Сравнение скорости передачи по протоколам TCP/UDP с STM32F103 и Wiznet W5500

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

ФГБОУ ВО НГТУ, Новосибирск, Россия

Аннотация: В данном материале рассматривается задача обмена данными между микроконтроллером *STM32F103* и компьютером через *Ethernet* с помощью модуля *Wiznet W5500*. При работе с микроконтроллером довольно часто требуется наблюдать какие-либо данные в реальном времени. Это могут быть как данные с датчиков, так и значения переменных, которые необходимо отслеживать. В случае, когда необходимо передавать большой объём данных за короткое время, использование *UART* часто не обеспечивает требуемую скорость передачи данных. Одним из возможных решений данной проблемы является использование канала Ethernet. В материале рассматриваются возможные решения и измеряется скорость передачи данных. По итогу работы приведён программный код (для микроконтроллера на языке СИ, для компьютера - на языке *Python* 3) передачи данных с частотой 20 *кГц* для восьми 16-байтовых значений. Также приведены достигнутые скорости передачи данных при использовании сетевых протоколов *TCP* и *UDP*. Итоговая скорость передачи данных по *TCP* (*UDP*) превышает скорость *UART* в десятки раз. Также можно отметить, что использование W5500 является более простым, например, по сравнению с *USB*.

Ключевые слова: микроконтроллер, *UART*, *STM32F103*, *STM32F103C8T6*, *Wiznet W5500*, *TCP*, *UDP*, *Python*.

ВВЕДЕНИЕ

В процессе разработки электронных устройств зачастую требуется добавить возможность передавать на компьютер данные с высокой скоростью, например, для отладки действий контроллера в реальном времени. В данной статье сравнивается скорость передачи данных по каналу *Ethernet* с использованием *Wiznet W5500* (протоколы *TCP* и *UDP*), а также приводятся исходные коды программ для микроконтроллера и компьютера.

1 ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ ПЕРЕДАЧИ ДАННЫХ МЕЖДУ МИКРОКОНТРОЛЛЕРОМ И КОМПЬЮТЕРОМ

1.1 UART/RS-232

Универсальный асинхронный приёмопередатчик (англ. *Universal Asynchronous Receiver-Transmitter*, *UART*) [1] – это устройство, предназначенное для организации связи с другими цифровыми устройствами. Преобразует передаваемые данные в последовательный вид так, чтобы было возможно передать их по одной физической цифровой линии другому аналогичному устройству. Метод преобразования хорошо стандартизован и широко применяется в компьютерной технике. Представляет собой логическую схему, с одной стороны подключённую к шине вычислительного устройства, а с другой имеющую два (*TX* – «*transceiver*», передача и *RX* – «*receive*», приём) или более выводов для внешнего соединения.

Передача данных в *UART* осуществляется по одному биту через равные промежутки времени. Этот временной промежуток определяется заданной скоростью *UART* и для конкретного соединения указывается в бодах (что в данном случае соответствует битам в секунду). Существует общепринятый ряд стандартных скоростей: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 *бод*.

Помимо информационных бит, *UART* автоматически вставляет в поток стартовый и стоповый биты. При приёме эти служебные биты удаляются из потока. Обычно стартовый и стоповый биты обрамляют один байт информации (8 *бит*), при этом младший информационный бит передаётся первым, сразу после стартового.

Принято соглашение, что пассивным (в отсутствие потока данных) состоянием входа и выхода *UART* является логическая 1. Стартовый бит всегда логический 0, поэтому приёмник *UART* ждёт перепада из 1 в 0 и отсчитывает от него временной промежуток в половину длительности бита (середина передачи стартового бита). Если в этот момент на входе всё ещё 0, то запускается процесс приёма данных. Для этого приёмник отсчитывает 9 битовых длительностей подряд (для 8-битных данных), и в каждый момент фиксирует состояние входа. Первые 8 значений являются принятыми данными, последнее значение - проверочное (стоп-бит). Значение стоп-бита всегда равно 1. Если реально принятое значение иное, *UART* фиксирует ошибку.

Для формирования временных интервалов передающий и приёмный *UART* имеют источник точного времени (тактирования). Точность этого источника должна быть такой, чтобы сумма погрешностей (приёмника и передатчика) установки временного интервала от начала стартового импульса до середины стопового импульса не превышала половины (а лучше - четверти) битового интервала. Для 8-битной посылки это значение 0,5/9,5 ≈ 5 %. На практике, с учетом возможных искажений сигнала в линии, общая ошибка тактирования должна быть не более 3 %.

Поскольку синхронизирующие биты занимают часть битового потока, то результирующая пропускная способность *UART*

меньше скорости соединения. Например, для 8-битных посылок формата 8-N-1 синхронизирующие биты занимают 20 % потока, что при физической скорости линии 115 200 *бод* означает полезную скорость передачи данных 92 160 *бит/с* или 11 520 *байт/с*. Использование скоростей передачи выше 115 200 *бод* не рекомендуется, так как не гарантируется отсутствие негативного влияния внешних факторов на линию передачи, что может привести к искажению передаваемых данных.

Более подробно ознакомиться с работой *UART* можно в статье [2].

1.2 ETHERNET

Ethernet — 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.

Для использования технологии *Ethernet* в связке с микроконтроллером необходимо использовать сетевой контроллер. Одним из вариантов является *Wiznet W5500*, интерфейс подключения – *SPI*. Производитель сетевого контроллера гарантирует скорость обмена по *SPI* до 33,3 *МГц*.

Более подробно ознакомиться с основой работы технологии *Ethernet*, подключением *Wiznet W5500* к микроконтроллеру и основой передачи данных по сети на компьютер можно в статье [3]. Текущая статья является логическим развитием указанной, крайне рекомендуется ознакомиться с ней. Также в следующей главе будет использоваться исходный код предыдущей статьи.

2 ПЕРЕДАЧА ДАННЫХ ПО СЕТИ ETHERNET

В качестве задачи выберем считывание и передачу состояния цифрового входа микроконтроллера и передачу этих данных на сервер с частотой не менее 20 *кГц*. Эта частота объясняется просто – механические устройства, например, исполнительные механизмы робота, управляемые *ШИМ*, работают на частоте, близкой к 20 *кГц*. Меньшая частота не всегда приемлема, так как находится в слышимом человеком диапазоне и может вызывать акустический дискомфорт. А работа на частотах выше 20 *кГц* приводит к повышенным динамическим потерям при переключениях ключевых элементов. Как правило, необходимо иметь возможность передавать не одно значение, а несколько.

Реализация данной задачи производится путём модификации программного кода для *STM32F103* из пункта «3.2 Обмен данными между микроконтроллером и компьютером» статьи [3]. Рассмотрим два протокола передачи данных – *TCP* и *UDP*. Тестирование будем производить в локальной сети, в которой компьютер и *Wiznet W5500* подключены *LAN*-кабелем категории «*5e*» к одному коммутатору со скоростью портов 100 *Мбит/с*. При изменении данных условий может наблюдаться изменение достигнутых скоростей. Схема сети для тестирования представлена на *Рис*. 1.

Рис. 1. Схема сети для тестирования

2.1 ИСПОЛЬЗОВАНИЕ ПРОТОКОЛА TCP

Протокол *TCP* гарантирует доставку пакетов. Для передачи данных требуется предварительное установление соединения и обмен флагами контроля соединения.

Для запуска примера необходимо в исходном коде заменить функции `w5500_connect()` и `main()` на следующие:

```
int  w5500_connect(uint8_t  http_socket,  uint8_t
addr[4], uint32_t port) {
 uint8_t  code  =  socket(http_socket,  Sn_MR_TCP,
10888, 0);
 if (code != http_socket) return -1;
 code = connect(http_socket, addr, port);
 if  (code  !=  SOCK_OK)  {  close(http_socket);
return -1; }
 while (1){
     {
  uint8_t req[1024];
  for(uint16_t t = 0;t<512;t++){
   if(t%8 == 0){

IntToTwoShort(req+(t*2),GPIO_ReadInputDataBit(GPI
OC, IN1));
   } else {
   IntToTwoShort(req+(t*2),0);
```

```
   }
  }
  uint16_t len = sizeof(req);
  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;
   }
  }
 }
 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);
  TIM_TimeBaseStructure.TIM_Period = 50-1; // 1
миллисек.
  TIM_TimeBaseStructure.TIM_Prescaler = 1440 - 1;
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode =
TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
  TIM_ARRPreloadConfig(TIM2, ENABLE);
  NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;

NVIC_InitStructure.NVIC_IRQChannelPreemptionPrior
ity = 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 */
  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);
  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);
  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);
  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);
 w5500_ini();
 uint8_t destip[4] = {192,168,1,80};
 while (1) {
  w5500_connect(0,destip,destport);
  }
}
```

Также необходимо добавить функцию: `IntToTwoShort()`. Она необходима для преобразования 16-битного числа в два 8-битных. Исходный код функции:

```
void IntToTwoShort(uint8_t* res,uint16_t number){
 res[0] = 0;res[1] = 0;
 res[0] = number/256;
 res[1] = number%256;
}
```

Программный код сервера на языке *Python* приведён ниже:

Файл: server-tcp.py

```
# coding: cp1251
import socket
import time
def toFixed(numObj, digits=0):
 return f"{numObj:.{digits}f}"
numOfSec = 10
print ('start...')
# 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 10
sock.listen(10)
count = 0
countBytes = 0
startTime = time.time()
n = 0
f = open('output.txt','w')
f.write("sec,d1,d2,d3,d4,d5,d6,d7,d8")
while n<(numOfSec+1):
 # задание тайм-аута
 sock.settimeout(2)
 # конструкция try: ... except: для обработки
исключений
 try:
  # приём входящего подключения
  conn,addr = sock.accept()
  print("Новое соединение")
  while n<(numOfSec+1):
   temp = 0
   number = 0
   # получение данных от клиента
   data = conn.recv(2048)
   if not data:
    conn.close()
    break
```

```
   while temp < len(data):
    res = data[temp+0]*256+data[temp+1]
    if n < numOfSec:
     if (number) % 8 == 0:
      f.write("\r\n")
      f.write(str(n))
      f.write(",")
     else :
      f.write(",")
     f.write(str(res))
    temp+=2
    number+=1
   count+=1
   countBytes+=len(data)
   if (time.time() - startTime) >= 1.0 :
    n+=1
    temptime = time.time() - startTime
    print                                          ("
",round(countBytes*8/temptime),"bits/s,",round((c
ountBytes)/temptime),"bytes/s,",toFixed((countByt
es)/temptime/16/1000,1),"kHz (8x2b)\r\n")
    count = 0
    countBytes = 0
    startTime = time.time()
  # закрытие соединения
  print("Закрытие соединения")
  conn.close()
 except Exception as msg:
  print(msg)
# закрытие сокета
sock.close()
```

Порядок запуска не имеет значения, так как контроллер передаёт данные непрерывно и программу на сервере можно запустить в любой момент. Результат запуска представлен на *Рис*. 2.

```
3014483 bits/s, 376810 bytes/s, 23.6 kHz (8x2b)
3068755 bits/s, 383594 bytes/s, 24.0 kHz (8x2b)
3109673 bits/s, 388709 bytes/s, 24.3 kHz (8x2b)
3093305 bits/s, 386663 bytes/s, 24.2 kHz (8x2b)
3098394 bits/s, 387299 bytes/s, 24.2 kHz (8x2b)
3109672 bits/s, 388709 bytes/s, 24.3 kHz (8x2b)
3101489 bits/s, 387686 bytes/s, 24.2 kHz (8x2b)
3093305 bits/s, 386663 bytes/s, 24.2 kHz (8x2b)
3103471 bits/s, 387934 bytes/s, 24.2 kHz (8x2b)
3117857 bits/s, 389732 bytes/s, 24.4 kHz (8x2b)
```

Рис. 2. Результат запуска

Как можно заметить из результатов тестирования, передача происходит стабильно, скорость практически неизменна в ходе работы.

2.2 ИСПОЛЬЗОВАНИЕ ПРОТОКОЛА UDP

Протокол *UDP*, в отличие от *TCP*, не требует предварительного установления соединения, что значительно уменьшает количество служебной информации при передаче данных. Основным недостатком данного протокола является отсутствие контроля доставки данных, но эту функцию можно переложить на программу-обработчик. Решением этой проблемы было добавление порядковой нумерации пакетов, переданных с микроконтроллера на компьютер, таким образом, можно легко установить порядковый номер отдельного набора данных или же определить их потерю.

Для запуска примера необходимо в исходном коде заменить функции `w5500_connect()` и `main()` на следующие:

```
int   w5500_connect(uint8_t   http_socket,uint8_t
addr[4], uint16_t port){
 uint8_t req[1024];
 for(uint16_t t = 0;t<512;t++){
  if(t%8 == 1){

IntToTwoShort(req+(t*2),GPIO_ReadInputDataBit(GPI
OC, IN1));
  } else if (t%8 == 0){
   IntToTwoShort(req+(t*2),sendcount);
   sendcount++;
  } else {
   IntToTwoShort(req+(t*2),0);
  }
 }
 uint16_t len = sizeof(req);
 uint8_t* buff = (uint8_t*)&req;
 while(len > 0) {
  int32_t  nbytes  =  sendto(http_socket,  buff,
len,addr,port);  len -= nbytes;
 }
    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);
  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);
 NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;

NVIC_InitStructure.NVIC_IRQChannelPreemptionPrior
ity = 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);
  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);
```

```
  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);
  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);
  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);
 w5500_ini();
 sock_open();
 uint8_t destip[4] = {192,168,1,80};
 while (1) {
  w5500_connect(0,destip,9999);
 }
}
```

Также необходимо добавить две функции: `IntToTwoShort()` (необходима для преобразования 16-битного числа в два 8-битных), `sock_open()` (задаёт режим работы сокетов как *UDP*).

Исходный код функций:

```
void IntToTwoShort(uint8_t* res,uint16_t number){
 res[0] = 0;res[1] = 0;
 res[0] = number/256;
 res[1] = number%256;
}
uint8_t sock_open(){
 for(uint8_t a = 0; a <8; a++){
  uint8_t  code  =  socket(a,  Sn_MR_UDP,  10888+a,
0);
 }
}
```

Программный код сервера на языке *Python* приведён ниже:

Файл: server-udp.py

```
import socket
import time
def toFixed(numObj, digits=0):
 return f"{numObj:.{digits}f}"
numOfSec = 10
print ('start...')
sock                                            =
socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
sock.bind(('', 9999))
count = 0
countBytes = 0
startTime = time.time()
n = 0
f = open('output.txt','w')
f.write("sec,number,d1,d2,d3,d4,d5,d6,d7")
while n<(numOfSec+1):
 data,address = sock.recvfrom(4096)
 temp = 0
 number = 0
 #f.write(str(n))
 #f.write(" ")
 while temp < len(data):
  res = data[temp+0]*256+data[temp+1]
  if n < numOfSec:
   if (number) % 8 == 0:
    f.write("\r\n")
    f.write(str(n))
    f.write(",")
   else :
    f.write(",")
   f.write(str(res))
  temp += 2
  number += 1
 count += 1
 countBytes += len(data)
 if (time.time() - startTime) >= 1.0 :
  n+=1
  temptime = time.time() - startTime
  print
(round(countBytes*8/temptime),"bits/s,",round((co
untBytes)/temptime),"bytes/s,",toFixed((countByte
s/8*7)/temptime/16/1000,1),"kHz (7x2b)\r\n")
  count = 0
  countBytes = 0
  startTime = time.time()
f.close()
sock.close()
```

Порядок запуска не имеет значения, так как контроллер передаёт данные непрерывно и программу на сервере можно запустить в любой момент. Результат запуска представлен на *рис. 3.*

```
3019655 bits/s, 377457 bytes/s, 20.6 kHz (7x2b)
3024816 bits/s, 378102 bytes/s, 20.7 kHz (7x2b)
3024818 bits/s, 378102 bytes/s, 20.7 kHz (7x2b)
3027838 bits/s, 378480 bytes/s, 20.7 kHz (7x2b)
3014484 bits/s, 376811 bytes/s, 20.6 kHz (7x2b)
3016643 bits/s, 377080 bytes/s, 20.6 kHz (7x2b)
3036022 bits/s, 379503 bytes/s, 20.8 kHz (7x2b)
3032992 bits/s, 379124 bytes/s, 20.7 kHz (7x2b)
3024817 bits/s, 378102 bytes/s, 20.7 kHz (7x2b)
3024817 bits/s, 378102 bytes/s, 20.7 kHz (7x2b)
3027839 bits/s, 378480 bytes/s, 20.7 kHz (7x2b)
```

Рис. 3. Результат запуска

Как можно заметить из результатов тестирования, передача происходит стабильно, скорость практически неизменна в ходе работы. Отдельно необходимо отметить, что в представленном скриншоте счётчик битов и байтов считает размер всего сообщения (с учётом служебного числа, необходимого для контроля), а итоговая скорость 20.7 *кГц* приводится для полезных данных, то есть для семи 2-байтовых чисел.

2.3 ВИЗУАЛИЗАЦИЯ ПОЛУЧЕННЫХ ДАННЫХ

Для построения графика полученных значений используем библиотеки *Pandas* (для работы с *csv*-файлом) и *Matplotlib* (для построения графиков). Для построения используется первый столбец из 8 доступных с полезными данными.

Файл: chart.py

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
matplotlib.style.use('ggplot')
url = 'output.txt'
df = pd.read_csv(url, sep=',')
df1 = df[['d1']]
df1.plot()
plt.show()
```

За десять секунд сбора данных (с использованием протокола *TCP*) количество выборок в файле достигло 220 тысяч строк. На *Рис.* 4 представлен общий вид графика за 10 секунд. Чёрная стрелка указывает на фрагмент графика, показанный крупным планом на *Рис.* 5.

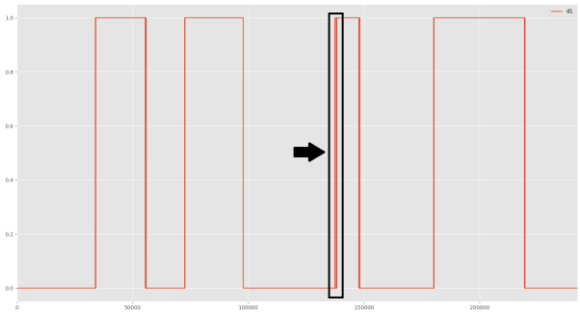

Рис. 4. Общий вид графика, временной промежуток 10 *с*

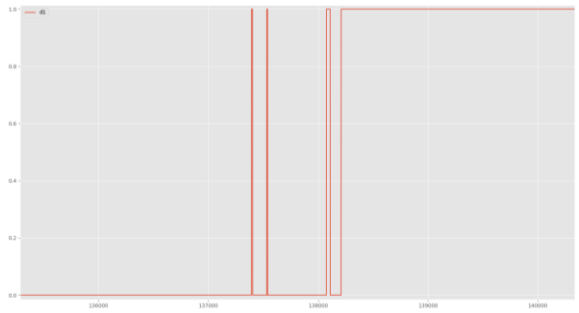

Рис. 5. Запечатлённый дребезг контактов кнопки в крупном масштабе (временной отрезок приблизительно в 20 мс)

Такой метод визуализации является крайне удобным, ввиду своей простоты и возможности комбинирования нескольких потоков (в данном примере столбцов) данных на одной временной шкале, что позволяет отслеживать зависимость одного значения от другого в процессе работы устройства.

3 СРАВНЕНИЕ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

По итогу тестирования сетевого контроллера были получены данные, приведённые в *Таблице* 1.

Таблица 1.

Скорости передачи данных на компьютер с помощью разных интерфейсов

Метод передачи	Скорость, *бит/сек*	Фактическая скорость, *бит/сек*
UART	115 200	92 160
W5500 (TCP)	3 100 000	3 100 000
W5500 (UDP)	3 027 000	2 648 625

Как видно из таблицы, достигнутая фактическая скорость передачи данных на компьютер при использовании сетевого контроллера *Wiznet W5500* примерно в 33 раза выше, чем при использовании стандарта *UART*.

Даже при использовании теоретически возможной скорости *UART* в 921 600 *бит/сек*, достигнутая скорость при использовании сетевого контроллера больше в 3,3 раза. Подключение сетевого контроллера несколько сложнее, чем использование *UART*, но его преимущества очевидны: скорость в разы выше и возможность расположения компьютера на любом расстоянии от контроллера (до 100 *м*).

Данная разработка направлена на дальнейшее развитие лаборатории микропроцессорной техники при кафедре Автоматики НГТУ с целью объединения и ее результатов с ранее полученными результатами в области робототехники [4–8]. Это позволяет улучшать образовательный процесс. Описанные результаты уже в текущем учебном году будут использоваться в образовательном процессе.

Работа выполнена в рамках Темплана НИР кафедры Автоматики.

ВЫВОДЫ

1. *Wiznet W5500* является недорогим и доступным средством для организации быстрого канала связи микроконтроллер-компьютер.
2. С помощью *Wiznet W5500* можно достичь скорости передачи между компьютером и микроконтроллером в десятки раз больше, чем с помощью *UART*.
3. Использование протокола *TCP* более рационально, ввиду равной с протоколом *UDP* скоростью передачи, и наличием встроенного в протокол контроля доставки данных.

ЛИТЕРАТУРА

[1] Wikipedia - Универсальный асинхронный приёмопередатчик. URL: https://ru.wikipedia.org/wiki/%D0%A3%D0%BD%D0%B8%D0%B2%D0%B5%D1%80%D1%81%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B0%D1%81%D0%B8%D0%BD%D1%85%D1%

80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BF%D1%80%D0%B8%D1%91%D0%BC%D0%BE%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%82%D1%87%D0%B8%D0%BA
[2] Жмудь В.А., Трубин И.В., Трубин М.В. Обмен данными между компьютером и микроконтроллером STM32F100 по последовательному интерфейсу связи RS-232. Автоматика и программная инженерия. 2015. № 1 (11). С. 45-51. URL: http://kb-au.ru/wp-content/uploads/AaSI-2015-1-6.pdf
[3] Незванов А.И., Трубин В.Г., Мищенко П.В., Жмудь В.А. Обмен данными между STM32F103 и компьютером с помощью Wiznet W5500. Автоматика и программная инженерия. 2019. № 2 (28). С. 24-35. URL: http://jurnal.nips.ru/sites/default/files/AaSI-2-2019-3_0.pdf
[4] Жмудь В.А., Французова Г.А., Востриков А.С. Динамика мехатронных систем. Учебное пособие. Новосибирск, 2014.
[5] Zhmud V., Vostrikov A., Semibalamut V. Feedback systems with pseudo local loops. В сборнике: Testing and Measurement: Techniques and Applications - Proceedings of the 2015 International Conference on Testing and Measurement: Techniques and Applications, TMTA 2015 2015. С. 411-417.
[6] Жмудь В.А., Ядрышников О. Численная оптимизация пид-регуляторов с использованием детектора правильности движения в целевой функции. Автоматика и программная инженерия. 2013. № 1 (3). С. 24-29.
[7] 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.
[8] Федоров Д.С., Ивойлов А.Ю., Жмудь В.А., Трубин В.Г. Использование измерительной системы mpu 6050 для определения угловых скоростей и линейных ускорений. Автоматика и программная инженерия. 2015. № 1 (11). С. 75-80.

Вадим Аркадьевич Жмудь – заведующий кафедрой Автоматики НГТУ, д.т.н. доцент.
E-mail: zhmud@corp.nstu.ru
Новосибирск, просп. К. Маркса, 20

Александр Игоревич Незванов – студент 2 курса магистратуры кафедры Автоматики НГТУ.
E-mail: nezvanovml@mail.ru
Новосибирск, просп. К. Маркса, 20

Виталий Геннадьевич Трубин – старший преподаватель кафедры Автоматики НГТУ, директор ООО «КБ Автоматика».
E-mail: trubin@ngs.ru
Новосибирск, просп. К. Маркса, 20

Статья получена 12.10.2019 г.

Compare TCP/UDP Rates Using STM32F103 and Wiznet W5500

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

Novosibirsk State Technical University

Abstract: This paper discusses the task of exchanging data between the STM32F103 microcontroller and a computer via Ethernet using the Wiznet W5500 module. When working with a microcontroller, it is often necessary to observe any data in real time. This can be either data from sensors or values of variables that need to be monitored. In the case when it is necessary to transfer a large amount of data in a short time, the use of UART often does not provide the required data rate. One possible solution to this problem is to use an Ethernet channel. The material discusses possible solutions and measures the data transfer rate. As a result of the work, the program code (for the microcontroller in the SI language, for the computer in the Python 3 language) of data transmission with a frequency of 20 kHz for eight 16-byte values is given. The data transfer rates achieved using the TCP and UDP network protocols are also shown. The total TCP data rate (UDP) is tens of times greater than the UART speed. It can also be noted that using the W5500 is simpler, for example, compared to USB.

Key words: microcontroller, UART, STM32F103, STM32F103C8T6, Wiznet W5500, TCP, UDP, Python.

REFERENCES

[1] Wikipedia - Universal Asynchronous Transceiver. URL: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter
[2] Zhmud V.A., Trubin I.V., Trubin M.V. Data exchange between the computer and the STM32F100 microcontroller via the serial RS-232 communication interface. Automation and software engineering. 2015. No. 1 (11). S. 45-51. URL: http://kb-au.ru/wp-content/uploads/AaSI-2015-1-6.pdf
[3] Nezvanov A.I., Trubin V.G., Mishchenko P.V., Zhmud V.A. Communication between the STM32F103 and a computer using the Wiznet W5500. Automation and software engineering. 2019.No 2 (28). S. 24-35. URL: http://jurnal.nips.ru/sites/default/files/AaSI-2-2019-3_0.pdf
[4] Zhmud V.A., Frantsuzova G.A., Vostrikov A.S. Dinamika mekhatronnykh sistem. Uchebnoye posobiye. Novosibirsk, 2014.
[5] Zhmud V., Vostrikov A., Semibalamut V. Feedback systems with pseudo local loops. V sbornike: Testing and Measurement: Techniques and Applications - Proceedings of the 2015 International Conference on Testing and Measurement: Techniques and Applications, TMTA 2015 2015. S. 411-417.

[6] Zhmud V.A., Yadryshnikov O. Chislennaya optimizatsiya pid-regulyatorov s ispol'zovaniyem detektora pravil'nosti dvizheniya v tselevoy funktsii. Avtomatika i programmnaya inzheneriya. 2013. № 1 (3). S. 24-29.
[7] Zhmud V.A., Fedorov D.S., Ivoylov A.YU., Trubin V.G. Razrabotka sistemy stabilizatsii ugla otkloneniya balansiruyushchego robota. Avtomatika i programmnaya inzheneriya. 2015. № 2 (12). S. 16-34.
[8] Fedorov D.S., Ivoylov A.YU., Zhmud V.A., Trubin V.G. Ispol'zovaniye izmeritel'noy sistemy mpu 6050 dlya opredeleniya uglovykh skorostey i lineynykh uskoreniy. Avtomatika i programmnaya inzheneriya. 2015. № 1 (11). S. 75-80.

Vadim Arkadevich Zhmud - Head of the Department of Automation NSTU, Doctor of Technical Sciences professor.
E-mail: zhmud@corp.nstu.ru
Novosibirsk, Prosp. K. Marksa, 20

Alexander Igorevich Nezvanov is a 2nd year student at the Department of Automation at NSTU.
E-mail: nezvanovml@mail.ru
Novosibirsk, Prosp. K. Marksa, 20

Vitaliy Gennadievich Trubin - Senior Lecturer, Department of Automation, NSTU, Director of “KB Avtomatika” LLC.
E-mail: trubin@ngs.ru
Novosibirsk, Prosp. K. Marksa, 20

The paper has been received on 12/10/ 2019.