не существует, т.к. их flash память не выходит за пределы 64Кбайт.

Размещение данных в flash памяти выполняется с помощью библиотеки AVR libс. Для подключения библиотеки необходимо в начале программы ее описать с помощью `#include <avr/pgmspace.h>`. HTML документ (текст и изображения) в программе должны быть представлены в виде массива байт, которые можно получить, например с помощью программы `makefsdata.exe`[4]. Для размещения массива байт в flash из файла `pgmspace.h` можно воспользоваться следующими описаниями:

```
#define pgm_read_byte(address_short)    pgm_read_byte_near(address_short)
#define pgm_read_byte_near(address_short) __LPM((uint16_t)(address_short))
- читает байт с flash памяти коротким адресом в пределах 64КБайт
#define pgm_read_byte_far(address_long) __ELPM((uint32_t)(address_long))
- читает байт с flash памяти "дальним" адресом за пределами 64КБайт
#define __LPM(addr)    __LPM_classic__(addr)
#define __ELPM(addr)  __ELPM_classic__(addr)
```

Здесь `__LPM_classic__(addr)` – макрос, который предназначен для чтения байта с памяти программ, используя 16-и битный адрес (т.е. в пределах 64КБайт). Он выглядит следующим образом:

```
#define __LPM_classic__(addr)
(
  __extension__({
    uint16_t __addr16 = (uint16_t)(addr); \
    uint8_t __result;
    __asm__
    (
      "lpm" "\n\t" \
      "mov %0, r0" "\n\t" \
      : "=r" (__result) \
      : "z" (__addr16) \
      : "r0"
    );
    __result;
  })
)
```

`__ELPM_classic__(addr)` – макрос, который предназначен для чтения байта с памяти программ, используя 32-х битный адрес (т.е. за пределами 64КБайт). Он записан следующим образом:

```
#define __ELPM_classic__(addr)
(
  __extension__({
    uint32_t __addr32 = (uint32_t)(addr); \
    uint8_t __result;
    __asm__
    (
      "out %2, %C1" "\n\t" \
      "mov r31, %B1" "\n\t" \
      "mov r30, %A1" "\n\t" \
      "elpm" "\n\t" \
      "mov %0, r0" "\n\t" \
      : "=r" (__result) \
      : "r" (__addr32), \
      "I" (_SFR_IO_ADDR(RAMPZ)) \
    );
    __result;
  })
)
```

```

        : "r0", "r30", "r31" \
    ); \
    __result; \
}))

```

В файле `pgmspace.h`, который включен в компилятор WinAVR модификации 2010-01-20, нет макроса для вычисления 32-х битного “дальнего” адреса (за пределами 64КБайт). Это может выполнить следующий макрос[5], который должен быть вставлен в программу сервера:

```

#define FAR(var) \
({ uint_farp_t tmp; \
  __asm ( \
    "ldi %A0, lo8(%1)" "\n\t" \
    "ldi %B0, hi8(%1)" "\n\t" \
    "ldi %C0, hh8(%1)" \
    : "=d" (tmp) \
    : "i" (&(var))); \
  tmp; \
})

```

При компиляции необходимо указать компоновщику, в каких сегментах необходимо разместить массивы данных. Для этого описываются сегменты в части flash памяти, где расположен код программы и следующий 64-х килобайтный блок:

```

byte tex[] __attribute__((section(".my_section"))) =
{0x3c,0x68,0x74,0x6d,0x6c,0x20,0x78,0x6d,0x6c,0x6e,0x73,0x3a,0x6f,...};
byte pic[] __attribute__((section(".far_section"))) =
{0x89,0x50,0x4e,0x47,0x0d,0x0a,0x1a,0x0a,0x00,0x00,0x00,0x0d,0x49,...};

```

Впоследствии при компоновке необходимо воспользоваться опциями:

```
-Wl,--section-start=.my_section=0x2600 -Wl,--section-start=.far_section=0x10000
```

Секция `.my_section` будет располагать массив `tex[]` с начального адреса `0x5600`, который должен следовать за кодом программы. Секция `.far_section` будет располагать массив `pic[]` с начального адреса `0x10000` в следующем блоке размером 64Кбайт. Опции устанавливаются, если в среде Atmel AVR Studio 4 перейти по ссылкам:

Project -> Configuration Options -> Custom Options -> [Linker Options]

Для прошивки микроконтроллера на плате Arduino Mega используется программатор `avrdude` с конфигурационным файлом, которые заимствованы из программной среды Arduino:

```
avrdude -C avrdude.conf -patmega1280 -carduino -PCOM5 -b57400 -D -V -
Uflash:w:w5100_mega.hex:i
```

В микроконтроллере `atmega1280` должен быть `bootloader` (стандартный загрузчик для Arduino). Как уже отмечалось, полный текст программы представлен в источнике [3].

**Выводы.** Из-за простоты сервера не желательно на одной html страничке размещать более одного изображения. На остальные изображения можно делать ссылки. Это связано с тем, что сервер может обслуживать только последовательные запросы со стороны браузера. Если выполняются параллельные запросы (быстрые браузеры), то все изображения на страничке просто не загружаются. Хорошо работают с этим сервером при наличии картинок браузеры Opera, Firefox. Плохо - Chrome, Яндекс - браузер, Internet Explorer. Если любой браузер подключен через прокси - сервер (Squid), изображения на страничке загружаются.

При неудовлетворительном качестве Интернет-соединения при загрузке объемных HTML страниц сервер часто сбрасывается. Поэтому html - странички должны быть небольшие.

Основным достоинством сервера является стабильность его работы (не виснет), относительная простота программы на Си и малый код программы (без данных он занимает примерно 2.5-3.0 Кбайт).

Представлен способ размещения в flash памяти микроконтроллера данных, размер которых может превышать 64Кбайт.

#### ЛИТЕРАТУРА:

1. R.W. Besinga. Integrating Wiznet W5100, WIZ811MJ network module with Atmel AVR Microcontroller. [Electronic resource]. - Mode of access: <http://www.ermicro.com/blog/?p=1773>, 2010.
2. Arduino. Официальный сайт. [Electronic resource]. - Mode of access: <http://arduino.cc>, 2014.
3. Мясичев А.А. Сервер на ATmega1280 + Wiznet W5100. Практика для студентов. [Electronic resource]. - Mode of access: [http://webstm32.sytes.net/mega\\_t.html](http://webstm32.sytes.net/mega_t.html), 2014.
4. Мясичев А.А. Web – сервер на платах STM32F4Discovery и STM32F4DIS-BB для удаленного управления по TCP/IP сети. [Electronic resource]. - Mode of access: [http://alex56ma.zapto.org/stm32\\_web/stm32\\_3.html](http://alex56ma.zapto.org/stm32_web/stm32_3.html), 2014.
5. AVR-GCC-Tutorial. [Electronic resource]. - Mode of access: [http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial#Programmspeicher\\_28F](http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial#Programmspeicher_28F)

**Рецензент:** д.т.н., проф. Ленков С.В., начальник научно-дослідного центру Військового інституту Київського національного університету імені Тараса Шевченка

д.т.н., проф. Мясичев О.А.

### СТВОРЕННЯ НАДІЙНОГО HTTP - СЕРВЕРА ДЛЯ ВІДДАЛЕНОГО УПРАВЛІННЯ ПО TCP/IP МЕРЕЖІ НА ОСНОВІ АТМЕГА1280 І WIZNET W5100

*У роботі розглядається створення http - сервера на мікроконтролері ATmega1280 і контролері мережі Wiznet W5100. Сервер пересилає браузеру клієнта HTML текст, зображення, показання температурного датчика DS18B20. Управляє мобільним телефоном. Можливо просте розширення його функцій. Програма для сервера написана на Сі (WinAVR). Особливістю сервера є його стійка робота через мережу Інтернет, можливість використання всієї пам'яті програм мікроконтролера (>64Кбайт) для розміщення HTML даних. Програмна середа Arduino на це не здатна.*

*Ключові слова: мікроконтролер, ATmega1280, Wiznet W5100, Arduino, TCP / IP, ethernet, SPI, реєстри, протокол.*

Prof. Myasishev O.A.

### ESTABLISHMENT OF A RELIABLE HTTP - SERVER FOR REMOTE CONTROL VIA TCP / IP-BASED NETWORK ATMEGA1280 AND WIZNET W5100

*The paper considers the creation of http - server microcontroller ATmega1280 and network controller Wiznet W5100. The server sends the client browser HTML text, images, readings of temperature sensor DS18B20. Controls the mobile phone. Perhaps a simple extension of its functions. The program for the server is written in C (WinAVR). Server feature is its steady work through the Internet, the ability to use the entire program memory of the microcontroller (> 64Kbait) to place HTML data. Arduino software environment is not capable of.*

*Keywords: microcontroller, ATmega1280, Wiznet W5100, Arduino, TCP / IP, ethernet, SPI, registers protocol.*