**Lesson 83**

**Part 1**

**LAN. ENC28J60. NTP. Find out the exact time**

Since at the [**last lesson**](http://narodstream.ru/stm-urok-82-lan-enc28j60-udalyonnyj-dostup/) we managed to communicate with the remote node, and we have practically mastered the transport level **protocol** - **UDP** , we can already get some data from the external network, in a particular case from the global world network - the Internet. Therefore, the goal of today's lesson is to get the exact time from one of the servers of the exact time using the **NTP (Network Time Protocol)**, which, according to the OSI model, is already an application layer protocol and carries in itself an already realized useful load. With protocols of this level, we have not met, well, it's nothing to worry about. Simply by the hierarchy of the protocole, the header and data of this level will be the data of some transport-level trans- port, and that's all. As a transport level, we will take UDP, since we are already familiar with it and are able to transmit data on it.

The **NTP** protocol is   also structured, as are all the protocols that we have considered so far, that is, it has an informed header with certain fields, which is basically also data. Also, along with structured protocols, there are also unstructured ones that do not contain a conscious or tangible heading in themselves, such as an HTTP prototype, which is fully textual, and the delimitation of data is determined either by tags or by a line feed. Before this protocol we will also get there shortly.

I will give the title of NTP (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2017/07/image00.png)

As we can see, this protocol has a rather big headline. Therefore, it would be possible in principle to get familiar with some of its fields in the process of writing code, but I still decided to give some information on these fields at once. To whom this is well known, it is possible to skip this explanation and go straight to writing the code and its subtleties.

Let's start.

**IR (correction identifier)** is an integer indicating a warning of the second of coordination (coordination second or leap second is an additional second added to the world coordinated time to match it with the average solar time UT1).

0 - No warning

1 - The last minute of the day contains 61 seconds

2 - The last minute of the day contains 59 seconds

3 - Unknown (time not synchronized)

**Version**  is an integer representing the version of the protocol.

**Mode**

0 - Reserved

1 - symmetric active mode

2 - balanced passive mode

3 - customer

4 - server

5 - broadcast mode

6 - NTP monitoring message

7 - reserved for private use.

**Time Layer**

0 - undefined or invalid

1 - primary server

2-15 - secondary server using NTP

16 - not synchronized

17-255 - Reserved.

Primary servers from secondary are more accurate, and. respectively, its server capacity. But to wait for an answer from them is much more difficult.

**The polling interval**  is a signed integer representing the maximum interval between consecutive messages. The value is equal to the binary logarithm of seconds. The default limits for minimum and maximum polls are 6 and 10, respectively.

**The precision**  is (sys.precision, peer.precision, pkt.precision). This is an entire signed variable indicating the accuracy of the clock in seconds and expressed as the nearest power of 2. The value should be rounded up to the closest power of 2, for example, the network frequency of 50 Hz (20 ms) or 60 Hz (16.67 ms ) will be assigned a value of -5 (31.25 ms), while a quartz frequency of 1000 Hz (1 ms) will be assigned a value of -9 (1.95 ms).

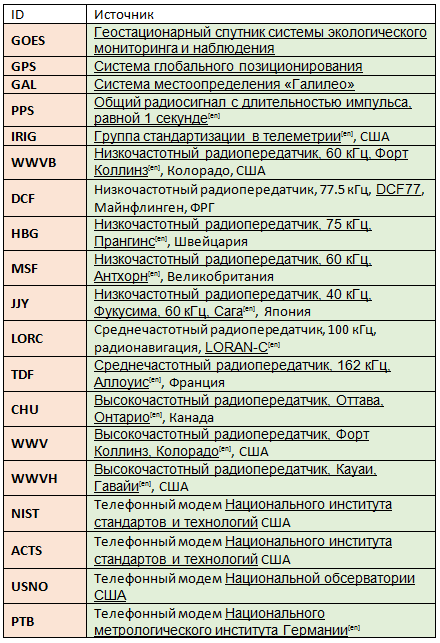
**Delay**  is the total propagation time of the signal in both directions in a short NTP format. The short format of NTP is a format where seconds are represented in the 16 most significant bits - and the fractions are 16 lower bits, the long format is the same, only 16 bits are replaced by 32 bits instead of 16 bits.

The thing is. that time on NTP servers in all fields of its header is stored unstructurally, that is, it is not broken into days, months, hours, minutes, etc., as it is done in RTC chips. Time there is stored in the number of seconds and fractions of seconds.

**Dispersion** - a fixed-point number greater than zero, carrying the maximum value of the time error with respect to the primary standard in seconds.

**Source identifier is the source**  code of the synchronization. Depends on the value in the "clock layer" field. It is stored in the 4-octet ASCII line format, aligned to the left and added with zeros if necessary.

Here are examples of such identifiers



**The update time**  is the time when the system last time set or corrected the time. It is stored in full NTP format.

**Start time**  is the time of the client when the request is sent to the server. It is stored in full NTP format.

**Receiving Time**  is the time of the server when the request comes from the client. It is stored in full NTP format.

**The dispatch**  time is the server time, when the request is sent to the client. It is stored in full NTP format.

Here approximately so with the report. About the last four fields: we are most interested in the last time, we will read it. A significant part of the fields will not be of interest to us at all, and many of them do not work at all in the case of the client, so we will fill them with zeros. If the information I provided on the NTP protocol is not enough or someone will be interested in knowing more about this topic, then such information in the network is quite abundant. I know myself. When I began to study this protocol, I even got confused. But then, after looking at the various examples of other bloggers, I eliminated all the confusion in my head.

In order to achieve the main goal of our lesson - to get to know the world time by all means, I had to resort to the timer counter, to the number of attempts to send a request to the server, because the server for some reason does not respond to all requests. And there are a lot of servers for these, I chose the server that is selected by default in my Windows operating system as the main server to correct the exact time. How to find out its IP address (and it is he who needs it, since the DNS protocol, which determines IP by domain name, is not yet known to us), I will show later in the process of writing the code, which we will now begin and write.

The project was created with the name  **ENC28J60\_NTP** from the project of the [**last lesson**](http://narodstream.ru/stm-urok-82-lan-enc28j60-udalyonnyj-dostup/) **ENC28J60\_REMOTE** .

As always, run the project in the code generator **Cube MX** , and then, nothing in it without touching, run it in **Keil**. Add all our library files to the project, and also configure the programmer to autocut.

To work with the NTP protocol, I decided to create a separate library in the form of a pair of files  **ntp.c**  and  **ntp.h**  with the following content

**ntp.c** :

**#include "ntp.h"**

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

**extern UART\_HandleTypeDef huart1;**

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

**extern char str1[60];**

**extern uint8\_t net\_buf[ENC28J60\_MAXFRAME];**

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

**ntp.h** :

**#ifndef NTP\_H\_**

**#define NTP\_H\_**

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

**#include "stm32f1xx\_hal.h"**

**#include <string.h>**

**#include <stdlib.h>**

**#include <stdint.h>**

**#include "enc28j60.h"**

**#include "net.h"**

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

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

**#endif /\* NTP\_H\_ \*/**

In today's lesson, we will send NTP requests as well as many requests from the command line of the terminal program, specifying the IP address and port of the exact time server. The port could not be included in the command line, since the reserved port for NTP protocol transmission is 123, but it is better to specify if there are not enough servers with a different port number than the devil does not joke, and it's not hard to enter a combination of numbers 1 2 3. Well, as many of you, I hope, guessed it. the line will end with the letter  **n** .

Therefore, go to the **net.c** file   in the **UART1\_RxCpltCallback** byte **handler**  and add the corresponding condition to the body of this handler

  net\_cmd();

}

**else if (b=='n')**

**{**

**usartprop.is\_ip=6;//статус попытки отправить NTP-пакет**

**net\_cmd();**

**}**

In the **net\_cmd** function,  **we**  add the appropriate conditions for trying and sending the packet itself

  usartprop.is\_ip=0;

}

**else if(usartprop.is\_ip==6)//статус попытки отправить NTP-пакет**

**{**

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

**usartprop.is\_ip=7;//статус отправки NTP-пакета**

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

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

**}**

**else if(usartprop.is\_ip==7)//статус отправки NTP-пакета**

**{**

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

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

**}**

Literally, the whole code of conditions is similar to the code of the last lesson, so it does not need an explanation.

Next, at the return points in the **net\_cmd** function,  **we**  add the status of sending the **NTP** packet

if((usartprop.is\_ip==3)||(usartprop.is\_ip==5**)||(usartprop.is\_ip==7)**)//статус отправки UDP-**,** ICMP**-** **или NTP** пакета

In the file  **net.h we** connect our new library, as usual,  **at the bottom of the**  file

#include "udp.h"

**#include "ntp.h"**

In the **ntp.c** file,  **we**  add the function of sending the NTP packet

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

**uint8\_t ntp\_request(uint8\_t \*ip\_addr, uint16\_t port)**

**{**

**uint8\_t res=0;**

**return res;**

**}**

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

Add a prototype to it and call it in  **net.c**  in the **net\_cmd** function   in the body of the corresponding condition

else if(usartprop.is\_ip==7)//статус отправки NTP-пакета

{

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

**ntp\_request(ip,port);**

Back in the file  **ntp.h**  and first add a structure to split the whole and fractional parts of seconds

#include "net.h"

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

**typedef struct ntp\_ts {**

**uint32\_t sec;//целая часть**

**uint32\_t frac;//дробная часть**

**} ntp\_ts\_ptr;**

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

We will also add a structure for the NTP header

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

**typedef struct ntp\_pkt {**

**uint8\_t flags; //флаги**

**uint8\_t peer\_clock\_stratum;//страта**

**uint8\_t peer\_pooling\_interval;//Интервал опроса**

**uint8\_t peer\_clock\_precision;//Точность**

**uint32\_t root\_delay;//Задержка**

**uint32\_t root\_dispersion;//Дисперсия**

**uint32\_t ref\_id;//Идентификатор источника**

**ntp\_ts\_ptr ref\_ts;//Время обновления**

**ntp\_ts\_ptr orig\_ts;//Начальное время**

**ntp\_ts\_ptr rcv\_ts;//Время приёма**

**ntp\_ts\_ptr tx\_ts;//Время отправки**

**} ntp\_pkt\_ptr;**

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

As we can see, there will not be separate data for the package, only the header.

Also, let's write a macro for the client port so that it is separate, since the server type 333 can not really like the option

#include "net.h"

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

**#define LOCAL\_PORT\_FOR\_NTP 14444**

Let's **go** to the file **ntp.c** and connect the arrays of MAC addresses and IP addresses

extern uint8\_t net\_buf[ENC28J60\_MAXFRAME];

**extern uint8\_t macaddr[6];**

**extern uint8\_t ipaddr[4];**

Continue to write the code for our function to send the NTP package **ntp\_request** .

Create a variable for the length and create pointers for the headers of all levels of our package

uint8\_t res=0;

**uint16\_t len;**

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

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

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

**ntp\_pkt\_ptr \*ntp\_pkt = (void\*)(udp\_pkt->data);**

Fill the whole NTP header with zeros

ntp\_pkt\_ptr \*ntp\_pkt = (void\*)(udp\_pkt->data);

**//заполним нулями всю структуру ntp**

**memset(ntp\_pkt, 0, sizeof(ntp\_pkt\_ptr));**

Next, fill in the necessary fields of the NTP header. The rest we do not need

memset(ntp\_pkt, 0, sizeof(ntp\_pkt\_ptr));

**//Заполним заголовок NTP**

**ntp\_pkt->flags = 0x1b;**

We see that we have filled only the flags. We do not need anything else.

We go down the protocol ladder OSI. The next protocol is UDP

ntp\_pkt->flags = 0x1b;

**//Заполним заголовок UDP**

**udp\_pkt->port\_dst = be16toword(port);**

**udp\_pkt->port\_src = be16toword(LOCAL\_PORT\_FOR\_NTP);**

**len = sizeof(ntp\_pkt\_ptr) + sizeof(udp\_pkt\_ptr);**

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

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

**udp\_pkt->cs=checksum((uint8\_t\*)udp\_pkt-8, len+8, 1);**

Next, fill the lower level header - IP

udp\_pkt->cs=checksum((uint8\_t\*)udp\_pkt-8, len+8, 1);

**//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\_UDP;**

**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);**

I think here everything is also very clear. We have done this many times.

The next level is the channel level. Therefore, fill the Ethernet header and send our packet

  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;

}

Next, we have nothing to do but wait for the NTP server response in the same way as we waited for the answer of remote nodes in the last lesson, sending ICMP packets.

Only today it will be weighed down by the fact that we can not see if our package has come to the server, since we will not send such a request to the local computer. Rather, we can send it, only the computer will not answer us, because there is not organized an NTP-server.

Therefore we will wait. And to wait not at the network level, but on the transport.

So first, let's  add **ntp.c** in our file  **ntp.c** above another function to read the NTP header

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

**uint8\_t ntp\_read(enc28j60\_frame\_ptr \*frame, uint16\_t len)**

**{**

**uint8\_t res=0;**

**return res;**

**}**

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

Create a prototype for this function, and then zaydom file  **udp.c**  in function  **udp\_read**  and in the appropriate place check the port, whether it is the standard port of **the NTP** -  **123** , in front of this moving higher setting pointers to packets

uint8\_t res=0;

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

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

**if(be16toword(udp\_pkt->port\_src)==123)**

**{**

**ntp\_read(frame,len);**

**return 0;**

**}**

Now if the port at us coincides with the value of  **123** , then we call the read function of the NTP package and exit the function without executing any further code.

We return to the file  **ntp.c**  and continue to write the body of our function of receiving the NTP packet.

As always, we first set pointers to headings of different levels

uint8\_t res=0;

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

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

**ntp\_pkt\_ptr \*ntp\_pkt = (void\*)(udp\_pkt->data);**

Well, accordingly we will need only one field from the NTP header - this is the sending time.

But since it will be in us in seconds, since we will not use the fractional part at all, we do not need microseconds and other shares at all, then we will have to choose one of these seconds, a year, a month, and everything else. I first wanted to write my algorithm and then realized that it was useless, because there would be a very long and complicated algorithm, especially in calculating the date, you need to take into account the different leap-and-run trivia, and in the standard library we already have ready-made functions, just connect the appropriate header file, which I did in the file  **ntp.h**

#include "net.h"

**#include <time.h>**

From this library we will use a structure of type **tm** , a pointer to which we will add in the file  **ntp.c**  in the function  **ntp\_read**

uint8\_t res=0;

**struct tm \*timestruct;**

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

Since seconds are also stored in the format of  **big endian**  (with the reversal of the byte precedence), and the number here is already 32-bit, we need to add one more definal replacement to the file  **net.h**

#define be16toword(a) ((((a)>>8)&0xff)|(((a)<<8)&0xff00))

**#define be32todword(a) ((((a)>>24)&0xff)|(((a)>>8)&0xff00)|(((a)<<8)&0xff0000)|(((a)<<24)&0xff000000))**

Let's return to our **ntp\_read** function   and show in the terminal program until the raw seconds value without structuring

ntp\_pkt\_ptr \*ntp\_pkt = (void\*)(udp\_pkt->data);

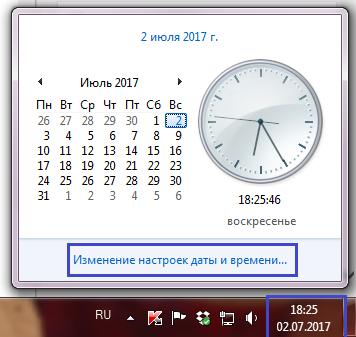
**sprintf(str1,"%lu\r\n", be32todword((unsigned long)ntp\_pkt->tx\_ts.sec));**

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

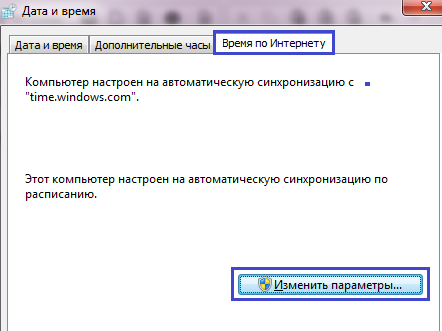
Now let's find out the network address of the NTP server. To do this, run WireShark and filter the packet analysis by NTP

image03

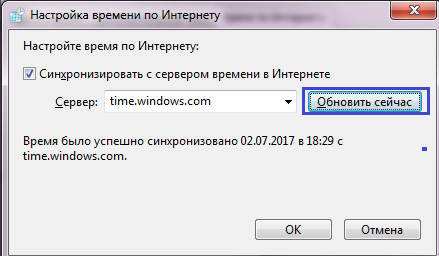
Clicking on the time in the tray, we run the time change



In the opened dialog, go to the tab "Time on the Internet", and there we run "Change parameters"



Leave the server by default and click "Update Now", then wait for the time synchronization



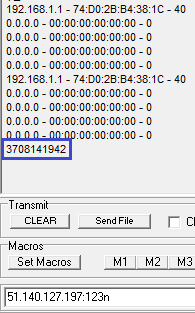
Now we can close all the dialogs and go to Wireshark and from the packet take the IP address of the NTP server

image07

In order not to write manually the address, you can pick it up using the context menu items - Copy-> Value.

Also you can find a list of servers for exact time on the Internet and, moreover, also see their workload at the moment. So the ways of recognizing the addresses of these servers are different, and what is convenient for you - you choose.

Finally, we can already compile our code, flash our controller and see the result of our work in the terminal program



As we can see, the package came to us from the server, but only with 2 attempts. But it's nothing, it's not difficult for us to make 2, 3 and more attempts, I would like, of course, that the attempts take place automatically, without our participation, but more on this later.

In the [**next part of**](http://narodstream.ru/stm-urok-83-lan-enc28j60-ntp-uznayom-tochnoe-vremya-chast-2/) our lesson, we will see the time from the NTP server in a convenient format, and also automate the process of re-querying the time.

**Lesson 83**

**Part 2**

# LAN. ENC28J60. NTP. Find out the exact time

In the [**previous part of**](http://narodstream.ru/stm-urok-83-lan-enc28j60-ntp-uznayom-tochnoe-vremya-chast-1/) this lesson we learned all that is necessary about the NTP protocol and wrote the code that sends the request, waits for a response from the server and displays the world time in the terminal program in the form of a time stamp in seconds.

Now we are faced with the task of converting the time in seconds into a readable format.

First, in the **ntp.h** file **,** add a macro for the time zone. We have a third time zone, I hope you know your own

#define LOCAL\_PORT\_FOR\_NTP 14444

**#define TIMEZONE 3**

Now let's return to our function and write the conversion code, for this we have the library time.h

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

**time\_t rawtime = (time\_t)(be32todword(ntp\_pkt->tx\_ts.sec)-2208988800+60UL\*60\*TIMEZONE);**

**timestruct = localtime(&rawtime);**

**sprintf(str1,"%02d.%02d.%04u %02d:%02d:%02d -%d-rn", timestruct->tm\_mday,timestruct->tm\_mon+1, //месяцы считаются от 0, а не от 1**

**timestruct->tm\_year+1900,timestruct->tm\_hour,timestruct->tm\_min,timestruct->tm\_sec,timestruct->tm\_wday);**

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

By the year 1900 is added, since from that moment they are considered taking into account the bias, this is indicated in the library time.h.

Displacement we use because of the fact that on NTP-servers time is counted from another year (the difference is 20 years or 2208988800 seconds).

Also we move forward in seconds to the time zone value multiplied by the number of seconds in the hour.

Then we just have to display everything in the terminal program

We will collect our code once again, we will sew the controller and see the result, making sure that it corresponds to the real time

image01

Days of the week are counted in the library from 0 to 6, so Sunday will be 0

**int tm\_wday; /\* days since Sunday, 0 to 6 \*/**

If you want, you can write a condition to make day 0 turn into 7, but we should not face such tasks.

We have a more serious task - to get time to do something that does not become, but it does not come at once, so we automate our inquiries well, or attempts. It was not so easy.

We will create for this purpose one more structure for the status of our NTP client in the file  **ntp.h**

} ntp\_pkt\_ptr;

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

**typedef struct ntp\_prop{**

**uint8\_t ntp\_cnt; //количество оставшихся попыток получить время**

**int32\_t ntp\_timer; //таймер для следующей попытки**

**uint8\_t set; //флаг получения времени**

**uint8\_t macaddr\_dst[6];**

**uint8\_t ip\_dst[4];**

**uint16\_t port\_dst;//порт получателя**

**} ntp\_prop\_ptr;**

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

In principle, the purpose of almost every field I highlighted in the comments

Since attempts will be made from another location, not always from **net\_cmd** , then the decision was taken to store all addresses in the structure.

In the **ntp.c** file,  **we**  create a variable of the type of our structure

extern uint8\_t ipaddr[4];

**ntp\_prop\_ptr ntpprop;**

In the file  **net.c**  we also connect it

USART\_prop\_ptr usartprop;

**extern ntp\_prop\_ptr ntpprop;**

In the initialization function of **net\_ini** , **we**  initialize the main fields of our structure just in case

enc28j60\_ini();

**ntpprop.set=0;**

**ntpprop.ntp\_cnt=0;**

**ntpprop.ntp\_timer=0;**

In one of the conditions in the **net\_cmd** function, **let's** put the **ip** address   in the appropriate field of our structure

else if(usartprop.is\_ip==6)//статус попытки отправить NTP-пакет

{

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

**memcpy(ntpprop.ip\_dst,ip,4);**

Also in this same function we will add a pointer to our package

static uint16\_t port=0;

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

In the following condition, we also enter the values ​​in the corresponding fields of the structure and use them just in case in the call to send the NTP request

else if(usartprop.is\_ip==7)//статус отправки NTP-пакета

{

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

**ntpprop.port\_dst = port;**

**ntpprop.ntp\_cnt = 10; //10 попыток**

**ntpprop.ntp\_timer = 5;//5 секунд до следующей попытки**

**ntpprop.set=1;//флаг запроса времени взведен**

**memcpy(ntpprop.macaddr\_dst,frame->addr\_dest,6);**

  ntp\_request(**ntpprop.ip\_dst,ntpprop.port\_dst**);

In general, here we are assigned to make 10 attempts for the NTP request, also a time interval of 5 seconds between attempts and the time request flag is turned on, which we will reset with a successful server response.

Now we go to the timer handler  **TIM\_PeriodElapsedCallback**  and add the following condition, which will determine if the NTP request flag is set at all

clock\_cnt++;

**if (ntpprop.set)**

**{**

**}**

Then, in the body of the added condition, we will reduce the seconds until the next attempt

if (ntpprop.set)

{

**ntpprop.ntp\_timer--;**

}

Then we add in this condition one more condition that confirms that the timer has reached zero, and the number of attempts has not yet ended

ntpprop.ntp\_timer--;

**if ((ntpprop.ntp\_timer<0)&&(ntpprop.ntp\_cnt>0))**

**{**

**}**

And in the body of this condition, we again set the time until the next attempt, we will reduce the number of attempts and send the request to the NTP server, before displaying the number of remaining attempts in the terminal program

if ((ntpprop.ntp\_timer<0)&&(ntpprop.ntp\_cnt>0))

{

**ntpprop.ntp\_timer = 5;**

**ntpprop.ntp\_cnt--;**

**sprintf(str1,"ntp\_cnt: %drn",ntpprop.ntp\_cnt);**

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

**ntp\_request(ntpprop.ip\_dst,ntpprop.port\_dst);**

}

We come out only from this condition and add the following, in which we reset all the flags

  ntp\_request(ntpprop.ip\_dst,ntpprop.port\_dst);

}

**else if (ntpprop.ntp\_cnt<=0)**

**{**

**//сбросим все флаги и счетчики**

**ntpprop.set=0;**

**ntpprop.ntp\_cnt=0;**

**ntpprop.ntp\_timer=0;**

**}**

Let's pass to the function of sending the NTP request  **ntp\_request**  to the file  **ntp.c**  and enter the MAC address of the receiver in the corresponding field, as it can be distorted if there is a reception or sending of other packets between the attempts, our buffer can be refilled, and in this function We do not explicitly mention this address anywhere.

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

**memcpy(frame->addr\_dest,ntpprop.macaddr\_dst,6);**

frame->type=ETH\_IP;

Well, also in the function of receiving the NTP packet from the server  **ntp\_read,**  we also reset the flags of our structure in case the packet does not come with the last attempt, so that we no longer need to send more requests in case of a successful response

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

**//сбросим все флаги и счетчики**

**ntpprop.set=0;**

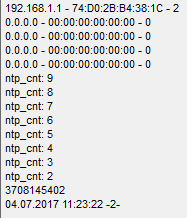
**ntpprop.ntp\_cnt=0;**

**ntpprop.ntp\_timer=0;**

  return res;

}

We will collect the code. We will sew a controller and see the result of our work



Well, the server returned the time already with 9 attempts, usually somewhere from 2 or 3. With the 1st very rare.

Also I tried to run endless pings on our module from the computer and at the same time ask for the exact time. The server returned it. Therefore, in this case, everything also works for us.

You can set your time until the next attempt, also your number of attempts.

The most important thing is for you to understand the very meaning of the work of the protocols and calculations in them, which I hope I have given you. Of course, it is not necessary to do everything in this way. You can come up with your own way of asking for time. Nevertheless, I hope that my lesson will benefit you and your thought or your vision of this issue to me was passed on to you.

One day, armed with the knowledge from this lesson, we will probably write some clock with synchronization from the Internet.

Thank you for attention!