Обмен данными между STM32F103 и компьютером с помощью Wiznet W5500

А.И. Незванов, В.Г. Трубин, П.В. Мищенко, В.А. Жмудь

ФГБОУ ВО НГТУ, Новосибирск, Россия

Аннотация: В статье рассматривается задача обмена данными между микроконтроллером и компьютером. Одним из доступных решений является канал Ethernet. Он позволяет достичь высоких скоростей передачи данных. В ходе работы рассмотрены основные аспекты обмена данными по сети Ethernet, приведён способ подключения сетевого контроллера Wiznet W5500 к микроконтроллеру STM32F103C8T6. Также приведены примеры программ обмена данными.

Ключевые слова: микроконтроллер, *STM32F103, STM32F103C8T6, Ethernet, TCP/IP, Wiznet W5500, TCP, UDP, Python*

ВВЕДЕНИЕ

В процессе разработки электронных устройств зачастую требуется добавить возможность обмена данными с компьютером, например, для получения управляющих команд или сбора данных с удалённых устройств [1–20]. Наряду с использованием системы *Bluetooth* [21–22], одним из наиболее простых решений данной задачи является передача данных по сети *Ethernet*. Инфраструктура сетей *Ethernet* проста в обслуживании, а оборудование имеет невысокую стоимость. Но передача данных по сети *Ethernet* от микроконтроллера (*МК*) имеет ряд трудностей, среди которых – сложность взаимодействия с сетью *Ethernet* и отсутствие понятных руководств для разработчиков. В данной статье рассмотрены основы передачи данных по сети Ethernet, способы подключения сетевого контроллера *W5500* и приведены примеры обмена данными между микроконтроллером и компьютером.

1. СЕТЬ ETHERNET

Ethernet (англ. *Ethernet* от *ether* «эфир» + *network* «сеть») [23] – семейство технологий передачи данных между устройствами для компьютерных и промышленных сетей.

Стандарты *Ethernet* определяют проводные соединения и электрические сигналы на физическом уровне, формат кадров и протоколы управления доступом к среде – на канальном уровне модели *OSI*. *Ethernet* в основном описывается стандартами *IEEE* группы 802.3. *Ethernet* стал самой распространённой технологией локальных вычислительных сетей (*ЛВС*) в середине 1990-х годов, вытеснив такие технологии, как *Token Ring*, *FDDI* и *ARCNET*.

Название «*Ethernet*» (буквально «эфирная сеть» или «среда сети») отражает первоначальный принцип работы этой технологии при использовании шинной топологии: всё, передаваемое одним узлом, одновременно принимается всеми остальными (то есть имеется некое сходство с радиовещанием). В настоящее время практически всегда подключение происходит через коммутаторы (*switch*), так что кадры, отправляемые одним узлом, доходят лишь до требуемого адресата (исключение составляют передачи на широковещательный адрес) – это повышает скорость работы и безопасность сети.

Передаваемые в сети данные имеют название «кадр». Согласно распространённому на сегодня формату *Ethernet V2* кадр имеет вид, представленный на *Рис*. 1.

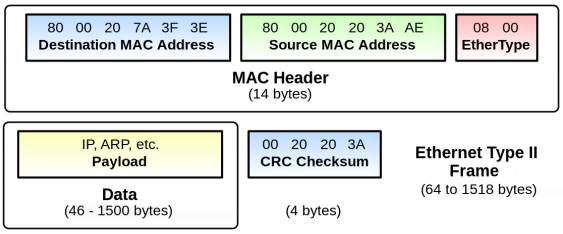

Рис. 1. Формат кадра *Ethernet V2*

Кадр размером от 64 до 1518 *байт* состоит из стандартных заголовков, контрольной суммы и передаваемых данных. При использовании контроллера *Wiznet W5500* разработчик работает только с передаваемыми данными, так как обработку заголовков и контрольной суммы сетевой контроллер полностью берёт на себя.

Обмен данными производится через сокет (англ. *socket* – разъём) [24] – программный интерфейс для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одном компьютере, так и на нескольких, связанных между собой сетью. Сокет является абстрактным объектом, представляющим конечную точку соединения.

Принято различать клиентские и серверные сокеты. Клиентское приложение (например, браузер) использует только клиентские сокеты, а серверное (например, веб-сервер, которому браузер посылает запросы) – как клиентские, так и серверные сокеты.

Упомянутое взаимодействие сокетов реализуется посредством протоколов стека *TCP/IP* (*Transmission control protocol/Internet protocol*). Рассмотрим два основных транспортных протокола передачи данных по сети – *TCP* и *UDP*, не углубляясь в особенности работы стека *TCP/IP*.

1.2. ПРОТОКОЛ TCP

Transmission Control Protocol (*TCP*, протокол управления передачей) [25] – один из наиболее востребованных протоколов передачи данных, гарантирующих доставку данных получателю.

Механизм *TCP* предоставляет поток данных с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым, в отличие от *UDP*, целостность передаваемых данных и уведомление отправителя о результатах передачи (квитирование) с последующим разрывом соединения только после положительного ответа на соответствующий запрос.

TCP осуществляет передачу потока байтов от одного процесса к другому, реализует управление потоком, перегрузкой, тройное рукопожатие, гарантированную передачу. Схема работы сервера по протоколу *TCP* представлена на *Рис*. 2, клиента на *Рис*. 3.

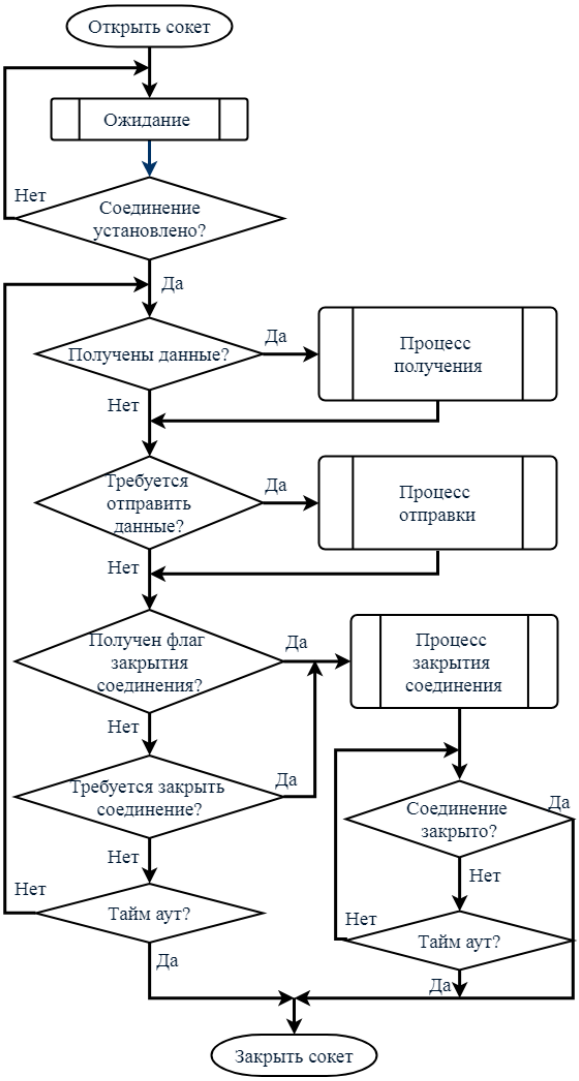

Рис. 2. Схема работы сервера по протоколу TCP

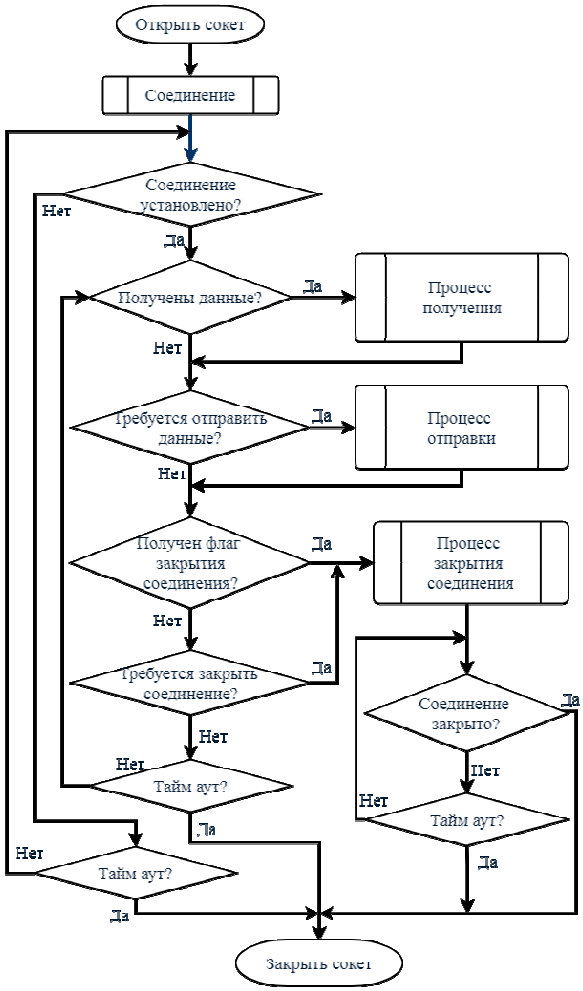

Рис. 3. Схема работы клиента по протоколу TCP

1.2 ПРОТОКОЛ UDP

UDP (англ. *User Datagram Protocol* – протокол пользовательских датаграмм) [26] – один из ключевых протоколов в стеке *TCP/IP*. С *UDP* компьютерные приложения могут посылать сообщения (в данном случае называемые датаграммами) другим устройствам по сети без необходимости предварительного установления соединения и без подтверждения о получении. Протокол был разработан Дэвидом П. Ридом в 1980 году и официально определён в *RFC 768*.

UDP использует простую модель передачи, без неявных «рукопожатий» для обеспечения надёжности, упорядочивания или целостности данных. Таким образом, *UDP* не гарантирует надежность передачи – датаграммы могут прийти не по порядку, дублироваться или вовсе исчезнуть без следа. Тем не менее, *UDP* подразумевает, что возникновение потерь маловероятно. В некоторых случаях проверка наличия ошибок и их исправление могут исполняться в приложении.

Природа *UDP* как протокола без сохранения состояния также полезна для серверов, отвечающих на небольшие запросы от огромного числа клиентов, например, *DNS* и потоковые мультимедийные приложения. Схема обмена

данными по протоколу *UDP* представлена на *Рис*. 4.

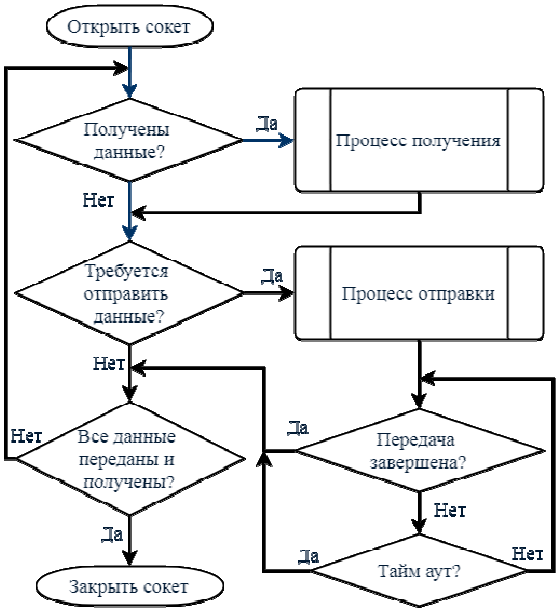

Рис. 4. Схема передачи данных по протоколу *UDP*

2. ПЕРЕДАЧА ДАННЫХ ПО СЕТИ

Рассмотрим передачу данных между двумя компьютерами. В качестве языка программирования используем *Python* 3 [27], так как он предоставляет очень удобные инструменты для работы с сетью. В качестве задачи выберем передачу числа от клиента на сервер.

2.1. ПЕРЕДАЧА С ИСПОЛЬЗОВАНИЕМ ПРОТОКОЛА TCP

Так как протокол *TCP* гарантирует доставку данных, то его использование рационально для двусторонней связи, работающей по схеме «запрос-ответ». Соответственно, в данном примере клиент отправляет серверу запрос, содержащий число, а сервер отправляет это число обратно клиенту. На *Рис*. 5 приведены исходные коды программ сервера и клиента.

Алгоритм запуска может быть описан следующим образом:

- Создание файлов *server-tcp.py*, *client-tcp.py*, содержащих указанный выше код, с помощью блокнота или аналогичных приложений в кодировке Windows-1251. Возможно сохранение в кодировке UTF-8, при этом необходимо удалить первую строку «# coding: cp1251».
- Запуск файлов *server-tcp.py*, *client-tcp.py* из проводника *Windows* или аналогичных приложений. Очерёдность запуска не влияет на ход работы.

Рис. 6 демонстрирует полученный результат.

Файл: server-tcp.py

```
# coding: cp1251
import socket
import time
# открытие сокета, тип TCP (SOCK_STREAM)
tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# задание прослушиваемого порта сервера
tcp_socket.bind(('localhost', 10000))
# прослушивание входящих соединений, максимум 1
tcp_socket.listen(1)
# бесконечный цикл
while 1:
 # задание тайм-аута
 tcp_socket.settimeout(2)
 # конструкция try: ... except: для обработки исключений
 try:
  # приём входящего подключения
  conn,addr = tcp_socket.accept()
  # получение 2 байт
  data = conn.recv(2)
  # если не получены данные - закрытие соединения
  if not data:
   conn.close()
   break
  # конвертация полученных от клиента байт в число
  n = int.from_bytes(data,byteorder='big')
  print("Получено от клиента: ", n)
  # конвертация числа в тип bytes (2 байта)
  data_send = (n).to_bytes(2, byteorder='big')
  # передача числа клиенту
  conn.send(data_send)
  # закрытие соединения
  conn.close()
 except socket.timeout:
  print("Тайм-аут")
# закрытие сокета
tcp_socket.close()
```

Файл: client-tcp.py

```
# coding: cp1251
import socket
import sys
import time
# задание адреса сервера и порта
addr = ('localhost', 10000)
# объявление переменных
n = 1; rec = 0
# бесконечный цикл
while True:
 # открытие сокета, тип TCP (SOCK_STREAM)
 tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
 # конструкция try: ... except: для обработки исключений
 try:
  # установка соединения с сервером
  tcp_socket.connect(addr)
  # конвертация числа в тип bytes (2 байта)
  data_send = (n).to_bytes(2, byteorder = 'big')
  # передача числа на сервер
  tcp_socket.send(data_send)
  # получение 2 байт от сервера
  data = tcp_socket.recv(2)
  # конвертация полученных от сервера байт в число
  rec = int.from_bytes(data, byteorder = 'big')
  print("Получено от сервера: ", rec)
  # закрытие сокета
  tcp_socket.close()
  # увеличение счётчика
  n+=1
  # задержка 1 секунда
  time.sleep(1)
 except socket.error:
  print("Ошибка подключения!")
```

Рис. 5. Исходные коды программ сервера и клиента

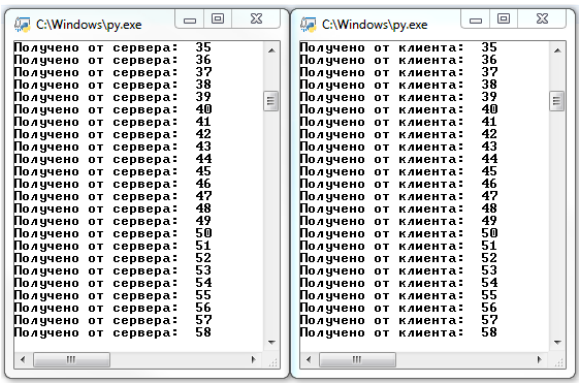

Рис. 6. Результат работы программ

2.2. ПЕРЕДАЧА С ИСПОЛЬЗОВАНИЕМ ПРОТОКОЛА UDP

Протокол *UDP* не гарантирует доставку данных, что может стать проблемой при двусторонней связи. В нижеприведённом примере реализована отправка числа на сервер, без возврата его клиенту. Ниже приведены исходные коды программ сервера и клиента.

Файл: server-udp.py

```
# coding: cp1251
import socket
# открытие сокета, тип UDP (SOCK_DGRAM)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# задание прослушиваемого порта сервера
sock.bind(('localhost', 9999))
# бесконечный цикл
while 1:
 # получение 2 байт от клиента
 data,address = sock.recvfrom(2)
 # конвертация полученных от клиента байт в число
 n = int.from_bytes(data,byteorder = 'big')
 print("Получено от клиента: ", n)
# закрытие сокета
sock.close()
```

Файл: client-udp.py

```
# coding: cp1251
import socket
import time
import sys
# задание адреса сервера и порта
addr = ('localhost', 9999)
# объявление переменных
n = 1
# открытие сокета, тип UDP (SOCK_DGRAM)
udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# бесконечный цикл
while True:
 print ("Передача на сервер: ", n)
 # конвертация числа в тип bytes (2 байта)
 data_send = (n).to_bytes(2,byteorder = 'big')
 # передача числа на сервер
 udp_socket.sendto(data_send, addr)
 # увеличение счётчика
 n+=1
 # задержка 1 секунда
 time.sleep(1)
# закрытие сокета
udp_socket.close()
```

Алгоритм запуска может быть описан следующим образом:

- Создание файлов *server-udp.py*, *client-udp.py*, содержащих указанный выше код, с помощью блокнота или аналогичных приложений в кодировке *Windows*-1251. Возможно сохранение в кодировке *UTF*-8, при этом необходимо удалить первую строку «# coding: cp1251».
- Запуск файлов *server-udp.py*, *client-udp.py* из проводника *Windows* или аналогичных приложений. Очерёдность запуска не влияет на ход работы.

Рис. 7 демонстрирует полученный результат.

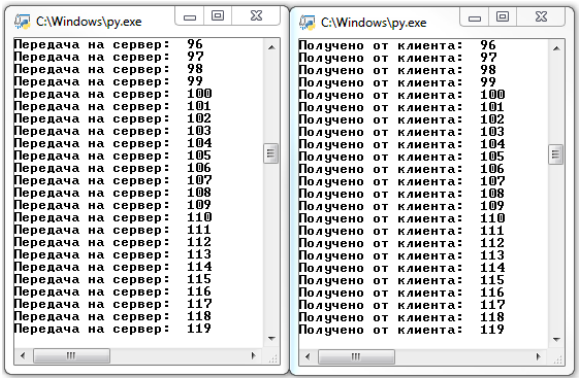

Рис. 7. Результат работы программ

3. СЕТЕВОЙ КОНТРОЛЛЕР WIZNET W5500

Wiznet W5500 – это микросхема-сетевой контроллер, позволяющая достаточно легко добавить в устройство проводное интернет-подключение. Интерфейс обмена данными с микроконтроллером – *SPI*, поддерживающий скорость обмена данными до 80 *МГц* (производитель оговаривает, что гарантированная скорость – 33,3 *МГц*, [28] – 5.5.4 *SPI Timing*).

*Возможности W*5500*:*

- Аппаратная поддержка протоколов: *UDP*, *TCP*, *ICMP*, *IPv*4, *ARP*, *IGMP*, *PPPoE*;
- Поддержка 8 одновременно действующих сокетов;
- Поддержка режима пониженного энергопотребления;
- Поддержка включения устройства по сети через *UDP*;
- Поддержка высокоскоростного *SPI*;
- Встроенные 32 *Кбайт* памяти для буферов сокетов;
- Поддержка 10 и 100 *Мбит/сек* подключения;
- Автоматическое определение режимов передачи (дуплекс, полудуплекс, 10/100 *Мбит/сек*)
- Напряжение питания 3.3 *В* и возможность нормального функционирования при 5 *В* уровнях на линиях от *МК*.

Рассмотрим широко распространённый модуль на базе *W*5500 (представлен на *Рис*. 8). Он имеет в себе все необходимые для работы компоненты и стоит относительно недорого – около 3 *$*. За эту цену разработчик получает полностью готовый контроллер, который необходимо подключить к *МК* через интерфейс *SPI*.

Рис. 8. Сетевой контроллер на базе *Wiznet W*5500

3.1 ПОДКЛЮЧЕНИЕ WIZNET W5500 К МК STM32F103C8T6

Сетевой контроллер имеет следующие выводы:

- 5*V* – линия питания 5 *B*, преобразуется установленным на плату стабилизатором *AMS*1117 в 3.3 *B*;
- 3.3*V* – выход со стабилизатора *AMS1117*;
- *GND* – общий провод;
- *RST* – управление перезагрузкой сетевого контроллера. Требуется для сброса контроллера в случае зависания;
- *INT* – выход с *W*5500, сигнализирующий о наличии входящего соединения на сетевом контроллере. Актуален при работе в режиме сервера, что в данной работе не потребуется;
- *NC* – не используется (*NOT CONNECTED*);
- *MISO* – линия интерфейса *SPI*;
- *MOSI* – линия интерфейса *SPI*;
- *SCS* – линия интерфейса *SPI*, выбор передающего устройства (*CHIP SELECT*);
- *SCLK* – линия интерфейса *SPI*, тактовые импульсы (формируются *MK*).

Отдельно необходимо упомянуть о недопустимости подачи напряжения +3.3 *B* на соответствующий контакт сетевого контроллера. Согласно рекомендации производителя стабилизатора *AMS*1117 [29], для его защиты требуется диод (*D*1 на *Рис*. 9). На используемой плате контроллера *W*5500 этот защитный диод отсутствует. Исходя из этого следует использовать только вывод 5 *V* для подачи питания на сетевой контроллер.

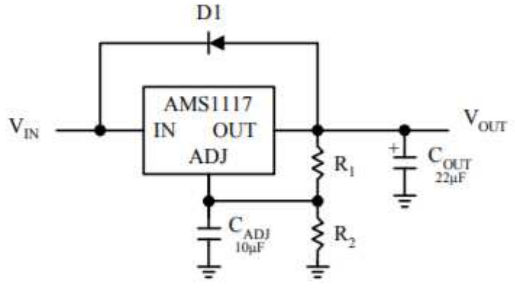

Рис. 9. Схема подключения *AMS*1117

Микроконтроллер *STM32F103* [30] имеет 2 аппаратных интерфейса *SPI*, для подключения используем *SPI1* (порты *PA4-PA7*), а для управления перезагрузкой *W5500* выберем порт *PA*3. Таблица подключения представлена ниже. Схема для тестирования в собранном виде представлена на *Рис*. 10.

Таблица. 1

Соответствие портов *W*5500 и портов *STM32F*103

Wiznet W5500	STM32F103
5 *V*	+5 *V*
MISO	*PA6*
MOSI	*PA7*
SCS	*PA4*
SCLK	*PA5*
GND	*GND*
RST	*PA3*

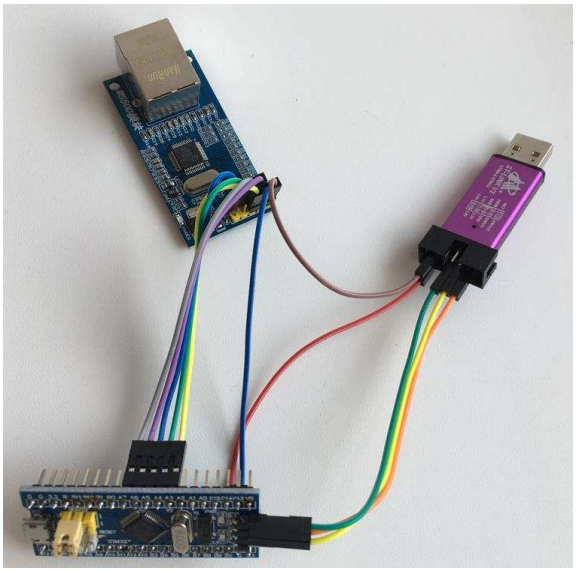

Рис. 10. Собранная схема для тестирования

3.2 ОБМЕН ДАННЫМИ МЕЖДУ МИКРОКОНТРОЛЛЕРОМ И КОМПЬЮТЕРОМ

Достаточно часто приходится взаимодействовать с внешними устройствами. Существует проверенный временем способ – интерфейс *UART*, но он непригоден для использования на большом расстоянии и имеет невысокую скорость обмена (до 115200 *бит/сек*). С появлением *W5500* стало возможным передавать данные на значительно большей скорости и большие расстояния.

Рассмотрим следующий пример: *МК* получает по сети от компьютера команды для управления светодиодом (который можно заменить на реле и т. д.). При получении 0 светодиод выключается, при 1 – включается. Серверная часть написана на языке *Python*. Ниже приведён исходный код сервера.

Файл: server.py

```
# coding: cp1251
import socket
import time
# открытие сокета, тип TCP (SOCK_STREAM)
sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
# задание прослушиваемого порта сервера
sock.bind(('', 10000))
# прослушивание входящих соединений, максимум 1
sock.listen(1)
n = 1
# бесконечный цикл
while 1:
 # задание тайм-аута
 sock.settimeout(2)
 # конструкция try: ... except: для обработки исключений
 try:
  # приём входящего подключения
  conn,addr = sock.accept()
  # получение данных от клиента
  data = conn.recv(1024)
  # если не получены данные - закрытие соединения
  if not data:
   conn.close()
   break
  print (data)
  # конструкция для формирования ответа
  if n%2 == 1:
   temp = "1"
  else:
   temp = "0"
  # перевод данных для отправки в тип bytes
  data_send = temp.encode()
  print("Sending:",data_send)
  # передача данных клиенту
  conn.send(data_send)
  # закрытие соединения
  conn.close()
  n += 1
 except socket.timeout:
  print("Socket timeout")
# закрытие сокета
sock.close()
```

Для взаимодействия микроконтроллера с сетевым контроллером требуется скачать библиотеку от производителя чипа [31]. Для индикации используем светодиод, подключенный к порту *PC13*. Программа имеет следующий алгоритм работы:

1. Инициализация *MK*;
2. Инициализация сетевого контроллера;
3. Попытка соединения с сервером (в случае неудачи производится перезагрузка сетевого контроллера);
4. Получение данных, которые используются для управления светодиодом.

Для работоспособности примера потребуется указать актуальный *IP* адрес сервера (компьютера, на котором запускается программа *server.py*) в переменной *destip*, порт изменять не требуется. Также необходимо указать параметры для сетевого контроллера *W*5500: *MAC-адрес* (можно указать любой, в допустимом диапазоне значений) в переменной mac, *IP* адрес и маску подсети (для сети, к которой подключается контроллер) в переменных *ip* и *netmask*. Переменные *gateway* (шлюз) и *dnsserv* (адрес *DNS* сервера) для данного примера не требуются, их значения можно оставить без изменений. Программный код для *STM*32*F*103 приведён ниже.

```
#include "stm32f10x.h"
#include "stm32f10x_spi.h"
#include "socket.h"
#include "socket.c"
#include "wizchip_conf.h"
#include "wizchip_conf.c"
#include "W5500/w5500.h"
#include "W5500/w5500.c"

#define LED1 GPIO_Pin_13
#define IN1 GPIO_Pin_14

//=============== настраиваемые параметры =================
uint8_t destip[4] = {192, 168, 1, 102}; // IP адрес сервера
uint32_t destport = 10000; // порт сервера

//================== переменные =================
volatile uint32_t timer = 0, timerT = 0;
volatile uint16_t sendcount = 0;

uint8_t mac[6] = {0x40, 0x08, 0xdc, 0x00, 0xab, 0xcd}, // можно задать любой
        ip[4] = {192, 168, 1, 151}, // задаем IP адрес W5500
        netmask[4] = {255, 255, 255, 0},
        gateway[4] = {0, 0, 0, 0},  // адрес шлюза
        dnsserv[4] = {0, 0, 0, 0};  // адрес DNS

wiz_NetInfo gWIZNETINFO;
GPIO_InitTypeDef Init_PORTB, Init_PORTA, Init_PORTC;
USART_InitTypeDef       USART_InitStructure;
SPI_InitTypeDef         SPI_InitStructure;
NVIC_InitTypeDef        NVIC_InitStructure;
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
```

```
ErrorStatus         HSEStartUpStatus;

//================== Переключение на внешний генератор =======================
void SetSysClockToHSE(void)
{
  /* SYSCLK, HCLK, PCLK2 and PCLK1 configuration ----------------------------*/
  /* RCC system reset(for debug purpose) */
  RCC_DeInit();
  RCC_HSEConfig(RCC_HSE_ON); /* Enable HSE */
  HSEStartUpStatus = RCC_WaitForHSEStartUp(); /* Wait till HSE is ready */
  if (HSEStartUpStatus == SUCCESS)
  {
        FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable); /* Enable Prefetch Buffer */
    FLASH_SetLatency(FLASH_Latency_2); /* Flash 2 wait state */
    RCC_HCLKConfig(RCC_SYSCLK_Div1); /* HCLK = SYSCLK */
    RCC_PCLK2Config(RCC_HCLK_Div1); /* PCLK2 = HCLK */
    RCC_PCLK1Config(RCC_HCLK_Div2); /* PCLK1 = HCLK/2 */
        RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9); /* PLLCLK = 8MHz * 9 = 72 MHz */
    RCC_SYSCLKConfig(RCC_SYSCLKSource_HSE); /* Select HSE as system clock source */
    RCC_PLLCmd(ENABLE); /* Enable PLL */
        while (RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET) {}; /* Wait till PLL is ready */
        RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK); /* Select PLL as system clock source */
    while(RCC_GetSYSCLKSource() != 0x08) {}; /* Wait till PLL is used as system clock source */
  }
  else
  { /* If HSE fails to start-up, the application will have wrong clock configuration.
       User can add here some code to deal with this error */
    while (1) { } /* Go to infinite loop */
  }
}

void TIM2_IRQHandler(void) {
//========== TIM2:обработчик прерывания (выполняется каждую 1 мс.) ========
        if (TIM_GetITStatus(TIM2, TIM_IT_Update) == SET) {
                TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
                timer++; timerT++;
                if(timerT >= 1000) { sendcount = 0;      timerT = 0; }
        }
}

void delay_ms(uint32_t time) { timer = 0; while(time > timer) {}; }

void SPIReadWrite (uint8_t data) {
        while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);
        SPI_I2S_SendData(SPI1, data);
        while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET);
        SPI_I2S_ReceiveData(SPI1);
}

void chip_on(void) { GPIO_ResetBits(GPIOA, GPIO_Pin_4); }

void chip_off(void) { GPIO_SetBits(GPIOA, GPIO_Pin_4); }

void chip_reset(void) {
        GPIO_ResetBits(GPIOA, GPIO_Pin_3); delay_ms(1000);
        GPIO_SetBits(GPIOA,GPIO_Pin_3);  delay_ms(2000);
}

// Инициализация W5500
int w5500_ini(void){
        uint8_t temp;
        // задание размеров буферов W5500 для сокетов по 2 Кбайта
        uint8_t W5500FifoSize[2][8] = {{2, 2, 2, 2, 2, 2, 2, 2}, {2, 2, 2, 2, 2, 2, 2, 2}};

        chip_reset();
        for (uint8_t i = 0; i <=5; i++) gWIZNETINFO.mac[i] = mac[i];
        for (uint8_t i = 0; i <=3; i++) {
```

```
          gWIZNETINFO.ip[i] = ip[i]; gWIZNETINFO.sn[i] = netmask[i];
          gWIZNETINFO.gw[i] = gateway[i]; gWIZNETINFO.dns[i] = dnsserv[i];
        }
        gWIZNETINFO.dhcp = NETINFO_STATIC;
        chip_off();
        // "привязка" функций SPI записи и чтения к библиотеке W5500
        reg_wizchip_spi_cbfunc(SPIReadWrite, SPIReadWrite);
        reg_wizchip_cs_cbfunc(chip_on, chip_off);
        // если размер буферов FIFO не установлен, то выход
        if (ctlwizchip(CW_INIT_WIZCHIP, (void*)W5500FifoSize) == -1) return -1;

        do // ожидаем подключения к среде передаче данных
          {     if (ctlwizchip(CW_GET_PHYLINK, (void*)&temp) == -1)      return -1; }
    while (temp == PHY_LINK_OFF);

        wizchip_setnetinfo(&gWIZNETINFO); // передача параметров MAC, IP и т.д. в W5500
        return 1;
}

int w5500_connect(uint8_t http_socket, uint8_t addr[4], uint32_t port) {
//открытие TCP сокета
        uint8_t code = socket(http_socket, Sn_MR_TCP, 10888, 0);
        if (code != http_socket) return -1;
//открытие соединения
        code = connect(http_socket, addr, port);
        if (code != SOCK_OK) { close(http_socket); return -1; }
//отправка запроса на сервер
    {
        char req[] = {"GET123"}; // текст запроса к серверу, может быть любым
        uint16_t len = sizeof(req) - 1;
        uint8_t* pbuff = (uint8_t*) &req;
        while (len > 0) {
          int32_t nbytes = send(http_socket, pbuff, len);
          if (nbytes <= 0) { close(http_socket); return -1; }
          len -= nbytes;
          }
        }
        {
        char buff[32]; uint8_t a = 0;
        //получение ответа от сервера
        while (1) {
          int32_t nbytes = recv(http_socket, (uint8_t*) & buff, sizeof(buff)-1);
          if (nbytes == SOCKERR_SOCKSTATUS) break;
          if (nbytes <= 0) break;
        }
        //управление включением/выключением светодиода
        if (buff[0] == '0') GPIO_SetBits(GPIOC, LED1);
          else if(buff[0] == '1') GPIO_ResetBits(GPIOC, LED1);
        }
//закрытие соединения
        close(http_socket);
        return 1;
}

int main(void)
{
        SetSysClockToHSE();
        RCC_ADCCLKConfig(RCC_PCLK2_Div2);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);

        Init_PORTC.GPIO_Pin = LED1;
        Init_PORTC.GPIO_Speed = GPIO_Speed_10MHz;
        Init_PORTC.GPIO_Mode = GPIO_Mode_Out_PP;
```

```
        GPIO_Init(GPIOC, &Init_PORTC);
        GPIO_SetBits(GPIOC, LED1);

        Init_PORTC.GPIO_Pin = IN1;
        Init_PORTC.GPIO_Speed = GPIO_Speed_50MHz;
        Init_PORTC.GPIO_Mode = GPIO_Mode_IN_FLOATING;
        GPIO_Init(GPIOC, &Init_PORTC);

        /*==================== Прерывание каждую миллисекунду ==================*/
        TIM_TimeBaseStructure.TIM_Period = 50-1; // 1 миллисек.
        TIM_TimeBaseStructure.TIM_Prescaler = 1440 - 1; // квант = 20 мкс
        TIM_TimeBaseStructure.TIM_ClockDivision = 0;
        TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
        TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
        TIM_ARRPreloadConfig(TIM2, ENABLE);
        /*====================== Настройка прерывания ======================*/
        NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStructure);
        TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
        TIM_Cmd(TIM2, ENABLE); /* TIM enable counter */

        //RST pin W5500
        Init_PORTA.GPIO_Pin = GPIO_Pin_3;
        Init_PORTA.GPIO_Mode = GPIO_Mode_Out_PP;
        Init_PORTA.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOA, &Init_PORTA);
        GPIO_SetBits(GPIOA, GPIO_Pin_3);
        //SCS(NSS) pin W5500
        Init_PORTA.GPIO_Pin = GPIO_Pin_4;
        Init_PORTA.GPIO_Mode = GPIO_Mode_Out_PP;
        Init_PORTA.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOA, &Init_PORTA);
        //MISO pin W5500
        Init_PORTA.GPIO_Pin = GPIO_Pin_6;
        Init_PORTA.GPIO_Speed = GPIO_Speed_50MHz;
        Init_PORTA.GPIO_Mode = GPIO_Mode_IPD;
        GPIO_Init(GPIOA, &Init_PORTA);
        //SCLK (SCK), MOSI pin W5500
        Init_PORTA.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_7;
        Init_PORTA.GPIO_Speed = GPIO_Speed_50MHz;
        Init_PORTA.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_Init(GPIOA, &Init_PORTA);
        SPI_I2S_DeInit(SPI1);
        SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
        SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
        SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
        SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
        SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
        SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;
        SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
        SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
        SPI_Init(SPI1, &SPI_InitStructure);
        SPI_Cmd(SPI1, ENABLE);
        //инициализация W5500
        w5500_ini();
        while (1) {
                //подключение к серверу, сокет 0,  адрес сервера destip, порт destport
                if (w5500_connect(0, destip, destport) != 1) w5500_ini();
                //конструкция выше требуется для перезагрузки сетевого
                //контроллера в случае невозможности соединения.
                delay_ms(1000);
        }
}
```

Данный пример достаточно просто модифицировать для требуемого количества каналов. В некоторых случаях целесообразным может оказаться замена *TCP* на *UDP*.

ВЫВОДЫ

1. *Wiznet W*5500 является доступным для покупки, недорогим и эффективным средством для организации быстрого канала связи микроконтроллер-компьютер.
2. С помощью *Wiznet W*5500 можно организовать обмен данными по протоколам *TCP* или *UDP*.
3. Работа с *Wiznet W*5500 отличается простотой, так разработчику не требуется разбираться с особенностями функционирования сети.

ЛИТЕРАТУРА

[1] Жмудь В. А. STM32VLDISCOVERY - платформа для построения простой системы сбора данных: учеб. пособие / В. А. Жмудь, В. Г. Трубин. - Москва: Русайнс, 2018. - 272 с. - ISBN 978-5-4365-2893-9.

[2] Жмудь В. А. Устройства сопряжения с объектом: учеб. пособие / В. А. Жмудь. - Новосибирск: Изд-во НГТУ, 2019. - 172 с. - 100 экз. - ISBN 978-5-7782-3809-1.

[3] Ескин А.В., Жмудь В.А., Трубин В.Г. STM32VLDISCOVERY - средство для быстрой разработки опытных образцов цифровых систем управления. Автоматика и программная инженерия. 2013. № 3 (5). С. 32-39.

[4] Жмудь В.А., Федоров Д.С., Ивойлов А.Ю., Трубин В.Г. Разработка системы стабилизации угла отклонения балансирующего робота. Автоматика и программная инженерия. 2015. № 2 (12). С. 16-34.

[5] Ескин А.В., Жмудь В.А., Трубин В.Г. Построение платформы моделирующей работу роботизированных средств на базе конструктора ЛЕГО МАЙНДСТОРМС 2.0 в части управления электродвигателями. Автоматика и программная инженерия. 2013. № 1 (3). С. 88-94.

[6] Печников А.Л., Жмудь В.А., Трубин В.Г. Удалённое управление роботом посредством xmpp-протокола. Сборник научных трудов Новосибирского государственного технического университета. 2013.№ 3 (73). С. 85-92.

[7] Ивойлов А.Ю., Ескин А.В., Жмудь В.А., Трубин В.Г. Особенности работы с ЖКИ дисплеем NOKIA 5110. Автоматика и программная инженерия. 2013. № 4 (6). С. 8-13.

[8] Колкер А.Б., Ливенец Д.А., Кошелева А.И., Жмудь В.А. исследование вариантов создания интеллектуальных систем робототехники на базе одноплатных компьютеров и свободных операционных систем. Автоматика и программная инженерия. 2012. № 1 (1). С. 84-98.

[9] Жмудь В.А., Трубин М.В., Трубин И.В. Проектирование сенсорных кнопок на базе микросхемы TTP-224. Автоматика и программная инженерия. 2015. № 1 (11). С. 70-74.

[10] Ивойлов А.Ю., Жмудь В.А., Трубин В.Г. разработка системы автоматической стабилизации в вертикальном положении двухколесной платформы. Автоматика и программная инженерия. 2014. № 2 (8). С. 15-21.

[11] Ескин А.В., Жмудь В.А., Трубин В.Г., Печников А.Л. Использование сети интернет для интерактивной лабораторной работы с дистанционным управлением моделью робота снегоуборщика. Автоматика и программная инженерия. 2014. № 1 (7). С. 95-103.

[12] Жмудь В.А., Каменская А.С., Курбетьев К.В., Трубин В.Г. Графический OLED дисплей UG-2864ASGGG14: первое включение. Автоматика и программная инженерия. 2016. № 1 (15). С. 29-37.

[13] Ескин А.В., Жмудь В.А., Трубин В.Г. Плагины Eclipse для ускорения разработки программ цифровых систем управления. Автоматика и программная инженерия. 2013. № 4 (6). С. 24-34.

[14] Жмудь В.А., Кузнецов К.А., Кондратьев Н.О., Трубин В.Г., Димитров Л.В. Ультразвуковой датчик измерения расстояния HC-SR04. Автоматика и программная инженерия. 2017. № 4 (22). С. 18-26.

[15] Федоров Д.С., Ивойлов А.Ю., Жмудь В.А., Трубин В.Г. Использование акселерометра ADXL335 для определения угла. отклонения от вертикали. Автоматика и программная инженерия. 2014. № 2 (8). С. 68-72.

[16] Васильев В.А., Воевода А.А., Жмудь В.А., Хассуонех В.А. Цифровые регуляторы: целевые функции настройки, выбор метода интегрирования, аппаратная реализация. Сборник научных трудов Новосибирского государственного технического университета. 2006.№ 4 (46). С. 3-10.

[17] Жмудь В.А., Французова Г.А., Востриков А.С. Динамика мехатронных систем. Учебное пособие. Новосибирск, 2014.

[18] Zhmud V., Yadrishnikov O., Poloshchuk A., Zavorin A. Modern key technologies in automatics: structures and numerical optimization of regulators. В сборнике: Proceedings - 2012 7th International Forum on Strategic Technology, IFOST 2012 2012. С. 6357804.

[19] Воевода А.А., Жмудь В.А. Астатическое управление объектами с нестационарными матричными передаточными функциями методом приближенного обращения функциональных комплексных матриц. Научный вестник Новосибирского государственного технического университета. 2006.№ 2 (23). С. 3-8.

[20] Жмудь В.А., Трубин М.В., Трубин И.В. Обмен данными между компьютером и микроконтроллером STM32F100 по последовательному интерфейсу связи RS-232. Автоматика и программная инженерия. 2015. № 1 (11). С. 45-51.

[21] Ескин А.В., Жмудь В.А., Трубин В.Г. Беспроводной удлинитель последовательного порта на базе радиоканала Bluetooth. Автоматика и программная инженерия. 2013. № 2 (4). С. 42-47.

[22] Жмудь В.А., Трубин В.Г., Ескин А.В. Реализация дистанционного управления по радиоканалу Bluetooth платформой, моделирующей работу роботизированных средств. Автоматика и программная инженерия. 2013. № 1 (3). С. 82-87.

[23] Wikipedia - Ethernet. URL: https://ru.wikipedia.org/wiki/Ethernet

[24] Wikipedia - Сокет. URL: https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BA%D0%B5%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%

D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81)
[25] Wikipedia – TCP. URL: https://ru.wikipedia.org/wiki/Transmission_Control_Protocol
[26] Wikipedia – UDP. URL: https://ru.wikipedia.org/wiki/UDP
[27] Официальный сайт Python.org. URL: https://www.python.org/
[28] Wiznet W5500 datasheet. URL: https://wizwiki.net/wiki/lib/exe/fetch.php?media=products:w5500_ds_v100e.pdf
[29] AMS1117 datasheet. URL: http://dalincom.ru/datasheet/AMS1117.pdf
[30] STM32F103 datasheet. URL: https://www.st.com/resource/en/datasheet/CD00161566.pdf
[31] Github – ioLibrary Driver for Wiznet W5XXX. URL: https://github.com/Wiznet/ioLibrary_Driver

Вадим Жмудь - заведующий кафедрой Автоматики НГТУ, профессор, доктор технических наук.

E-mail: oao_nips@bk.ru

630073, Новосибирск, просп. К.Маркса, д. 20

Александр Игоревич Незванов – студент 1 курса магистратуры кафедры Автоматики НГТУ.

E-mail: nezvanovml@mail.ru

Новосибирск, 630073, просп. К. Маркса, д. 20, НГТУ

Виталий Геннадьевич Трубин – старший преподаватель кафедры Автоматики НГТУ, директор ООО «КБ Автоматика».

E-mail: trubin@ngs.ru

Новосибирск, 630073, просп. К. Маркса, д. 20, НГТУ

Полина Валерьевна Мищенко – старший преподаватель кафедры Вычислительной техники НГТУ.
E-mail: p.mishhenko@corp.nstu.ru
Новосибирск, 630073, просп. К. Маркса, д. 20, НГТУ

Статья получена 12.05.2019.

Data Exchange between STM32F103 and the Computer Using Wiznet W5500

V.A. Zhmud, A.I. Nezvanov, V.G. Trubin, P.V. Mischenko

Novosibirsk State Technical University, Novosibirsk, Russia

Abstract: This material describes the problem of data exchange between the microcontroller and the computer. Available solution is the Ethernet link. It allows to reach high data transmission rates. During work the main aspects of data exchange of an Ethernet network are considered, the way of connection of the Wiznet W5500 network controller to the STM32F103C8T6 microcontroller is given. Examples of programs of data exchange are also given.

Key words: microcontroller, STM32F103, STM32F103C8T6, Ethernet, TCP/IP, Wiznet W5500, TCP, UDP, Python.

REFERENCES

[1] Zhmud' V. A. STM32VLDISCOVERY - platforma dlya postroyeniya prostoy sistemy sbora dannykh: ucheb. posobiye / V. A. Zhmud', V. G. Trubin. - Moskva: Rusayns, 2018. - 272 s. - ISBN 978-5-4365-2893-9.
[2] Zhmud' V. A. Ustroystva sopryazheniya s ob"yektom: ucheb. posobiye / V. A. Zhmud'. - Novosibirsk: Izd-vo NGTU, 2019. - 172 s. - 100 ekz. - ISBN 978-5-7782-3809-1.
[3] Yeskin A.V., Zhmud' V.A., Trubin V.G. STM32VLDISCOVERY - sredstvo dlya bystroy razrabotki opytnykh obraztsov tsifrovykh sistem upravleniya. Avtomatika i programmnaya inzheneriya. 2013. № 3 (5). S. 32-39.
[4] Zhmud' V.A., Fedorov D.S., Ivoylov A.Y., Trubin V.G. Razrabotka sistemy stabilizatsii ugla otkloneniya balansiruyushchego robota. Avtomatika i programmnaya inzheneriya. 2015. № 2 (12). S. 16-34.
[5] Yeskin A.V., Zhmud' V.A., Trubin V.G. Postroyeniye platformy modeliruyushchey rabotu robotizirovannykh sredstv na baze konstruktora LEGO MAYNDSTORMS 2.0 v chasti upravleniya elektrodvigatelyami. Avtomatika i programmnaya inzheneriya. 2013. № 1 (3). S. 88-94.
[6] Pechnikov A.L., Zhmud' V.A., Trubin V.G. Udalonnoye upravleniye robotom posredstvom xmpp-protokola. Sbornik nauchnykh trudov Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta. 2013.№ 3 (73). S. 85-92.
[7] Ivoylov A.Y., Yeskin A.V., Zhmud' V.A., Trubin V.G. Osobennosti raboty s ZHKI displeyem NOKIA 5110. Avtomatika i programmnaya inzheneriya. 2013. № 4 (6). S. 8-13.
[8] Kolker A.B., Livenets D.A., Kosheleva A.I., Zhmud' V.A. issledovaniye variantov sozdaniya intellektual'nykh sistem robototekhniki na baze odnoplatnykh komp'yuterov i svobodnykh operatsionnykh sistem. Avtomatika i programmnaya inzheneriya. 2012. № 1 (1). S. 84-98.
[9] Zhmud' V.A., Trubin M.V., Trubin I.V. Proyektirovaniye sensornykh knopok na baze

mikroskhemy TTP-224. Avtomatika i programmnaya inzheneriya. 2015. № 1 (11). S. 70-74.
[10] Ivoylov A.YU., Zhmud' V.A., Trubin V.G. razrabotka sistemy avtomaticheskoy stabilizatsii v vertikal'nom polozhenii dvukhkolesnoy platformy. Avtomatika i programmnaya inzheneriya. 2014. № 2 (8). S. 15-21.
[11] Yeskin A.V., Zhmud' V.A., Trubin V.G., Pechnikov A.L. Ispol'zovaniye seti internet dlya interaktivnoy laboratornoy raboty s distantsionnym upravleniyem model'yu robota snegouborshchika. Avtomatika i programmnaya inzheneriya. 2014. № 1 (7). S. 95-103.
[12] Zhmud' V.A., Kamenskaya A.S., Kurbet'yev K.V., Trubin V.G. Graficheskiy OLED displey UG-2864ASGGG14: pervoye vklyucheniye. Avtomatika i programmnaya inzheneriya. 2016. № 1 (15). S. 29-37.
[13] Yeskin A.V., Zhmud' V.A., Trubin V.G. Plaginy Eclipse dlya uskoreniya razrabotki programm tsifrovykh sistem upravleniya. Avtomatika i programmnaya inzheneriya. 2013. № 4 (6). S. 24-34.
[14] Zhmud' V.A., Kuznetsov K.A., Kondrat'yev N.O., Trubin V.G., Dimitrov L.V. Ul'trazvukovoy datchik izmereniya rasstoyaniya HC-SR04. Avtomatika i programmnaya inzheneriya. 2017. № 4 (22). S. 18-26.
[15] Fedorov D.S., Ivoylov A.YU., Zhmud' V.A., Trubin V.G. Ispol'zovaniye akselerometra ADXL335 dlya opredeleniya ugla. otkloneniya ot vertikali. Avtomatika i programmnaya inzheneriya. 2014. № 2 (8). S. 68-72.
[16] Vasil'yev V.A., Voyevoda A.A., Zhmud' V.A., Khassuonekh V.A. Tsifrovyye regulyatory: tselevyye funktsii nastroyki, vybor metoda integrirovaniya, apparatnaya realizatsiya. Sbornik nauchnykh trudov Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta. 2006.№ 4 (46). S. 3-10.
[17] Zhmud' V.A., Frantsuzova G.A., Vostrikov A.S. Dinamika mekhatronnykh sistem. Uchebnoye posobiye. Novosibirsk, 2014.
[18] Zhmud V., Yadrishnikov O., Poloshchuk A., Zavorin A. Modern key technologies in automatics: structures and numerical optimization of regulators. V sbornike: Proceedings - 2012 7th International Forum on Strategic Technology, IFOST 2012 2012. S. 6357804.
[19] Voyevoda A.A., Zhmud' V.A. Astaticheskoye upravleniye ob"yektami s nestatsionarnymi matrichnymi peredatochnymi funktsiyami metodom priblizhennogo obrashcheniya funktsional'nykh kompleksnykh matrits. Nauchnyy vestnik Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta. 2006.№ 2 (23). S. 3-8.
[20] Zhmud' V.A., Trubin M.V., Trubin I.V. Obmen dannymi mezhdu komp'yuterom i mikrokontrollerom STM32F100 po posledovatel'nomu interfeysu svyazi RS-232. Avtomatika i programmnaya inzheneriya. 2015. № 1 (11). S. 45-51.
[21] Yeskin A.V., Zhmud' V.A., Trubin V.G. Besprovodnoy udlinitel' posledovatel'nogo porta na baze radiokanala Bluetooth. Avtomatika i programmnaya inzheneriya. 2013. № 2 (4). S. 42-47.
[22] Zhmud' V.A., Trubin V.G., Yeskin A.V. Realizatsiya distantsionnogo upravleniya po radiokanalu Bluetooth platformoy, modeliruyushchey rabotu robotizirovannykh sredstv. Avtomatika i programmnaya inzheneriya. 2013. № 1 (3). S. 82-87.
[23] Wikipedia - Ethernet. URL: https://ru.wikipedia.org/wiki/Ethernet
[24] Wikipedia - Сокет. URL: https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BA%D0%B5%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81)
[25] Wikipedia – TCP. URL: https://ru.wikipedia.org/wiki/Transmission_Control_Protocol
[26] Wikipedia – UDP. URL: https://ru.wikipedia.org/wiki/UDP
[27] Официальный сайт Python.org. URL: https://www.python.org/
[28] Wiznet W5500 datasheet. URL: https://wizwiki.net/wiki/lib/exe/fetch.php?media=products:w5500_ds_v100e.pdf
[29] AMS1117 datasheet. URL: http://dalincom.ru/datasheet/AMS1117.pdf
[30] STM32F103 datasheet. URL: https://www.st.com/resource/en/datasheet/CD00161566.pdf
[31] Github – ioLibrary Driver for Wiznet W5XXX. URL: https://github.com/Wiznet/ioLibrary_Driver

Vadim Zhmud – Head of the Department of Automation in NSTU, Professor, Doctor of Technical Sciences.
E-mail: oao_nips@bk.ru

630073, Novosibirsk,
str. Prosp. K. Marksa, h. 20

Alexander Nezvanov - a student of the 1st year of the master's program at the Automation Department of the

NSTUE-mail: nezvanovml@mail.ru

Novosibirsk, 630073, str. Prosp. K. Marksa, h. 20, NSTU

Vitaly Trubin - Senior Lecturer, Department of Automatics, NSTU, Director of KB Automatics LLC.

E-mail: trubin@ngs.ru

Novosibirsk, 630073, str. Prosp. K. Marksa, h. 20, NSTU

Polina V. Mishchenko - Senior Lecturer, Department of Computing Engineering, NSTU.

E-mail: p.mishhenko@corp.nstu.ru

630073, Novosibirsk,
str. Prosp. K. Marksa, h. 20

The paper has been recieved 12.05.2019.