**STM Lesson 120. LAN8742A. LWIP. NETCONN. UDP Server**

Posted on [May 29, 2018](http://narodstream.ru/stm-urok-120-lan8742f-lwip-netconn-udp-server/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Published in [FreeRTOS](http://narodstream.ru/freertos/) , [LAN](http://narodstream.ru/lan/) , [Programming STM32](http://narodstream.ru/rub_stm32/)- [1 comment ↓](http://narodstream.ru/stm-urok-120-lan8742f-lwip-netconn-udp-server/#comments)

[How did I learn to draw at home?Participate in a series of webinars and get your favorite painting lessons immediately!To learn morenewdirections.ru](https://an.yandex.ru/count/V8T1tWmusWK50Cu1CJx0ari00000ECgs7402I09Wl0Xe173mgPjnW07Cl-VExEgVyII80OF0sv4ra06ojhIyBvW1YgYrjowW0SwtjBmlg07qfhMtBhW1qE_sxXd00GBO0UQClmtW0UgOlWte0Go80WEW0fBxjnMv0e07RD9drHWXy0BHjSIT3VW2We20W82W5803xQhkeGw80zpicVi2g0CIi0C4k0J_0UW4Ymhu1AMV0eW5fPy2a0NhgWAW1PXMg0M5IR05XKcu1VrBm0NmcGV81R3g0T05xo3W1Lhm1G6O1iZNn3UW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000Aa7-54Km7nznKYm1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08oVtR2e0A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i6gWiG3C7lJlXH001wpIr0MtK50C0BWAC5o0k0r9C1sGluKHJ0V7t5IEWBfPy2y0i6w0m2c0tud6No3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0E-QGTi0u1eGy00000003mFwWF-AFnlBslt-98?stat-id=3&test-tag=89060471234561&format-type=24&banner-test-tags=eyI2MzcxNTc3MjQxIjoiODkwNjA0NDE4ODI2MjQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Earn remotely, working 2-3 hours.Free course "Specialist in Internet Advertising" for 10 days! Sign up before 10.09.18To learn moreprofiinternet.artur-grant.ru](https://an.yandex.ru/count/IFcS3s4M0ku50Du1CJ_0ari00000ECgs7402I09Wl0Xe173qmTYm0e01u_CDY06viP23CP01-ewzZosO0UBngS8ge07yZhsFBQW1pEwfmYgu0PYHfjeOm042s068r8uIu06wgvW6w04a-041Y084e0AixfmHkGA01spIPzKO8SW2YehjmVgOsTX_y0AWrCKI-0A2W820WA0KW0FuZTH7Y0FJgPMd1x031BW4_m7e1E82-0JEw0I81Sxe1905zFu5e0N-k06e1UAS0R05ufm1k0NzmW701VM06CW5e8q6q0N5Hk052V050PW6WD2iymwW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO002rmSG000Aa7-54G4gTznKYm1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08bhVE280A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i8gWiGqj3a3FXH000NOZX0MtK50C0BWAC5o0k0r9C1sGluKH0Iftt5IEWBpkW4y0i6Y0o-lTw-0UWC0fWDoR9dyWq0-Wq0WWu0YGu00000003mFv0Em8Gzc0x6rypKbOdWkXEW3i24FR0E0Q4F00000000y3-e3_YZaVNYjD_YI000?stat-id=4&test-tag=89060471234561&format-type=24&banner-test-tags=eyI2MDU2MDI3MDY5IjoiODkwNjA0NDE4ODI2MjQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

Well, it's time to move to the more advanced LWIP protocol API - **NETCONN** .

We all know that we could not use it because we did not know how to work with the FreeRTOS operating system, which is a necessary condition for using this interface.

Now, after watching a series of classes on this operating system, we are already quite ready to work with NETCONN.

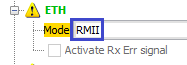
Working with the NETCONN interface and programming it is very similar to working with the RAW interface, only here is even more simplified the mechanism for configuring the structures, callback functions, etc. and much of it works through the library. Therefore, we do not need to configure this manually. In the rest, approximately all the same.

Well, what's wrong, we'll see in the programming process, so we will not delve into the theory of the interface, in the description of its structures and their fields, and also in the description of all functions. Better, I believe, still it is to feel on a real network exchange, which we will set up in this lesson with our own hands.

We will work with the debug card **STM32F746-Discovery** .

To us not to configure at least the operating system, the project we will make from the project [**lesson 111**](http://narodstream.ru/stm-urok-111-freertos-ocheredi-chast-1/)on working with **TASKS\_QUEUES** queues and call it **LAN8742\_UDP\_SERVER\_NETCONN** . Therefore, we will only have to configure the LWIP protocol stack and the ETH network interface.

Therefore, we will launch our project in Cube MX and first of all we will include **ETH**

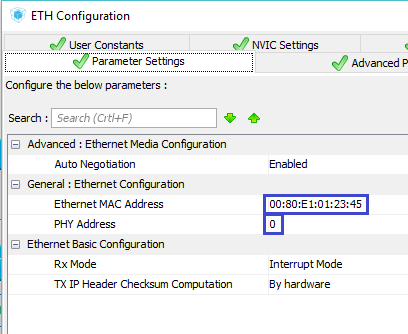


We do not redefine any legs.

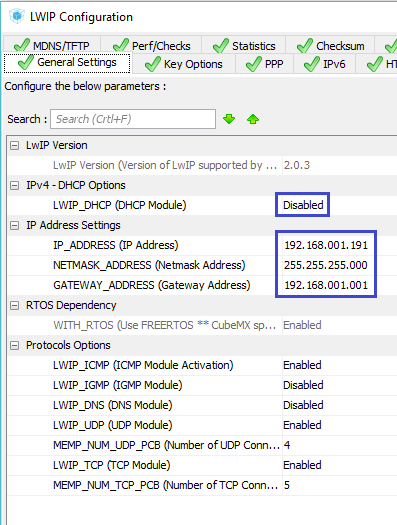
Also include the LWIP protocol stack library

http://narodstream.ru/wp-content/uploads/2018/05/stm120img01.png

Go to **Configuration** and start with the **ETH** setting . Although the chip in our board and LAN 8742A, but there is no difference in programming. So remember our lessons on LAN8720 and configure the interface



Next, configure **LWIP** . While we use static addressing



In the **Key Options** section,  you can not change anything.

Generate our project and open it in the System Workbench.

Set the optimization level to **1** , remove all debugging settings, comment out the lines in the DMA2D configuration, unknown to the compiler, and try to assemble the project.

If everything is ready, then we will start working with it.

First of all, of course, the hat.

Open the file **main.c** and in the **main ()** function, we will correct the header, removing all unnecessary

TFT\_DisplayString(0, 10, (uint8\_t \*)"**UDP Server**", *CENTER\_MODE*);

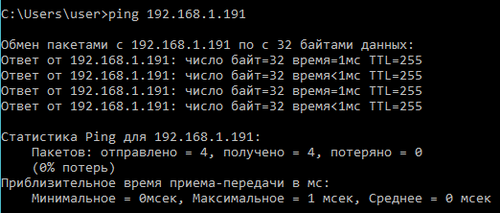
~~TFT\_SetTextColor(LCD\_COLOR\_MAGENTA);~~

~~TFT\_DisplayString(14, 60, (uint8\_t \*)"Task1:",~~*~~LEFT\_MODE~~*~~);~~

~~TFT\_DisplayString(14, 110, (uint8\_t \*)"Task2:",~~*~~LEFT\_MODE~~*~~);~~

~~TFT\_DisplayString(14, 160, (uint8\_t \*)"Task3:",~~*~~LEFT\_MODE~~*~~);~~

We will collect the project, we will fix the controller, not forgetting to connect our motherboard also to the network, and we will check its network availability by sending pings



First of all we will connect several modules necessary for working with the LWIP protocol stack

#include "fonts.h"

**#include "lwip/opt.h"**

**#include "lwip/arch.h"**

**#include "lwip/api.h"**

The three tasks that we worked with earlier, we will delete. We will not need them.

Accordingly, first remove the declarations of their identifiers

osThreadId ~~Task01Handle,Task02Handle,Task03Handle,~~TaskStringOutHandle;

Then delete the prototype of the task data function

~~void Task01(void const \* argument);~~

Remove the creation code from **main ()**

~~osThreadDef(tsk01, Task01,~~*~~osPriorityIdle~~*~~, 0, 128);~~

~~Task01Handle = osThreadCreate(osThread(tsk01), (void\*)&arg01);~~

~~osThreadDef(tsk02, Task01,~~*~~osPriorityIdle~~*~~, 0, 128);~~

~~Task02Handle = osThreadCreate(osThread(tsk02), (void\*)&arg02);~~

~~osThreadDef(tsk03, Task01,~~*~~osPriorityLow~~*~~, 0, 128);~~

~~Task03Handle = osThreadCreate(osThread(tsk03), (void\*)&arg03);~~

Delete the task function with its entire body

~~void Task01(void const \* argument)~~

~~{~~

~~. . .~~

~~}~~

From the function of the task by default **StartDefaultTask**  remove the body of the infinite loop, leaving there only a delay

for(;;)

{

  osDelay(1);

}

Remove this global structure

~~typedef struct struct\_arg\_t {~~

~~char str\_name[10];~~

~~uint16\_t y\_pos;~~

~~uint32\_t delay\_per;~~

~~} struct\_arg;~~

Instead, we will write a structure for transferring to the UDP-port parameters and the vertical coordinate of the output of the incoming line on the display.

**typedef struct struct\_sock\_t {**

**uint16\_t y\_pos;**

**uint16\_t port;**

**} struct\_sock;**

We remove the declaration of the structure variables, and instead of them we add new

~~struct\_arg arg01, arg02, arg03;~~

**struct\_sock sock01, sock02;**

Let's add a few elements of the queue

#define MAIL\_SIZE (uint32\_t) **5**

In the **main ()** function, we delete the initialization of the parameters

~~strcpy(arg01.str\_name,"task1");~~

~~strcpy(arg02.str\_name,"task2");~~

~~strcpy(arg03.str\_name,"task3");~~

~~arg01.y\_pos = 60;~~

~~arg02.y\_pos = 110;~~

~~arg03.y\_pos = 160;~~

~~arg01.delay\_per = 1000;~~

~~arg02.delay\_per = 677;~~

~~arg03.delay\_per = 439;~~

**Let's** change the function of displaying the lines in the **TaskStringOut** display a **little** .

We will remove the display of the number of system quanta. Today they do not need us

sprintf(str1,"%s~~%lu~~", qstruct->str~~, qstruct->tick\_count~~);

Also, move the line slightly to the left, or suddenly our information will not fit

TFT\_DisplayString(**50**, qstruct->y\_pos, (uint8\_t \*)str1, *LEFT\_MODE*);

After the **TaskStringOut** function,  **we** add a function for UDP connection tasks. I think everyone already guessed that we will have a maximum of 2 connections, since we added 2 variable structures

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

**static void udp\_thread(void \*arg)**

**{**

**for(;;)**

**{**

**osDelay(1);**

**}**

**}**

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

In the body of the default task function, **StartDefaultTask,**  remove the variable declaration

~~uint32\_t syscnt;~~

Here, we initialize our structures, which we will then serve as parameters for tasks

**sock01.port = 7;**

**sock01.y\_pos = 60;**

**sock02.port = 8;**

**sock02.y\_pos = 180;**

Create Tasks

sock02.y\_pos = 180;

**sys\_thread\_new("udp\_thread1", udp\_thread, (void\*)&sock01, DEFAULT\_THREAD\_STACKSIZE,*osPriorityNormal* );**

**sys\_thread\_new("udp\_thread2", udp\_thread, (void\*)&sock02, DEFAULT\_THREAD\_STACKSIZE,*osPriorityNormal* );**

We used a slightly different technique to create a task. It is somewhat simplified in comparison with the traditional one. We obviously do not create an intermediate structure using the osThreadDef function. Without it, it's easier.

Also, when creating tasks, we pass parameters using the structure variable.

Now let's go directly to the task function for connections **udp\_thread**

static void udp\_thread(void \*arg)

{

**struct\_out \*qstruct;**

**err\_t err, recv\_err;**

**struct netconn \*conn;**

**struct netbuf \*buf;**

**ip\_addr\_t \*addr;**

**unsigned short port;**

**struct\_sock \*arg\_sock;**

**arg\_sock = (struct\_sock\*) arg;**

**TFT\_SetTextColor(LCD\_COLOR\_BLUE);**

**conn = netconn\_new(*NETCONN\_UDP*);**

We made some initial settings, and in particular announced a structure for displaying the lines on the display with a queue, declared two variables for examining the returned arguments of some important functions, declared a connection structure, a pointer to the connection buffer, a special structure for the IP address, a variable for storing the port address, the structure variable for the task parameters. Then we got a pointer to the parameters of the task, and also led them to the type of structure we needed, declared the desired color of the output of the lines on the display, and also initialized the structure for the connection.

Continue on.

If the initialization of the structure is normal, we will link it to the local IP address and port

conn = netconn\_new(*NETCONN\_UDP*);

**if (conn!= NULL)**

**{**

**err = netconn\_bind(conn, IP\_ADDR\_ANY, arg\_sock->port);**

**}**

Continuing to write the body code of this condition, we will create in it one more condition that will check the result of calling the previous function

err = netconn\_bind(conn, IP\_ADDR\_ANY, arg\_sock->port);

**if (err == *ERR\_OK*)**

**{**

**}**

**else**

**{**

**netconn\_delete(conn);**

**}**

With a negative result, we destroy our initialized structure and fall into an endless loop.

And with a positive outcome, we end up in another infinite cycle

if (err == *ERR\_OK*)

{

**for(;;)**

**{  
  }**

}

We write the body of a given infinite cycle.

Let's try to accept the package

for(;;)

{

**recv\_err = netconn\_recv(conn, &buf);**

Here we will hang until the package comes from the client. As soon as the package comes, we will fail.

Next, we'll check that the package has started normally, that is, without errors

recv\_err = netconn\_recv(conn, &buf);

**if (recv\_err == *ERR\_OK*)**

**{**

**}**

In the body of this condition, we first take into the appropriate variables the client's IP address and port using special functions

if (recv\_err == *ERR\_OK*)

{

**addr = netbuf\_fromaddr(buf);**

**port = netbuf\_fromport(buf);**

Then create a connection with them

port = netbuf\_fromport(buf);

**netconn\_connect(conn, addr, port);**

Voobsche, the connection - this is loudly said. Just the function name is. And in fact with her only our addresses are assigned to certain fields of the **conn** structure .

Next, we display the client port and also the incoming line from the client, then we clear the memory of the structure of the queue

netconn\_connect(conn, addr, port);

**qstruct = osMailAlloc(strout\_Queue, osWaitForever);**

**qstruct->y\_pos = arg\_sock->y\_pos;**

**sprintf(qstruct->str,"%5u %-20s",port, (char\*) buf->p->payload);**

**//Пробел вместо переноса строки**

**qstruct->str[5 + strlen((char\*) buf->p->payload)] = ' ';**

**osMailPut(strout\_Queue, qstruct);**

**osMailFree(strout\_Queue, qstruct);**

We applied an interesting format of **% -20s** , with which we space the remaining elements of the string with spaces. The minus sign means alignment to the left. If it is missing, the line is aligned to the right and spaces are filled with characters to the left. Then we also replaced the space with the new line break from the client.

And when finished, we send the string back to the client as an echo and free the buffer memory

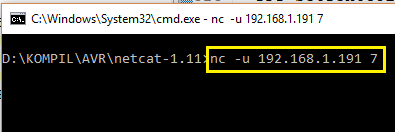
  osMailFree(strout\_Queue, qstruct);

**netconn\_send(conn,buf);**

**netbuf\_delete(buf);**

}

We'll collect the code, we'll run the controller and run the netcat program, in which we will connect to our server



We send some line to the server, trying not to exceed 20 characters

http://narodstream.ru/wp-content/uploads/2018/05/stm120img06.png

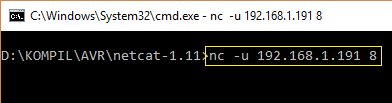
As we could see, the line returned to the client, then the server is functioning normally.

Also see the information on the display of our board

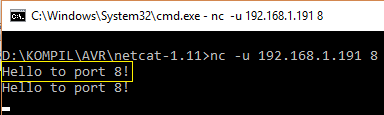


Everything has come and is displayed!

Run another session of netcat without disconnecting from port 7, and connect to it with port 8 of the server



Let's pass also here a line to the server



Here we also see that the string is returned, so the second connection on the server also functions perfectly.

Let's see the information on the display



It's also all here!

Thus, in this lesson we have today created a simple but fully functioning UDP server using the NETCON interface of the LWIP protocol stack library. It turned out that creating this server is very easy, much easier than using the RAW interface.

Thank you all for attention!