**Lesson 82**

**LAN. ENC28J60. Remote access**

So far we have been using our **ENC28J60** module  could connect, receive and transmit data only from nodes that are on the local network. I think that the moment has come when we should try using it "to communicate" with the network of the outside network - with the Internet network. Also, the external network can be not only the Internet, but also some other network. The most important thing is that we will already access nodes outside our router. I think everyone knows how the routing and interrogation of nodes that are outside the network is going on, well, or can find out .. This information is very plentiful on the vastness of the global network. But how to connect with them using our microcircuit and how to organize it in the code - not everyone knows, Although the information is still there, but everywhere is different. I want to give my own answer to this question.

Nevertheless, I will tell you a little about how the external network is polled. For example, we assigned our module IP address 192.168.1.197, and it is connected to the network with the mask 255.255.255.0, at the router address 192.168.1.1. In this case, if we turn to the device in our network with the address from 192.168.1.1 to the address 192.168.1.254, it will be considered that we have turned to the local network device and we have the full right to know its MAC address, which we know we can by sending an ARP request. And if the device has an IP address different from the above (for example, 8.143.111.23), then it will be considered that we have already turned to the external network device and the router will have to learn the route to this device and provide us with a certain access to it. Only the MAC address of an external device the router will not be required to report to us, so at the channel level we will work with the MAC address of the router. Therefore, we need to organize our code so that this code first finds out that the device is outside our network and after this clarification returned us the MAC address of the router, or as it is called - gateway or gateway, in short, like so.

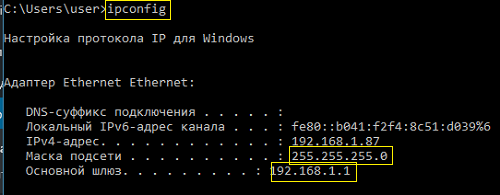
The ultimate goal of our today's lesson is to send an ICMP request or PING to an external node and wait for it to respond, thereby confirming that we can communicate normally with this node on different network layers.

So let's get down to business from the theory right away.

Let's create a project named  **ENC28J60\_REMOTE** , reworking it from the project of the  [**last lesson**](http://narodstream.ru/stm-urok-81-lan-enc28j60-udp-client/)  with the name  **ENC28J60\_UDPC** .

Run the project in the **Cube MX** generator and, without touching anything, generate a project for the Keil programming environment, open it, make the programmer settings for auto-reload, and connect our libraries to the project tree.

The first thing in the **net.h** file **is to** add two macro substitutions for the subnet mask and the address of the router (gateway). At you they can differ from mine. I think you know your network settings, if you do not know, then you can see them in the properties of your network adapters. Only this will happen if you use static addressing, in the case of using dynamic addressing (DHCP), you can find out all this by typing the  **ipconfig** command on the command line



After reading your parameters, add macros to the above location

#include "enc28j60.h"

//--------------------------------------------------

#define IP\_ADDR {192,168,1,197}

**#define IP\_GATE {192,168,1,1}**

**#define IP\_MASK {255,255,255,0}**

Also add global arrays for our network addresses in the **net.c** file

uint8\_t ipaddr[4]=IP\_ADDR;

**uint8\_t ipgate[4]=IP\_GATE;**

**uint8\_t ipmask[4]=IP\_MASK;**

Then we connect these arrays in the file **arp.c**

extern uint8\_t ipaddr[4];

**extern uint8\_t ipgate[4];**

**extern uint8\_t ipmask[4];**

Further, when we try to send an ARP request to a network node, we must determine to which network this node belongs - to local or external (remote). Therefore, go to the arp.c file and in the **arp\_request** function  **we will** first add a local array to store the IP address by which we will determine the MAC address using the ARP request, as well as an integer variable that will accumulate the results of logical operations on addresses and mask

uint8\_t arp\_request(uint8\_t \*ip\_addr)

{

  uint8\_t i, j;

**uint8\_t ip[4];**

**uint8\_t iptemp = 0;**

Then, with the help of a simple cycle, we define the membership of the local network

uint8\_t iptemp = 0;

**for(i=0;i<4;i++)**

**{**

**iptemp += (ip\_addr[i] ^ ipaddr[i]) & ipmask[i];**

**}**

And then we add the following condition

enc28j60\_frame\_ptr \*frame=(void\*)net\_buf;

**//проверим принадлежность адреса к локальной сети**

**if( iptemp == 0 ) memcpy(ip,ip\_addr,4);**

**else memcpy(ip,ipgate,4);**

That is, if the condition is not confirmed, that is, the address will not be local, then we will ask for the MAC address from the router and all further packets destined for the node we will send to the physical address of the router (or router).

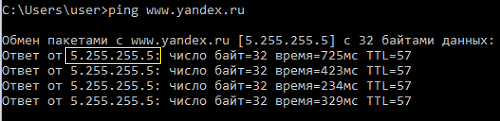
Accordingly, in the future code in the body of the function, we will correct the use of the array on our local

**if(!memcmp(arp\_rec[j].ipaddr,ip,4))**

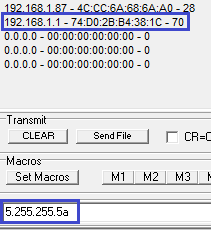
...

**memcpy(msg->ipaddr\_dst,ip,4);**

Let's collect the code, we'll tell the controller and check our code, sending the terminal program first by requesting an IP address first to the local network, and then to the remote one. For a remote network, we will use for example the IP address of the site " **www.yandex.ru** ", which we can learn by pinging it on the command line



Here's what we should see in the terminal program after sending ARP requests



Judging by the report in the program, we did everything right and on an external IP-address, our router returned our MAC-address to us.

Good. Part of the case is done. Now our task is to ensure that not the router, and the external node to us something returned. So let's try to "ping" it by sending it an **ICMP** poll, or a few. We have not done this yet, we just answered such requests, so we will need a separate function, which we add in the net.c file after the function ip\_read

**//-----------------------------------------------**

**uint8\_t icmp\_request(uint8\_t\* ip\_addr)**

**{**

**uint8\_t res=0;**

**return res;**

**}**

**//--------------------------------------------------**

Also, to control the operation of this function while writing code to its body, it is necessary to call it somewhere. To do this, we add another condition to the handler from USART, through which we will send ICMP requests. They will be tracked using a string with the ip-address of the character ' **p** ' from the word "ping"

  net\_cmd();

}

**else if (b=='p')**

**{**

**usartprop.is\_ip=4;//статус попытки отправить ICMP-пакет**

**net\_cmd();**

**}**

We return to the file  **net.c** , only in the **net\_cmd** function   and by analogy with the sending of the **UDP** request  , we will add the sending of the **ICMP** request

    usartprop.is\_ip=0;

  }

**else if(usartprop.is\_ip==4)//статус попытки отправить ICMP-пакет**

**{**

**ip\_extract((char\*)usartprop.usart\_buf,usartprop.usart\_cnt,ip);**

**usartprop.is\_ip=5;//статус отправки ICMP-пакета**

**usartprop.usart\_cnt=0;**

**arp\_request(ip);//узнаем mac-адрес**

**}**

**else if(usartprop.is\_ip==5)//статус отправки ICMP-пакета**

**{**

**icmp\_request(ip);**

**usartprop.is\_ip=0;**

**}**

}

In the same way, as usual, we first send an ARP request, then we return here with the next status after the ARP response or when there is a complete pair of addresses in the ARP table. Therefore, we must also add a new status in the return places to the net\_cmd function.

First, in the function  **eth\_read** , and then certain functions  **arp\_request**  file  **arp.c**

if**(**(usartprop.is\_ip==3)**||(usartprop.is\_ip==5))**//статус отправки UDP-**или ICMP-**пакета

Well, after returning now not empty-handed to the function  **net\_cmd** , we boldly send an **ICMP** request  , calling the corresponding function whose body code we are now and will continue to write.

But before continuing, first add a global variable to count the **ICMP** packets sent , since for such a number there is a separate field in the header

char str1[60]={0};

**uint32\_t ping\_cnt=0;//счетчик отправленных пингов**

We return to the function  **icmp\_request** and first, as usual, create a variable to calculate the length of the packet, and then arrange the pointers to the packet headers in the corresponding places

uint8\_t res=0;

**uint16\_t len;**

**enc28j60\_frame\_ptr \*frame=(void\*) net\_buf;**

**ip\_pkt\_ptr \*ip\_pkt = (void\*)(frame->data);**

**icmp\_pkt\_ptr \*icmp\_pkt = (void\*)ip\_pkt->data;**

We fill the ICMP header with the required values ​​in its fields, not forgetting to increment the packet numbers, as well as about a data line that has a certain composition that can be read in the WireShark network traffic analysis utility as part of any ICMP packet

icmp\_pkt\_ptr \*icmp\_pkt = (void\*)ip\_pkt->data;

**//Заполним заголовок пакета ICMP**

**icmp\_pkt->msg\_tp = 8;**

**icmp\_pkt->msg\_cd = 0;**

**icmp\_pkt->id = be16toword(1);**

**icmp\_pkt->num = be16toword(ping\_cnt);**

**ping\_cnt++;**

**strcpy((char\*)icmp\_pkt->data,"abcdefghijklmnopqrstuvwabcdefghi");**

**icmp\_pkt->cs = 0;**

**len = strlen((char\*)icmp\_pkt->data) + sizeof(icmp\_pkt\_ptr);**

**icmp\_pkt->cs=checksum((void\*)icmp\_pkt,len,0);**

Next on the hierarchy, fill the IP header

icmp\_pkt->cs=checksum((void\*)icmp\_pkt,len,0);

**//Заполним заголовок пакета IP**

**len+=sizeof(ip\_pkt\_ptr);**

**ip\_pkt->len=be16toword(len);**

**ip\_pkt->id = 0;**

**ip\_pkt->ts = 0;**

**ip\_pkt->verlen = 0x45;**

**ip\_pkt->fl\_frg\_of=0;**

**ip\_pkt->ttl=128;**

**ip\_pkt->cs = 0;**

**ip\_pkt->prt=IP\_ICMP;**

**memcpy(ip\_pkt->ipaddr\_dst,ip\_addr,4);**

**memcpy(ip\_pkt->ipaddr\_src,ipaddr,4);**

**ip\_pkt->cs = checksum((void\*)ip\_pkt,sizeof(ip\_pkt\_ptr),0);**

And, at the end - the header of the Ethernet packet, which is then sent to the network

  ip\_pkt->cs = checksum((void\*)ip\_pkt,sizeof(ip\_pkt\_ptr),0);

**//Заполним заголовок пакета Ethernet**

**memcpy(frame->addr\_src,macaddr,6);**

**frame->type=ETH\_IP;**

**enc28j60\_packetSend((void\*)frame,len + sizeof(enc28j60\_frame\_ptr));**

  return res;

}

I think it's clear with the code, we've chewed this code more than once. Well, it's good, now we do not know about network packets, about their formation, reading, the composition of their headings and the methods of sending.

We'll collect the code, we'll tell the controller, and while we try to send the ping to our computer, since if we send it to an external network, it will not be of any use, since we will not see the answer, and sending it to the computer, we'll see it at least in Utilize WireShark . We will send several ICMP requests in the terminal program

Image03

and see the result in the utility (click on the image to enlarge the image)

[Image02_0500](http://narodstream.ru/wp-content/uploads/2017/06/Image02-2.png)

We can see that all of our packages came and sent a response to everything on them.

The next task is to see the answer in the terminal program, and, of course, not only this, but also the answers from remote nodes.

To do this, go to the **icmp\_read** function   and separate the two conditions from each other in the body of the function

**if(len>=sizeof(icmp\_pkt\_ptr))**

**{**

**if(icmp\_pkt->msg\_tp==ICMP\_REQ)**

**{**

    icmp\_pkt->msg\_tp=ICMP\_REPLY;

Now we add one more condition in the lower nesting level of the conditions, which will react to another type of packet - the PING response

  HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);

}

**else if (icmp\_pkt->msg\_tp==ICMP\_REPLY)**

**{**

**sprintf(str1,"%d.%d.%d.%d-%d.%d.%d.%d icmp reply\r\n",**

**ip\_pkt->ipaddr\_src[0],ip\_pkt->ipaddr\_src[1],ip\_pkt->ipaddr\_src[2],ip\_pkt->ipaddr\_src[3],**

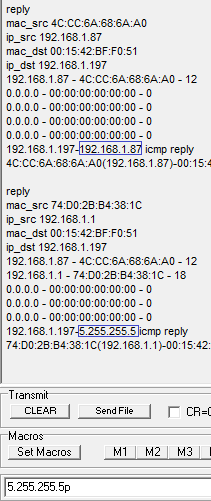
**ip\_pkt->ipaddr\_dst[0],ip\_pkt->ipaddr\_dst[1],ip\_pkt->ipaddr\_dst[2],ip\_pkt->ipaddr\_dst[3]);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**}**

Now we will see the answer in our terminal program.

Therefore, we will compile our code, sew our controller and send first several pings to the local node, and then several queries to the global node. Somewhat because we need to make sure that this all works continuously and regardless of whether we send an ARP request or retrieve the ready physical address from the table. We will have to see these answers in the terminal program window



As we can see, packages come to us from both local IP addresses and global ones.

Thus, during our today's lesson, we learned how to communicate on the network now not only with devices in our local network, but also with devices and nodes that are located remotely - in the global network, in our case - the Internet.