**STM Lesson 123. LAN87XX. LWIP. NETCONN. UDP. We connect three inspectors. Part 1**

Posted on [June 25, 2018](http://narodstream.ru/stm-urok-123-lan87xx-lwip-netconn-udp-soedinyaem-tri-kontrolera-chast-1/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Posted in [LAN](http://narodstream.ru/lan/) , [Programming STM32](http://narodstream.ru/rub_stm32/)- [No Comments ↓](http://narodstream.ru/stm-urok-123-lan87xx-lwip-netconn-udp-soedinyaem-tri-kontrolera-chast-1/#respond)

[10 eternal themes for paintingThe opportunity to get to the series of webinars on painting is still there! Register now!To learn morenewdirections.ru](https://an.yandex.ru/count/Aoq4PlYC8SW50Cu1CHB1ari00000ECgs7402I09Wl0Xe173mgPjnW07Cl-VExEgVyII80OF0sv4ra06gjhIyBvW1-gUrjowW0RYtjBmlg07afhMtBhW1qE_sxXd00GBO0UQClmtW0UgOlWte0Go80WEW0fBxjnMv0X6XKL3ZWl50y0AHdT6_2FW2We20W82W5803xQhkeGw80zpicVi2g0CIi0C4k0J_0UW4p0du19I10OW5b841a0Nwh0AW1U0cg0Nh8B05wo2u1S5Cm0NmcGV81R3g0T05umxW1Lhm1G6O1hxsgJUW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000Aa7-55qWkMlnKYm1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08oVtR2e0A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i6gWiGdWd62VXH000a_iTCMtK50C0BWAC5o0k0r9C1sGluKNI2vQ_5IEWBb841y0i6w0m2c0tud6No3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0E-QGTi0u1eGy00000003mFwWF-AFHqipyw-98?stat-id=3&test-tag=89060471234561&format-type=24&banner-test-tags=eyI2MzcxNTc3MjM3IjoiODkwNjA0NDE4ODI2MjQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Programming for beginnersDownload the free book and become a real programmer. Get it now:To learn moresheremetev.info](https://an.yandex.ru/count/QX8986LCUS450Dm1CHB1ari00000ECgs7402I09Wl0Xe172WmvR10O01vE3CgmU80VY6tP0fa06YakID99W1Zhs-tYEW0VwIp8qag06M_w_U8xW1XgMPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y084e0BUkAe2kG8HeL5GuuBnGF02nFtRYmtu0eA0W820e1I00__Fl_qBY0ElXgkW2fW3YSK4i0C4k0J_0UW4vGJu182p0OW5WBC1a0M8bGAW1Terg0N1BR05mIsu1RX6m0NmcGV81R3g0T05lHJW1Lhm1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XHw7i6iCL8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2A-SbG602W712W0000000F0_s0e2u0g0YNhu2i3y5OWB2AeB4CU7UIJuKG005ppGJ5jr1G302u2Z1SWBWDIJ0TaB-57eUmQmnKZe2u2p0V0B1eWCjfNUlW7e30AO3ScoP_8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_uez7dte3iuaW0?stat-id=4&test-tag=89060471234561&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY1IjoiODkwNjA0NDE4ODI2MjQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

In this lesson, we will try to connect the three controllers over the LAN using the UDP protocol.

We already have all the conditions for this, since we wrote a server supporting two simultaneous connections, for which separate tasks (threads) were organized.

Only we, of course, will not simply connect another card to our network, otherwise the lesson would turn out to be completely uninteresting. We will also use the NETCONN interface instead of RAW on clients.

And it turned out that it is not quite easy, but I think we will overcome all difficulties and everything will work for us.

Let's start with the server. The server code for us has not changed at all. However, we will still create a project to at least change the name and so that we do not confuse it with the project of the [**previous lesson**](http://narodstream.ru/stm-urok-122-lan87xx-lwip-netconn-udp-soedinyaem-dva-kontrolera/) , from which we will take the project of the server named **LAN87XX\_UDP\_SERVER**  and call it **LAN8742\_UDP\_SERVER\_NETCONN** .

Connect our PC to the PC, which we have for the server.

Run our converted project in Cube MX and, without changing anything in it, we will start the generation of the project for System Workbench, in which we will then delete it if there is a debugging configuration, and also enable the optimization level traditionally at **1** . In the file **main.c** we as usual comment out the lines unknown to the compiler relating to the video accelerator, we will collect the project and we will edit the controller.

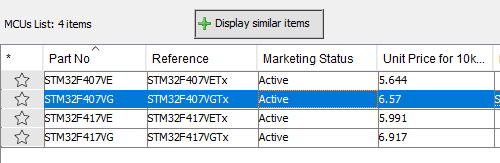
Now disconnect the server card from the PC and connect the card intended for the client. We will have the same card that we used in the previous lesson - **STM32F4-Discovery**  with the expansion board  **DIS-BB** (click on the image to enlarge)

[](http://narodstream.ru/wp-content/uploads/2018/06/stm123img00.jpg)

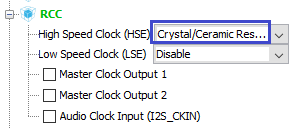
Since we connect more than two cards, we can not do with the usual cable. So we connected the switch. The yellow color on the channels of the cards indicates that the speed of our network chips is only supported by 100 mbps, not 1000.

Only for this board we will create the project anew, because not only a completely different API will be used, but the project will also work using the FREERTOS operating system.

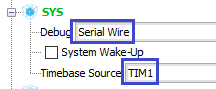
Run the Cube MX and select the controller



We connect a quartz resonator



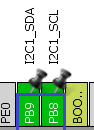
Let's choose a debugger and a system timer, since the standard Systick does not like FREERTOS



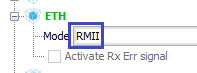
Turn on the I2C bus for the indicator

http://narodstream.ru/wp-content/uploads/2018/06/stm123img03.png

For convenience of communication on this bus, we redefine the legs



Turn on the network interface



Include the use of the operating system

http://narodstream.ru/wp-content/uploads/2018/06/stm123img06.png

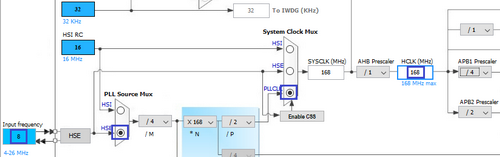
We connect the LWIP protocol stack library

http://narodstream.ru/wp-content/uploads/2018/06/stm123img07.png

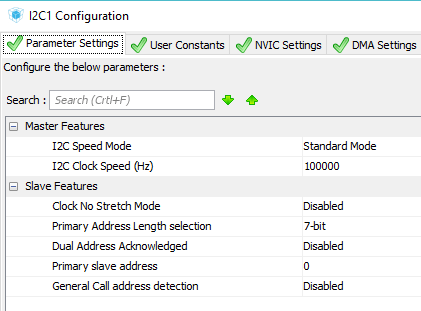
Activate the legs of the ports to which the LEDs are connected



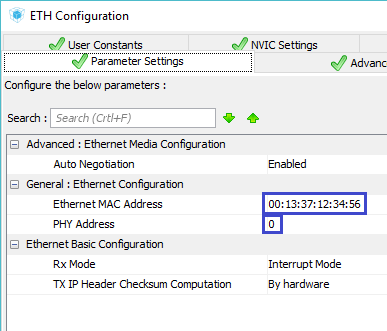
Let's go to **System Configuration** and configure everything there as follows (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2018/06/stm123img10.png)

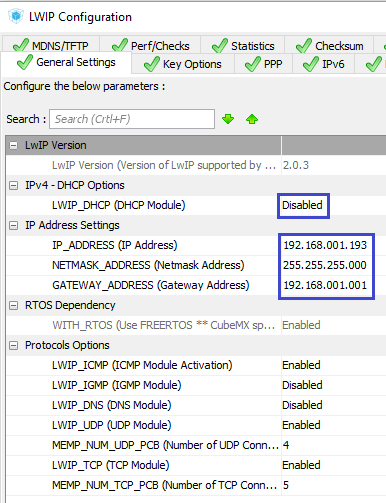
Go to **Configuration** and first of all make sure that we have correctly configured the **I2C** bus



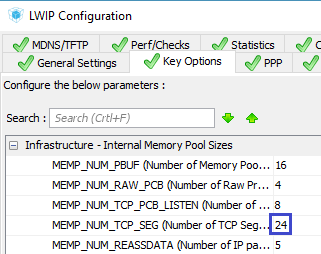
We will also configure the network interface by assigning the desired MAC address to the microchip and assigning the physical address 0 (in case of using a separate module - 1)

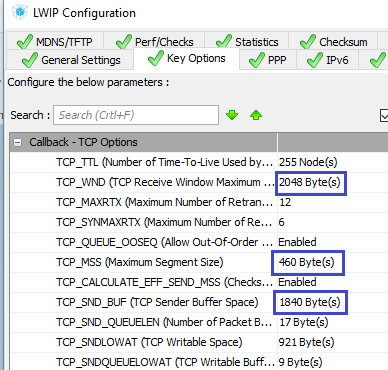


Let's pass in adjustments LWIP and at first we will adjust there the static addressing

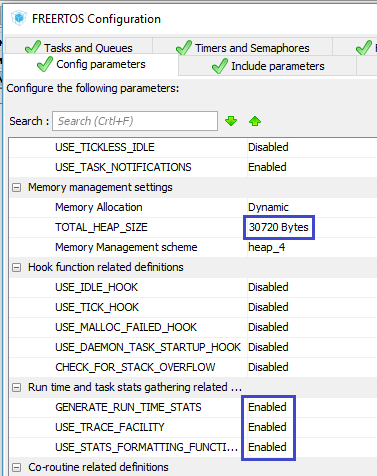


Going to the **Key Options** tab , we will also make some customary corrections here.

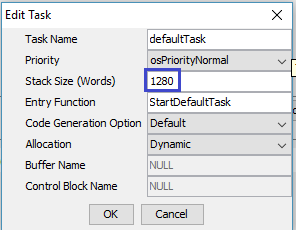




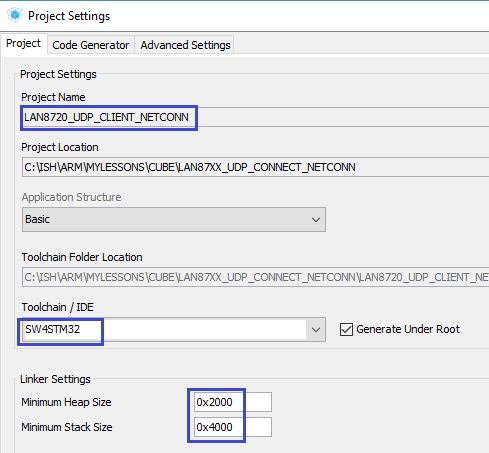
Also we will make some adjustments FREERTOS, having added a little the size of a heap and including some debugging possibilities familiar to us from the previous lessons



By clicking on the Tasks and Queues tab, we will also increase the default task stack



Enter the data in the project properties, increasing here the value of the stack and heap, and save them



Generate the project for System Workbench, run it, delete it with a debug configuration, and enable the optimization level in **1** . Then we will try to assemble the project.

If everything is fine, then **Copy the** library files for the symbol display **lcd.h**  and **lcd.c** from the last lesson project .

We will update the project tree and connect this library in the **main.c** file , along with a library for working with strings, as well as libraries for working with the protocol stack

/\* USER CODE BEGIN Includes \*/

**#include "lcd.h"**

**#include "string.h"**

**#include "lwip/opt.h"**

**#include "lwip/arch.h"**

**#include "lwip/api.h"**

/\* USER CODE END Includes \*/

In the main function, we call the display initialization, after waiting a while before to give the display to turn on

/\* USER CODE BEGIN 2 \*/

**HAL\_Delay(200);**

**LCD\_ini();**

/\* USER CODE END 2 \*/

Let's add a global string array

/\* Private variables ---------------------------------------------------------\*/

**char str1[30];**

Add the handles for the queue and the task of displaying the information on the display

char str1[30];

**osMailQId strout\_Queue;**

**osThreadId TaskStringOutHandle;**

Add structures for the queue and for the connection, and also create a variable of the connection structure type

osThreadId TaskStringOutHandle;

**typedef struct struct\_out\_t {**

**uint16\_t y\_pos;**

**char str[30];**

**} struct\_out;**

**typedef struct struct\_conn\_t {**

**uint32\_t conn;**

**uint32\_t buf;**

**} struct\_conn;**

**struct\_conn conn01;**

The size of our queue is profitable

struct\_conn conn01;

**#define MAIL\_SIZE (uint32\_t) 5**

Let's add a prototype of the function of the task of displaying information on the display

/\* Private function prototypes -----------------------------------------------\*/

**void TaskStringOut(void const \* argument);**

We initialize to main () the task of displaying information on the display

/\* USER CODE BEGIN RTOS\_THREADS \*/

/\* add threads, ... \*/

**osThreadDef(tskstrout, TaskStringOut, *osPriorityBelowNormal*, 0, 256);**

**TaskStringOutHandle = osThreadCreate(osThread(tskstrout), NULL);**

/\* USER CODE END RTOS\_THREADS \*/

Also initialize the queue

/\* USER CODE BEGIN RTOS\_QUEUES \*/

/\* add queues, ... \*/

**osMailQDef(stroutqueue, MAIL\_SIZE, struct\_out);**

**strout\_Queue = osMailCreate(osMailQ(stroutqueue), NULL);**

/\* USER CODE END RTOS\_QUEUES \*/

Add the function of the task of displaying information on the display

**/\* USER CODE BEGIN 4 \*/**

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

**void TaskStringOut(void const \* argument)**

**{**

**osEvent event;**

**struct\_out \*qstruct;**

**for(;;)**

**{**

**event = osMailGet(strout\_Queue, osWaitForever);**

**if (event.status == *osEventMail*)**

**{**

**qstruct = event.value.p;**

**sprintf(str1,"%s", qstruct->str);**

**LCD\_SetPos(0,qstruct->y\_pos);**

**LCD\_String(str1);**

**}**

**}**

**}**

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

**/\* USER CODE END 4 \*/**

The code in this function is familiar to us from the past lessons and does not need an explanation.

Add a task function for the UDP connection

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

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

**{**

**err\_t err;**

**struct netconn \*conn;**

**ip\_addr\_t DestIPaddr;**

**for(;;)**

**{**

**osDelay(1);**

**}**

**}**

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

This is so far only the work function.

In the function of the task by default  **StartDefaultTask** create this task so that it starts up

/\* USER CODE BEGIN 5 \*/

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

After the **udp\_thread** connection  **task** function, we add the callback function for receiving UDP packets

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

**void udp\_receive\_callback(struct netconn\* conn, enum netconn\_evt evt, u16\_t len)**

**{**

**}**

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

Let's create in this function some local variables, as well as pointers to structures

void udp\_receive\_callback(struct netconn\* conn, enum netconn\_evt evt, u16\_t len)

{

**struct\_out \*qstruct;**

**uint32\_t syscnt;**

**unsigned short port;**

**err\_t recv\_err;**

**struct netbuf \*buf;**

If we received a package, we will accept it

struct netbuf \*buf;

**if(evt==*NETCONN\_EVT\_RCVPLUS*)**

**{**

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

**}**

If the packet is normally received, then we know the port address from which the packet came

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

**if (recv\_err == *ERR\_OK*)**

**{**

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

**}**

Using the queue, we display the value of the 32-bit integer variable that came in the buffer and the port number of the sender of the packet, and then free the memory allocated to the structure of the queue and the structure of the receive buffer of the packet

  port = netbuf\_fromport(buf);

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

**qstruct->y\_pos = 1;**

**syscnt = \*(uint32\_t\*) buf->p->payload;**

**sprintf(qstruct->str,"%5u %7lu", port, syscnt);**

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

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

**netbuf\_delete(buf);**

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

}

Create a prototype for this function

void TaskStringOut(void const \* argument);

**void udp\_receive\_callback(struct netconn\* conn, enum netconn\_evt evt, u16\_t len);**

Continue to write the body function of the connection task  **udp\_thread** .

We will create a connection structure, declaring also the callback function for receiving packets

ip\_addr\_t DestIPaddr;

**conn = netconn\_new\_with\_callback(*NETCONN\_UDP*,udp\_receive\_callback);**

Enter the IP address of the server in the structure

conn = netconn\_new\_with\_callback(*NETCONN\_UDP*,udp\_receive\_callback);

**IP4\_ADDR(&DestIPaddr, 192, 168, 1, 191);**

Let's check if the connection structure was successful

IP4\_ADDR(&DestIPaddr, 192, 168, 1, 191);

**if (conn!= NULL)**

**{**

**}**

for(;;)

Assign a port to our client and associate it with the connection structure

if (conn!= NULL)

{

**err = netconn\_bind(conn, NULL, 1555);**

In case of successful execution of this function, we will try to connect to the server, and in the opposite case we will free up the memory allocated for the connection structure

  err = netconn\_bind(conn, NULL, 1555);

**if (err == *ERR\_OK*)**

**{**

**err = netconn\_connect(conn, &DestIPaddr, 7);**

**}**

**else**

**{**

**netconn\_delete(conn);**

**}**

}

We have not completed the function yet, but now we will add the function of the task of sending the package to the server above the given function, adding at once a number of local variables and pointers to various structures

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

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

**{**

**struct\_conn \*arg\_conn;**

**struct\_out \*qstruct;**

**struct netconn \*conn;**

**struct netbuf \*buf;**

**uint32\_t syscnt = 0;**

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

**conn = (void\*)arg\_conn->conn;**

**}**

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

We also got a pointer to the structure of our connection as a parameter.

We organize in this function a loop in which we will regularly send packets to the server

conn = (void\*)arg\_conn->conn;

**for(;;)**

**{**

**osDelay(1000);**

**}**

As usual, as part of the package, we will have a 32-bit integer value of the number of system quanta that have passed since the controller was turned on or rebooted.

Therefore, we first obtain this value

for(;;)

{

**syscnt = osKernelSysTick();**

Initialize the structure to the transfer buffer and reserve the memory for it

syscnt = osKernelSysTick();

**buf = netbuf\_new();**

**netbuf\_alloc(buf, 4);**

Send the value of the number of system quants to the buffer and send the packet to the server

netbuf\_alloc(buf, 4);

**pbuf\_take(buf->p, (void \*) &syscnt, 4);**

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

Free the buffer structure memory and switch the LED state

netconn\_send(conn,buf);

**netbuf\_delete(buf);**

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

We display the value that we sent to the server and free the memory for the queue

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

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

**qstruct->y\_pos = 0;**

**sprintf(qstruct->str,"%10lu", syscnt);**

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

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

osDelay(1000);

Well, in principle, it remains for us to create this task in the connection task **udp\_thread**

  err = netconn\_connect(conn, &DestIPaddr, 7);

**if (err == *ERR\_OK*)**

**{**

**conn01.conn = conn;**

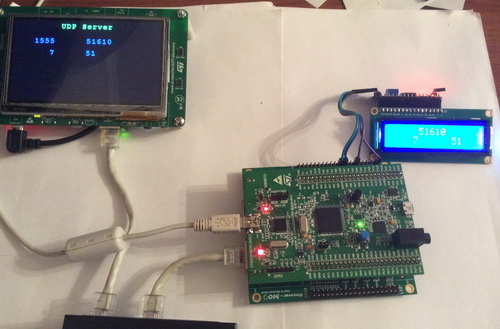
**sys\_thread\_new("send\_thread1", send\_thread, (void\*)&conn01, DEFAULT\_THREAD\_STACKSIZE,*osPriorityNormal* );**

**}**

}

else

We will collect the code, we'll tell the controller and make sure that packets from us to the server go away, and then come back already with the value divided by 1000 (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2018/06/stm123img19.jpg)

Everything works fine!

In the [**next part of**](http://narodstream.ru/stm-urok-123-lan87xx-lwip-netconn-udp-soedinyaem-tri-kontrolera-chast-2/) our lesson we will create a project for the second client and check the joint work of all three nodes of our network.

**STM Lesson 123. LAN87XX. LWIP. NETCONN. UDP. We connect three inspectors. Part 2**

Posted on [June 25, 2018](http://narodstream.ru/stm-urok-123-lan87xx-lwip-netconn-udp-soedinyaem-tri-kontrolera-chast-2/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Posted in [LAN](http://narodstream.ru/lan/) , [Programming STM32](http://narodstream.ru/rub_stm32/)- [No Comments ↓](http://narodstream.ru/stm-urok-123-lan87xx-lwip-netconn-udp-soedinyaem-tri-kontrolera-chast-2/#respond)

[Fundamentals of Programming LanguagesDownload the free book and become a real programmer. Get it now:To learn moresheremetev.info](https://an.yandex.ru/count/JzfpZBjXJx850Du1CKp1ari00000ECgs7402I09Wl0Xe172WmvR10O01ZSt-9uW1jeRTa2cG0VYKhOyac07okhxU8w01lfFCZIIe0PB-hzuZk062f9dq5S010jW1bh2j4E01vfJJ3kW15lW1fkdUlW680WEW0jwugWAv0f8ahj3xOhcGy0ANe9YR2lW2We20W82m4u03YVo9_Gs80uR2xf84c0FoqmMe0nAm0mIu1Fy1w0IR1FW4pDi1Y0NCsm6G1V6K0g05y46e1Ueti0NgDxW5iaR01V2P1yW5iEe1q0MF6U05Fl050PW6WD2iymwW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000Aa7-56Suj6znKYm1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08hvoL0O0A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i6gWiGCXkt9lXH00275KDGMtK50C0BWAC5o0k0r9C1sGluKPpYqRt5IEWBpDi1y0i6Y0ocwTw-0UWC0fWD-9nbyWq0-Wq0WWu0YGu00000003mFv0Em8Gzc0x6rypKbOdWkXEW3i24FR0E0Q4F00000000y3-e3_YZeV6Fkk_YI000?stat-id=3&test-tag=85212180537345&format-type=24&banner-test-tags=eyI0ODQ3OTM2ODI4IjoiODUyMTIxNTExODU0MDgifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[PHP Programming Courses2 lectures a week, personal tutor, final project site, certificateTo learn morehtmlacademy.ru](https://an.yandex.ru/count/I1QBK_p4o6m50E41CKt1ari00000ECgs7402I09Wl0Xe173MZO-53e01oexaomU80V6TcjGqa06KnkkWBvW1wBlk-IsW0Oh6ww0lg06Gk-xvBRW1w9tytXd00GBO0OgrwnJW0TIzXWVe0QO2-07czzw-0OW21A02hCMi3xa2aYIkqFjYkP3m0i6tc-8B-0A2W820WB0JW0FdwUV-2eW3qCQsq0QO0zUL0h031BW4_m7e1F01-0Joq0I81VBG1905ikW7e0NliG6e1TwM0R05tfO1k0Mh_W701Pkq6iW5iDi6q0NgG-05Fl050PW6mA_S_GsW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000Aa7-57KQVAznKYm1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08rgBM1u0A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i8gWiG3t4T4VXH003aDqjGMtK50C0BWAC5o0k0r9C1sGluKTHfyht5IEWByj04y0i6Y0pczzw-0UWC0fWDoR9dyWq0-Wq0WWu0YGu00000003mFv0Em8Gzc0x6rypKbOdWkXEW3gAa3R0E0Q4F00000000y3-e3_YZiUcOlk_YI000?stat-id=4&test-tag=85212180537345&format-type=24&banner-test-tags=eyI2MzQyNjY4NjgyIjoiODUyMTIxNTExODU0MDgifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)18+

In the [**previous part of**](http://narodstream.ru/stm-urok-123-lan87xx-lwip-netconn-udp-soedinyaem-tri-kontrolera-chast-1/) this lesson, we set up a project for the server, and also created and wrote a project for the first client and checked the work of the connection in practice.

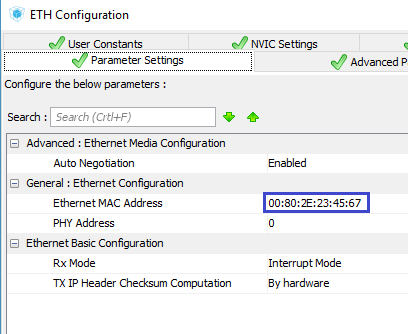
Now we'll work with the second client. As it will be another board **STM32F746G-DISCOVERY** .

Disconnect the board of our first client from the PC and connect to an independent source, and connect the 32F746G-DISCOVERY card to the PC, well. respectively, also to our network (click on the image to enlarge the image)

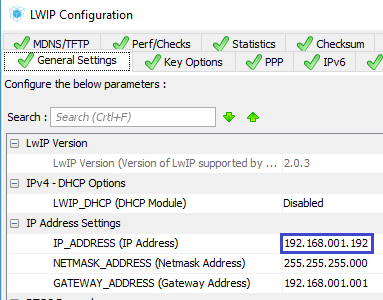
[](http://narodstream.ru/wp-content/uploads/2018/06/stm123img20.jpg)

We will create a project for the second client from the server project, since the boards are the same. And we will assign a name to this project **LAN8742\_UDP\_CLIENT\_NETCONN** . From the project for the first client, the name of this project differs in the cipher in the labeling of the chip.

Run our project in Cube MX and go to **Configuration** . First, configure the MAC address in the network interface configuration. These addresses can not be the same in the same network



Now we'll go to LWIP and change the network address a bit



Generate the project, open it in the System Workbench, set the optimization level to **1** , delete it with debug settings. Also comment out the unknown to the compiler two lines concerning DMA2D, we will try to assemble the project.

If everything is fine, then we'll start to slowly transform the server into a client.

First of all - the design of the screen

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

Remove the global structure destined for the socket

~~typedef struct struct\_sock\_t {~~

~~uint16\_t y\_pos;~~

~~uint16\_t port;~~

~~} struct\_sock;~~

Also delete the variables of this structure

~~struct\_sock sock01, sock02;~~

Instead, add a structure for the connection and create a variable of this structure

  } struct\_out;

**typedef struct struct\_conn\_t {**

**uint32\_t conn;**

**uint32\_t buf;**

**} struct\_conn;**

**struct\_conn conn01;**

From the function of the **udp\_thread** connection **task,**  delete the entire body for the time being.

Add over this function the transfer function of the package, completely analogous to the one we wrote in the project for the first client

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

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

**{**

**struct\_conn \*arg\_conn;**

**struct\_out \*qstruct;**

**struct netconn \*conn;**

**struct netbuf \*buf;**

**uint32\_t syscnt = 0;**

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

**conn = (void\*)arg\_conn->conn;**

**for(;;)**

**{**

**syscnt = osKernelSysTick();**

**buf = netbuf\_new();**

**netbuf\_alloc(buf, 4);**

**pbuf\_take(buf->p, (void \*) &syscnt, 4);**

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

**netbuf\_delete(buf);**

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

**qstruct->y\_pos = 60;**

**sprintf(qstruct->str,"%16lu",syscnt);**

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

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

**osDelay(1000);**

**}**

**}**

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

Also below the function **udp\_thread** we add a callback function that reacts to receiving packets, which is also completely similar to the one we used in the first client

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

**void udp\_receive\_callback(struct netconn\* conn, enum netconn\_evt evt, u16\_t len)**

**{**

**struct\_out \*qstruct;**

**uint32\_t syscnt;**

**unsigned short port;**

**err\_t recv\_err;**

**struct netbuf \*buf;**

**if(evt==*NETCONN\_EVT\_RCVPLUS*)**

**{**

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

**if (recv\_err == *ERR\_OK*)**

**{**

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

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

**qstruct->y\_pos = 100;**

**syscnt = \*(uint32\_t\*) buf->p->payload;**

**sprintf(qstruct->str,"%5u %7lu", port, syscnt);**

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

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

**netbuf\_delete(buf);**

**}**

**}**

**}**

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

Create a prototype for this function

void TaskStringOut(void const \* argument);

**void udp\_receive\_callback(struct netconn\* conn, enum netconn\_evt evt, u16\_t len);**

Now add the body of our function, also similar to the body of the first client's project function

static void udp\_thread(void \*arg)

{

**err\_t err;**

**struct netconn \*conn;**

**ip\_addr\_t DestIPaddr;**

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

**conn = netconn\_new\_with\_callback(*NETCONN\_UDP*,udp\_receive\_callback);**

**IP4\_ADDR(&DestIPaddr, 192, 168, 1, 191);**

**if (conn!= NULL)**

**{**

**err = netconn\_bind(conn, NULL, 1551);**

**if (err == *ERR\_OK*)**

**{**

**err = netconn\_connect(conn, &DestIPaddr, 8);**

**if (err == *ERR\_OK*)**

**{**

**conn01.conn = conn;**

**sys\_thread\_new("send\_thread1", send\_thread, (void\*)&conn01, DEFAULT\_THREAD\_STACKSIZE, *osPriorityNormal* );**

**}**

**}**

**else**

**{**

**netconn\_delete(conn);**

**}**

**}**

**for(;;)**

**{**

**osDelay(1);**

**}**

}

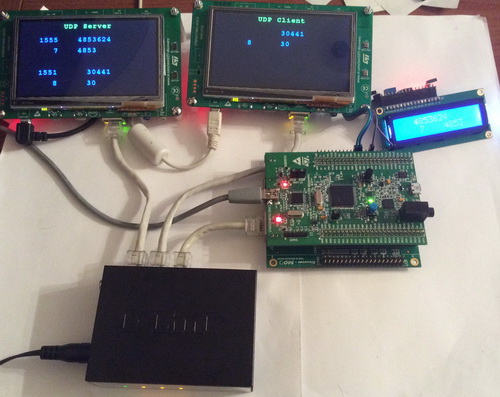
From the default task function, delete all user code to an infinite loop and add the creation of a task for the connection

/\* USER CODE BEGIN 5 \*/

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

/\* Infinite loop \*/

We will collect the code, we will sew the controller and see the result (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2018/06/stm123img23.jpg)

Everything at us is perfectly passed and accepted. Two UDP connections work independently of each other.

Thus, today we created a UDP client using the FreeRTOS operating system and the NETCONN interface, and also tested it on two different boards, and in the work of both clients simultaneously.

Thank you all for attention!