Применение микроконтроллера AVR32UC3. Модуль Ethernet MAC

Вячеслав Бородулин, Александр Шитиков (г. Хабаровск)

Микроконтроллерам семейства AVR32 AP7 и UC3 посвящено множество статей, но большинство из них носят обзорный характер. Данной публикацией авторы открывают цикл статей, посвящённых практическому применению микроконтроллеров AVR32UC3. В работе приводятся результаты экспериментов с различными реализациями стека протоколов TCP/IP для микроконтроллера UC3. Также рассмотрены варианты увеличения пропускной способности сетевого интерфейса.

В состав микроконтроллера AT32UC3A входит модуль MAC-уровня Ethernet 10/100 Мбит/с [1]. Блок-схема модуля приведена на рисунке 1. Для реализации нижних уровней стека TCP/IP на микроконтроллере AVR32UC дополнительно необходимо использовать микросхему приёмопередатчика физического уровня. Например, на отладочном стенде ATMEL EVK1100 (см. рис. 2) используется микросхема DP83848I в BGA-корпусе [2,3].

Для тестирования производительности контроллера MAC Ethernet было разработано две программы – для микроконтроллера и для ПК. Исследования проводились на отладочной плате ATMEL EVK1100 с микроконтроллером AVR32 UC3A0512. Это – один из самых мощных микроконтроллеров серии UC3 (включает 512 Кб флэш-памяти).

За основу проекта на стороне микроконтроллера был взят проект *EVK1100-SERVICES-LWIP* (поставляется вместе с *AVR32 studio*) и реализация эхо-сервера, взятого с интернет-страницы [3], где представлены три варианта исходных текстов. Первый построен на использовании RAW API-функций – самый нижний уровень программирования *LWIP*. Вторая реализация использует функцию *netconn* – надстройку над RAW-функциями. Третий вариант эхо-сервера использует BSD-сокеты и является надстройкой над функциями *netconn*.

Измерения скорости передачи данных проводились на двух реализациях из трёх, представленных на интернет-странице: на первой (*Socket programming*) и последней (*RAW API programming*). В обоих случаях микроконтроллер являлся сервером. Сначала была измерена скорость передачи данных с использованием третьего варианта как самого простого, но потенциально более медленного. Для этого в программе эхо-сервера были внесены изменения в файл *BasicTFTP.c*.

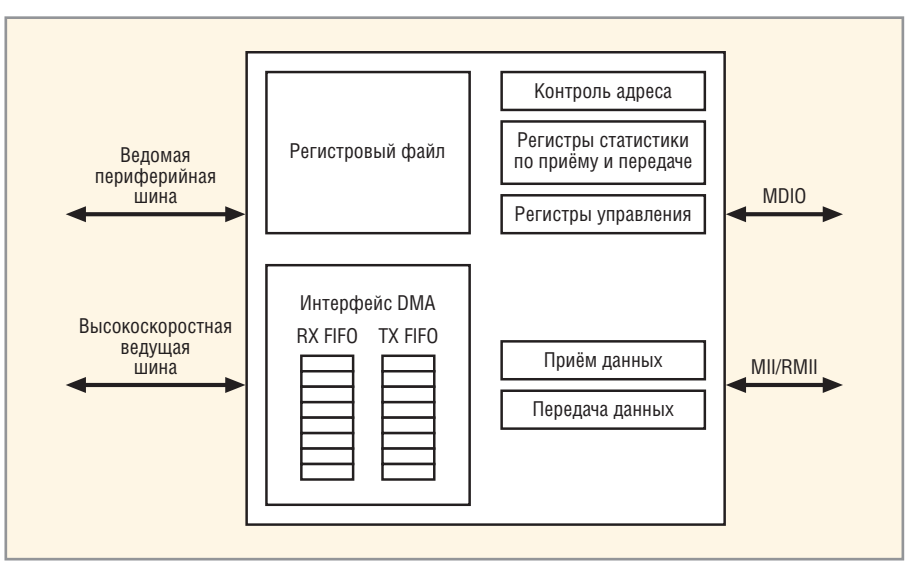

Рис. 1. Блок-схема модуля Ethernet MAC

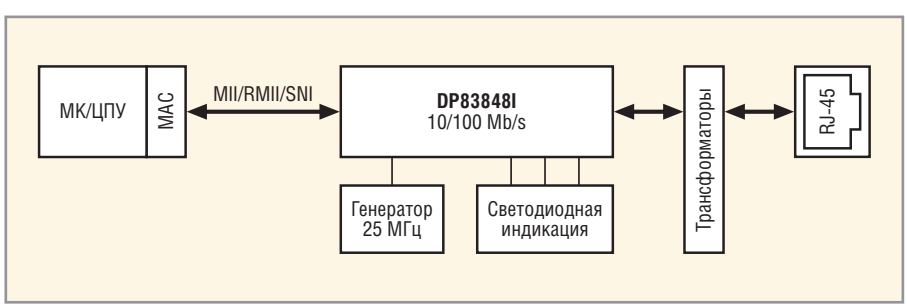

Рис. 2. Блок-схема соединения МК и приёмопередатчика Ethernet

Листинг 1. Фрагмент кода файла BasicTFTP.c

```
#define SIZE1 102400   //объём посылаемых данных в байтах
char buffer[1024];
//данные, которые мы посылаем по интерфейсу
int nn=0;
for (nn=0;nn<1024;nn++)
//инициализация массива
        {
        buffer[nn]='Q';
        }
…
if (clientfd>0){
do{
nbytes=lwip_send(clientfd,
buffer, sizeof(buffer), 0);
nnbytes=nbytes+nnbytes;
} while (nnbytes<SIZE1);
```

Чтобы поменять направление передачи данных, следует изменить тело бесконечного цикла (*while (1)*) на следующий код:

Листинг 2. Фрагмент кода файла BasicTFTP.c

```
{ int clientfd;
struct sockaddr_in client_addr;
int addrlen=sizeof(client_addr);
char buffer1[]="end";
int nbytes=0;
```

```
int nnbytes=0;
clientfd = lwip_accept(lSocket,
(struct sockaddr*)&client_addr,
(socklen_t)&addrlen);
if (clientfd>0){
do{
nbytes=lwip_recv(clientfd,
buffer, sizeof(buffer),0);
if (nbytes>0)
nnbytes=nbytes+nnbytes;
if (nnbytes>102400)
lwip_send(clientfd, buffer1, 3, 0);
} while (nnbytes<102400);
lwip_close(clientfd);
}
```

Для реализации программы на стороне микроконтроллера, позволяющей измерить скорость передачи данных первым способом (RAW API), код эхо-сервера был изменён в соответствии с листингом 3, который размещён на сайте журнала.

Данная программа принимает данные с ПК, т.е. осуществляется передача данных с ПК на микроконтроллер. Ниже представлен листинг программы для микроконтроллера, которая передаёт данные на ПК.

Листинг 4.

```
#define SIZE1 1024000
char mydata[1024];
char end[]="end";
int count=0;
static void close_conn(struct
tcp_pcb *pcb){
tcp_arg(pcb, NULL);
tcp_sent(pcb, NULL);
tcp_recv(pcb, NULL);
tcp_close(pcb);
}
static err_t f_send(void *arg,
struct tcp_pcb *pcb, err_t err){
        tcp_write(pcb, mydata,
sizeof(mydata), 0);//передача
массива.
        return ERR_OK;
}
static err_t echo_accept(void
*arg, struct tcp_pcb *pcb, err_t
err){
LWIP_UNUSED_ARG(arg);
LWIP_UNUSED_ARG(err);
tcp_setprio(pcb, 1);
tcp_sent(pcb, f_send);
tcp_err(pcb, NULL); //Don't care
about error here
tcp_poll(pcb, NULL, 4); //No
polling here
return ERR_OK;
}
portTASK_FUNCTION( vBa-
sicTFTPServer, pvParameters ){
struct tcp_pcb *ptel_pcb;
ptel_pcb = tcp_new();
tcp_bind(ptel_pcb, IP_ADDR_ANY, 23);
int i;
for (i=0;i<1024;i++){
        mydata[i]='f';
}
while (1){
        count=0;
        ptel_pcb =
tcp_listen(ptel_pcb);
        tcp_accept(ptel_pcb,
echo_accept);
}
}
```

Рассмотрим некоторые аспекты создания сервера под управлением стека *LWIP*. Для начала необходимо инициализировать структуру *tcp_pcb (protocol control block)*, используя функцию *tcp_new()*. В этой структуре содержится вся информация о подключении. При успешном создании *PCB* следует

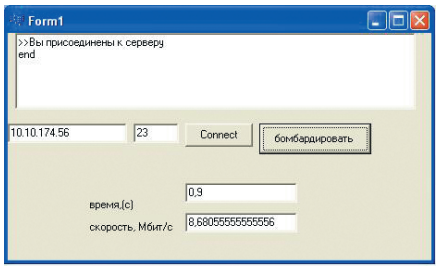

Рис. 3. Клиент на стороне ПК

связать порт с IP-адресом при помощи функции *tcp_bind()*. Если порт уже используется, то функция вернёт значение *ERR_USE*.

Далее необходимо дать серверу команду ожидания входящего подключения с помощью функции *tcp_listen()*. Последним этапом установки соединения является вызов функции извлечения запросов на соединение *tcp_accept ()*. Одним из аргументов этой функции является функция обратного вызова, которая будет обрабатывать принятое соединение. В нашей программе этой функцией является *echo_accept()*. В ней вы должны определить, какие функции будут вызываться при определённом событии. Например, чтобы получение данных обрабатывалось написанной нами функцией *echo_recv()*, в теле подпрограммы *echo_accept()* должна вызывается функция *tcp_recv(pcb, echo_recv)*, где первый параметр – указатель на структуру PCB, второй – функция обработки события.

Помимо функции *tcp_recv()*, мы видим в тексте программы следующие функции: *tcp_sent()* – для отправки данных, *tcp_err()* – обработка ошибок, *tcp_poll()* – функция последовательного опроса, когда соединение простаивает, т.е. никакие данные не передаются; *LWIP* будет неоднократно вызывать указанную в ней функцию. Её можно использовать как сторожевой таймер, например, чтобы закрывать подключения, которые долго простаивают. Последний параметр этой функции – интервал времени, через который будет вызываться эта функция; если этот параметр равняется двум, то функция будет вызываться каждую секунду.

Для измерения скорости в среде *Borland Builder C++ 6.0* были разработаны два приложения, работающие под ОС Windows . Одно из них измеряет скорость передачи данных с ПК на микроконтроллер, второе – в обратном направлении. В листинге 5 представлен код программы, которая передаёт на стенд *SIZE1* байт.

Программа работает следующим образом. Приложение на стороне ПК запускает таймер и начинает пересылать данные на стенд. Как только микроконтроллер принял контрольный объём информации (*SIZE* байт), он посылает на ПК сообщение, что данные приняты. Программа на стороне ПК фиксирует время передачи и вычисляет скорость. Следует отметить, что контрольный объём данных (параметр *SIZE*) следует согласовывать между программой на МК и программой на ПК.

Листинг 5. Фрагмент кода программы, передающей данные на стенд

```
…
#define SIZE1 1024000 //контрольный размер данных
…
void __fastcall
TForm1::ClientSocket1Read(TObject
*Sender,
TCustomWinSocket *Socket)
{
AnsiString Rtext ; //подтверждение сервером принятия данных
Form1 -> Timer1 -> Enabled = false;//
Rtext = ClientSocket1->Socket-
>ReceiveText() ;
speed = (SIZE1*8)/(1024*1024*TotalTime);
Edit4 -> Text = speed;
//TotalTime = 0.0;
Memo1->Lines->Add(Rtext);
}
//------------------------------
//обработчик кнопки начала отправки данных
void __fastcall
TForm1::Button3Click(TObject
*Sender)
{
char* array = new char [SIZE1];

for (int i=0; i< SIZE1;i++)
        {
        *(array +i)= 'Q';
        }
int nn= SIZE1+1;
TotalTime = 0;
Timer1 -> Enabled = true;//Старт!
ClientSocket1->Socket-
>SendBuf(array,nn);
}
```

Рабочее окно программы представлено на рисунке 3.

Программа, принимающая данные (см. листинг 6), работает до тех пор, пока объём принятых данных не будет равен контрольному размеру *SIZE1*, после этого соединение закрывается и подсчитывается скорость передачи.

Листинг 6. (фрагмент кода программы, принимающей данные со стенда)

Таблица 1. Результаты измерения скорости передачи данных при исходном ПО

Способ	Направление передачи данных	Скорость, Мбит/с
Socket programming	Микроконтроллер принимает данные, см. листинг 2	0,05
	Микроконтроллер передаёт данные, см. листинг 1	0,055
RAW API programming	Микроконтроллер принимает данные, см. листинг 3	8,6
	Микроконтроллер передаёт данные, см. листинг 4	0,05

Таблица 2. Результаты измерения скорости передачи данных после модернизации ПО

Способ	Направление передачи данных	Скорость, Мбит/с
Socket programming	Микроконтроллер принимает данные	0,122
	Микроконтроллер передаёт данные	6,3
RAW API programming	Микроконтроллер принимает данные	8,7
	Микроконтроллер передаёт данные	11,7

```
void __fastcall TForm1::Client-
Socket1Read(TObject *Sender,
TCustomWinSocket *Socket)
{
        AnsiString Rtext ;
//текст, который посылает сервер
        Rtext = ClientSocket1-
>Socket->ReceiveText();
        if (Rtext.Length()>1)
        {
        sz=sz+Rtext.Length();
        }
        Memo1->Lines-
>Add((sz));//.Length());
        if (sz>=SIZE1)
        {
ClientSocket1-> Active = false;
        Form1 -> Timer1 -> Enabled
= false;
        speed =
(sz*8)/(1000000*TotalTime);
        Edit4 -> Text = speed;
}
}
```

В таблице 1 приведены результаты измерения скорости передачи данных.

В заголовочном файле *lwipports.h* можно изменять размеры различных буферов протокола TCP (например, *TCP_MSS* – максимальный размер сегмента TCP), включать или выключать различные части стека. С целью увеличения скоростных показателей в программе были произведены следующие изменения:

- увеличены размеры буферов TCP_WND (размер окна приёма) и TCP_SND_BUF (размер буфера приёма TCP):
  #define TCP_WND 2000
  #define TCP_SND_BUF 2150*4
  (начальные значения были 1500 и 2150 соответственно);
- увеличен размер массива mydata (см. листинги 3 и 4) в четыре раза, т.е. mydata[4096].

Результаты измерения скорости передачи после указанных изменений представлены в таблице 2.

Заключение

Первые результаты измерения скорости не были впечатляющими. Но, как оказалось, чтобы увеличить скорость, необходимо лишь настроить размеры буферов протокола TCP. Чтобы представлять, как тот или иной буфер влияет на скорость передачи данных, следует обратиться к документации этого протокола. Возможно, при более тонкой настройке параметров протокола как на принимающей, так и на передающей стороне удастся ещё больше увеличить пропускную способность интерфейса МК. В ходе экспериментов авторам удалось повысить скорость загрузки данных (от сервера) до 11,7 Мбит/с, а скорость выгрузки – до 8,7 Мбит/с.

Литература

1. Документация на микроконтроллер AVR32UC3A, http://www.atmel.com/dyn/resources/prod_documents/doc32058.pdf.
2. Принципиальная электрическая схема стенда ATMEL EVK1100, http://www.atmel.com/dyn/resources/prod_documents/EVK1100_SCHEMATICS_REVC.pdf.
3. Перечень элементов стенда ATMEL EVK1100, http://www.atmel.com/dyn/resources/prod_documents/EVK1100_%20BILL%20OF%20MATERIALS_RevC.xls.
4. lwIP TCP Example: How to write a TCP echo server (telnet), http://www.ultimaserial.com/avr_lwip_tcp.html.