**STM Lesson 118. LAN8720. LWIP. RAW. UDP Client**

Posted on [May 10, 2018](http://narodstream.ru/stm-urok-118-lan8720-lwip-raw-udp-client/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Published in [Programming STM32](http://narodstream.ru/rub_stm32/)- [No Comments ↓](http://narodstream.ru/stm-urok-118-lan8720-lwip-raw-udp-client/#respond)

[Programming LessonsLearn all the in-house cooking **programming** training ! Get the book on the e-mail!To learn moresheremetev.info](https://an.yandex.ru/count/NP_-euB8jJW50Dy1CUs-ari00000ECgs7402I09Wl0Xe172WmvR10O01sE3CgmU80QE8tP0fa06cakID99W1eiM-tYEW0SAapOqag07iWR3U8xW1tAUPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y083e0BUkAe2kGA97mLkFaJ1EV02-F2dl0lu0eA0W820e1I00-RYgP81Y0EIhhwt29W3ky06g0CIi0C4k0J_0UW4jWFu1Ekk0eW5wwu2a0M3nmAW1UrQg0N_JB05_qou1P9Jm0NmcGV81R3g0T05qYBW1GRm1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XHrAKIBSL8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2A-SbG602W712W0000000F0_s0e2u0g0YNhu2i3y5OWB1geB41OgPjFuKG000qSJB5jr1G302u2Z1SWBWDIJ0TaB-57KfH8jnKZe2-kk0l0B1eWCjfNUlW7e30AO3PMCQF8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_uex6VdwBBuaW0?stat-id=3&test-tag=89060471234561&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY3IjoiODkwNjA0NDE4ODI2MjQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Buy robot constructorTeach your child to program. 6900 for a set of shares. OrderTo learn moredorado-uno.ru](https://an.yandex.ru/count/GbrWSyodLC450Cu1CUw-ari00000ECgs7402I09Wl0Xe172AffoG1e01xCtW6OW1rBMagJMG0VZAnCelc07gZ-3LBg01zih4oY-e0UYEuDKkk07st_Rr6S010jW1ieMB5U01qBto4UW1OFW1elhUlW680WIW0g2-sHQv0eaV1Mu-HC4vy0BemDxg3lW2We20W82W5803uU7RcWc80wRI-B43i0C4w0I_0lW4etY81QDua0NvsmAW1Oiag0NA7h05oXwu1RjOm0Mm-mp81V2b0z05uGtW1Lhm1G6O1jwfZXcW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000Aa7-57eXrCjnKYm1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08aD-i2u0A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i8gWiGO8G_ilXH001sVoCiMtK50C0BWAC5o0k0r9C1sGluKUY7Kot5IEWBetZm2mQ83ABwthu1w0m2c0tylMRo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Em8Gzi0u1eGy00000003mFwWF-AEHxh2go-98?stat-id=4&test-tag=89060471234561&format-type=24&banner-test-tags=eyI2Mzg2Mzg5NjkyIjoiODkwNjA0NDE4ODI2MjQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

We return to the topic of data transfer via wire communication channels and also return to the data transfer module via LAN - **LAN8720** .

And before moving on to the NETCONN API, let's work with the RAW API again and write a simple UDP Client.

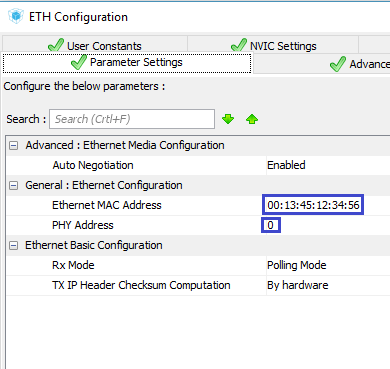
Firstly, it is caused by a huge number of requests for this protocol, and secondly, we are likely to first work with the UDP protocol in the NETCONN API and write the server, and for this we necessarily need a client. Here we will connect all this later.

The fee for our lesson will be the same as STM32F4-Discovery, and the LAN8720 chip will be used on the **DIS-BB** board . You, of course, can use a separate module from WaveShare, just do not forget to include the one in the " **PHY Addres** " field in the **ETH** settings in the settings in the Cube .

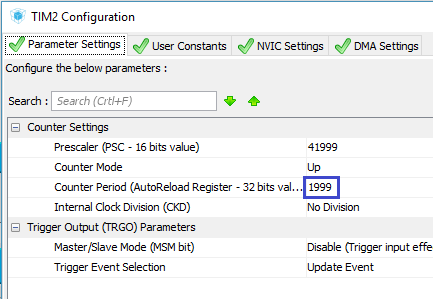
Since we already know perfectly well what the UDP transport level protocol is, and also feel good in programming the RAW API of the LWIP protocol stack, we can, in principle, go directly to the project.

In order not to suffer from the project settings, we will do the project on the basis of the project of the [**lesson 96**](http://narodstream.ru/stm-urok-96-lan8720-lwip-tcp-client-chast-1/) **LAN8720\_TCP\_CLIENT** and call it **LAN8720\_UDP\_CLIENT\_RAW** .

Open our project in the **Cube MX** , go to **Configuration** in the **ETH** settings , make sure that there is **0**in the physical address, and also slightly change the **MAC address**



Then go to the timer settings and set up a period of about 1 second



Also make sure that the interrupt from the timer we have included.

Generate the project and open it in the **System Workbench** .

As usual, configure the optimization level in 1, remove the debugging configuration if available, save the settings and try to collect our project.

If the project is going well, then we will start working with it.

Go to the file **net.c** and delete all the code there, except to connect the header file. Only this will remain

#include "net.h"

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

Add a handler from the timer, in which so far only change the level of the leg of the blue LED

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

**void TIM1\_Callback(void)**

**{**

**HAL\_GPIO\_TogglePin(GPIOD, GPIO\_PIN\_15);**

**}**

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

Add a global pointer variable to the UDP structure, a string array, and add the UDP connection setup function. Although this protocol does not provide for this, we will mainly have the initialization of the structure for UDP exchange, as well as the declaration of the function of the handler of incoming packets via UDP. While in the function we only call the constructor of the structure

#include "net.h"

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

**struct udp\_pcb \*upcb;**

**char str1[30];**

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

void udp\_client\_connect(void)

{

**ip\_addr\_t DestIPaddr;**

**err\_t err;**

**upcb = udp\_new();**

}

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

Go to the header file net.h, remove unnecessary prototypes of functions and add prototypes of the functions we added

~~void net\_ini(void);~~

~~void UART6\_RxCpltCallback(void);~~

**void udp\_client\_connect(void);**

**void TIM1\_Callback(void);**

The structure announced below will also be deleted.

Also, change the tcp plug-in to udp

#include "lwip/**udp**.h"

Go now to **main.c** , from **main (),**  remove the initialization function call and call the byte function from USART, and instead call the UDP connection organization function

~~net\_ini();~~

~~HAL\_UART\_Receive\_IT(&huart6,(uint8\_t\*)str,1);~~

**udp\_client\_connect();**

From the USART interrupt handler function, we remove the call to our handler function

if(huart==&huart6)

{

~~UART6\_RxCpltCallback();~~

}

After this function, we add another function-interrupt handler from the timer, in which we call our self-made function-handler

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

**void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)**

**{**

**if(htim==&htim2)**

**{**

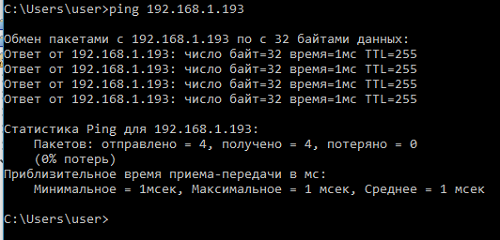
**TIM1\_Callback();**

**}**

**}**

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

At this stage, let's collect the project, let's tell the controller and try to ping our board in order to check its availability



Everything is normal! Module and board are available.

Let's return to the file **net.c** and add in it a prototype of the function-handler of incoming packets on UDP

char str1[30];

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

**void udp\_receive\_callback(void \*arg, struct udp\_pcb \*upcb, struct pbuf \*p, const ip\_addr\_t\*addr, u16\_t port);**

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

We continue to write the function **udp\_client\_connect** .

Add the verification condition for creating the structure

upcb = udp\_new();

**if (upcb!=NULL)**

**{**

**}**

In the body of this condition, we initialize the IP variable of the server to which we will send packets, and from which we will also receive them. As a server we will have a PC, so we will use the address exactly

if (upcb!=NULL)

{

**IP4\_ADDR( &DestIPaddr, 192, 168, 1, 87);**

}

Also, we will write the port of our client into the structure in the corresponding field. This port will be accessed by the PC to send packets

IP4\_ADDR( &DestIPaddr, 192, 168, 1, 87);

**upcb->local\_port = 1555;**

Call the connection function, in which as one of the input arguments we will also pass the server port (in our case, the programs on the PC)

upcb->local\_port = 1555;

**err= udp\_connect(upcb, &DestIPaddr, 1556);**

If the connection is normally initialized, we also declare the name of the handler function of the incoming packets over UDP

err= udp\_connect(upcb, &DestIPaddr, 1556);

**if (err == *ERR\_OK*)**

**{**

**udp\_recv(upcb, udp\_receive\_callback, NULL);**

**}**

After this function, we add the function of packet transfer, in the body of which we just declare the pointer to the buffer structure

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

**void udp\_client\_send(void)**

**{**

**struct pbuf \*p;**

**}**

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

After the function of transferring the package, we add the function-handler of the incoming packets by UDP

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

**void udp\_receive\_callback(void \*arg, struct udp\_pcb \*upcb, struct pbuf \*p, const ip\_addr\_t\*addr, u16\_t port)**

**{**

**strncpy(str1,p->payload,p->len);**

**str1[p->len]=0;**

**pbuf\_free(p);**

**HAL\_GPIO\_TogglePin(GPIOD, GPIO\_PIN\_12);**

**}**

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

In this function, we will copy the incoming data from the buffer to a string array, end it with zero, free the buffer memory and the green LED flashes. We will not show the data yet. We will need this in the next lessons, but for now this is not the purpose of our lesson, but we'll somehow see these data somehow later.

Call the function of sending a packet from the timer interrupt handler function

void TIM1\_Callback(void)

{

**udp\_client\_send();**

Now let's return to the function of passing the package  **udp\_client\_send** and something we will try to transfer there.

First, prepare a line for transmission, taking in it the number of system ticks that have passed since the controller was turned on or rebooted

struct pbuf \*p;

**sprintf(str1,"%lu\r\n",HAL\_GetTick());**

Let's allocate memory for the data in the buffer

sprintf(str1,"%lu\r\n",HAL\_GetTick());

**p = pbuf\_alloc(*PBUF\_TRANSPORT*, strlen(str1), *PBUF\_POOL*);**

If memory allocation was successful, then pass our line to the server and free the buffer

p = pbuf\_alloc(*PBUF\_TRANSPORT*, strlen(str1), *PBUF\_POOL*);

**if (p != NULL)**

**{**

**pbuf\_take(p, (void \*) str1, strlen(str1));**

**udp\_send(upcb, p);**

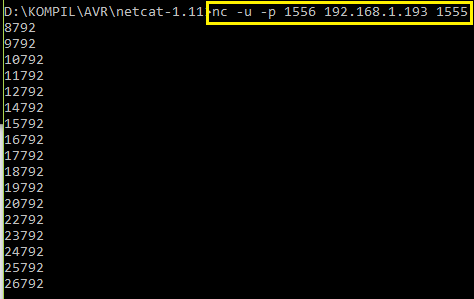
**pbuf\_free(p);**

**}**

We'll collect the code, we'll tell the controller and go to the PC.

We'll go into the directory with the netcat program and run the **cmd** command .

then on the command line we run the following command



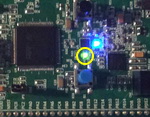
The value after the **-p** option is the netcat port number. It turns out that it can also be assigned, I only recently learned by studying the built-in help. I have not seen such information anywhere else, and then the IP address of the board is already coming, and then the port number of our connection in the board.

And, as we can see, the packages come in normally. Due to the very busy network, in which I checked all this, some packages are lost. We see this clearly. For example, after the package with the string "20792" a package with the string "22792" goes straight, which means that the package "21792" was lost. In general, thanks to the use of a timer, we packets are sent at a strictly specified time, as evidenced by the unchanged last three digits. Therefore, we set the timer correctly.

Now let's give something from the PC to the board. To do this, enter the command line without interrupting the connection of any line and press the "Enter"

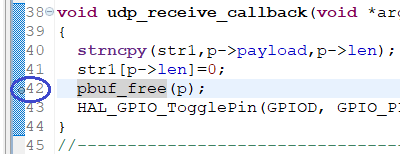


After that, the board should light green LED

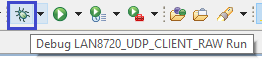


If we send another package, the LED should go out.

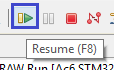
Let's still read from our controller the line transferred to it from the PC. To do this, put a breakpoint in the function-handler for receiving the packet here in this place



Run debugging with this button



Then run the project to run in debugging

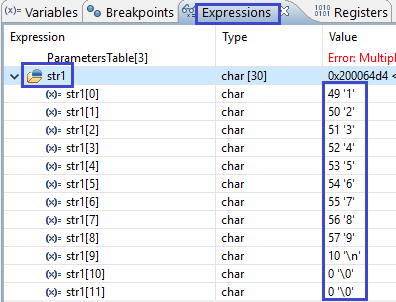


The project will start working, the packages will also go to the PC via the network.

Send the same line from the PC. After that we will have to get to the breakpoint

http://narodstream.ru/wp-content/uploads/2018/04/stm117img08.png

Add our line array to **Expression** and see its contents



Everything corresponds.

Stop debugging, return to the normal view, and run our project for execution. Now we can continue to enjoy receiving the sent lines in the PC.

So, today we wrote a simple client for working with the UDP protocol.

Thank you all for attention!