**STM Lesson 122. LAN87XX. LWIP. NETCONN. UDP. We connect two inspectors**

Posted on [June 19, 2018](http://narodstream.ru/stm-urok-122-lan87xx-lwip-netconn-udp-soedinyaem-dva-kontrolera/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Posted in [FreeRTOS](http://narodstream.ru/freertos/) , [Programming STM32](http://narodstream.ru/rub_stm32/)- [No Comments ↓](http://narodstream.ru/stm-urok-122-lan87xx-lwip-netconn-udp-soedinyaem-dva-kontrolera/#respond)

[Buy robot constructorTeach your child to program. 6900 for a set of shares. OrderTo learn moredorado-uno.ru](https://an.yandex.ru/count/RlJTz_NjANm50De1CSt0ari00000ECgs7402I09Wl0Xe172AffoG1e01xCtW6OW1rBMagJMG0VZAnCelc07gZ-3LBg01zih4oY-e0UYEuDKkk07st_Rr6S010jW1ieMB5U01qBto4UW1OFW1elhUlW680WEW0g2-sHQv0cbtYt4JgPLQy0BuyCJv1_W2We20W82G4u03jv_Romc80yAK-Vq5c0Eix0Ae0nAm0mJe1De1-0Ivj0E81Rcq0v05vBW3e0NuWW6e1Vjki0NxRhW5Y7301R3x3CW5yAK3q0NtCU05Ml050PW6tgcE6Q06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00071n0000gGVuKLHdQf_5IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0YGtwmBW0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mQg2n3w550Y-54004StgKXRTGK0m0k0emN82u3Kam7P2_XHL6TgdyL8w0kvj0Fm2mQ83ABwthu1w0m2c0tud6No3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Em8Gzi0u1eGy00000003mFwWF-AFXZ9djv-98?stat-id=3&test-tag=84112668909569&format-type=24&banner-test-tags=eyI2Mzg2Mzg5NjkyIjoiODQxMTI2Mzk1NTc2MzIifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

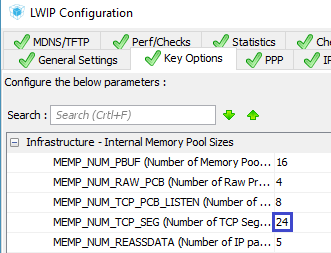
[Programming LessonsLearn all the in-house cooking **programming** training ! Get the book on the e-mail!To learn moresheremetev.info](https://an.yandex.ru/count/9Fm686kGlN850Cq1CSt0ari00000ECgs7402I09Wl0Xe172WmvR10O01sE3CgmU80QE8tP0fa06cakID99W1eiM-tYEW0SAapOqag07iWR3U8xW1tAUPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y084e0BUkAe2kG9fTujn4wcLMl02vDAGjGtu0eA0W820a1EO0wKVi0C4k0J_0UW4rmFu1DJC0OW5rCm1a0Nnsm6W1RCzg0M4DB05X3Iu1VCtm0NmcGV81R3g0T05jHVW1J_m1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XHv8U9dyL8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2A-SbG602W712W0000000F0_s0e2u0g0YNhu2i3y5OWB2AeB48gFYKduKG00SF-mI5jr1G302u2Z1SWBWDIJ0TaB-57aXucVnKZe2zJC0V0B1eWCjfNUlW7e30AO3ScoP_8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_uew6YaF7duaW0?stat-id=4&test-tag=84112668909569&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY3IjoiODQxMTI2Mzk1NTc2MzIifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

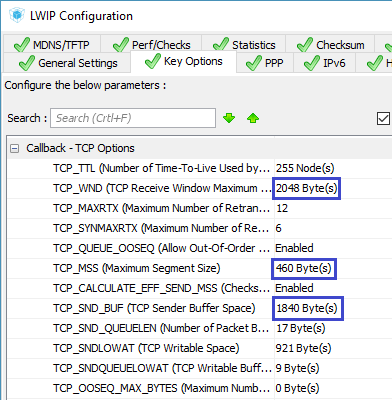
We continue to work with the **LWIP** protocol **stack** with the **NETCONN** interface and now we will try to connect another card to our motherboard - **STM32F4-Discovery** together with the expansion board **DIS-BB** . And on this board will continue to be used as a client interface **RAW** . In other words, we will try to combine two lessons at once - [**lesson 118**](http://narodstream.ru/stm-urok-118-lan8720-lwip-raw-udp-client/) and [**lesson 120**](http://narodstream.ru/stm-urok-120-lan8742f-lwip-netconn-udp-server/) . Thus, we organize the exchange of information without using a PC.

Let's start with the server.

We will create the project from the project [**lesson 120**](http://narodstream.ru/stm-urok-120-lan8742f-lwip-netconn-udp-server/) **LAN8742\_UDP\_SERVER\_NETCONN** and call it **LAN87XX\_UDP\_SERVER** .

Open the project in the Cube MX and adjust the **LWIP** settings in the **Key Options** section a bit so that they are the same with the settings in the client





Save the settings, generate the project for System Workbench, open it there, remove the debugging settings if available, and also change the optimization level to 1.

Open the file **main.c** , comment out the unknown to the compiler lines for setting up the video accelerator and try to assemble the project.

If everything is fine, then we will proceed to some edits of the project.

We will transfer 4-byte values ​​to our server, so take it from the buffer, adding first a local variable to the task function of our sockets **udp\_thread**

struct\_sock \*arg\_sock;

**uint32\_t syscnt = 0;**

Then, in the body of the same function, we will take this value from the received buffer

qstruct->y\_pos = arg\_sock->y\_pos;

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

We display this value, as well as the sender's port, correcting the corresponding code further.

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

**sprintf(qstruct->str,"%5u %10lu",port, syscnt);**

Divide it by 1000 to send it back to the client. Simply transfer the same thing - it somehow looks more like an echo, and if we change the value and give it an already changed one, it will be like a separate transfer of the newly formed buffer

sprintf(qstruct->str,"%5u %10lu",port, syscnt);

**syscnt /= 1000;**

Remove the insertion of a space, we do not need it

~~//Пробел вместо переноса строки~~

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

Send the modified 32-bit value back to the buffer

syscnt /= 1000;

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

We display after sending the data to the client on the display the sent value with the port number of our server

  netbuf\_delete(buf);

**osDelay(1);**

**qstruct->y\_pos = arg\_sock->y\_pos + 40;**

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

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

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

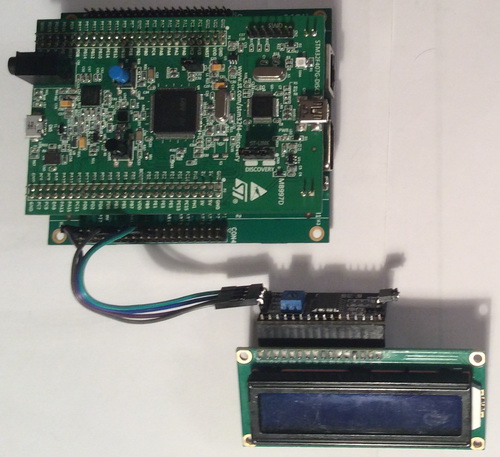
}

I think that you noticed that we delayed before sending information to the display. It is necessary in order to give a less priority to the task of displaying the display to the end, since we have just sent it to the queue just now. One system quantum for this is quite enough. Otherwise, without this delay, we will see only the bottom line.

We will collect the code and let's say the controller.

With the server like everything, go to the client.

We will connect to our fourth Discovery a small character display LCD1602, there also see information about the contents of the sent and received packets



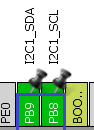
We connect our motherboards with a network cable, connect the server card from an independent power source, and connect the customer's card to the PC.

We will make the project from the project of the [**lesson 118**](http://narodstream.ru/stm-urok-118-lan8720-lwip-raw-udp-client/) **LAN8720\_UDP\_CLIENT\_RAW** and give it the name **LAN87XX\_UDP\_CLIENT** .

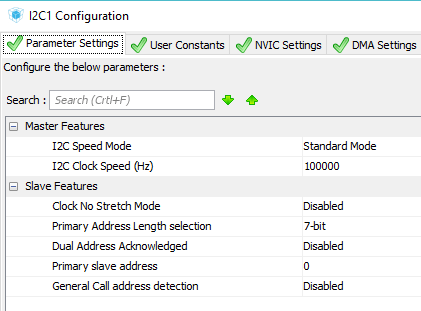
Open the project in the Cube MX and first turn on the I2C to connect the display

http://narodstream.ru/wp-content/uploads/2018/05/stm122img03.png

Override the legs for easy connection to the DIS-BB board



Go to **Configuration** and make sure that this bus is properly configured



Save the settings, generate the project for System Workbench, open it there, remove the debugging settings if available, and also change the optimization level to 1.

Open the **main.c** file and try to build the project.

If everything is normal, then we will start to edit this project as well.

From the project [**lesson 115**](http://narodstream.ru/stm-urok-115-nrf24l01-neskolko-priemnikov-chast-1/) **NRF24\_RX\_02\_LCD** we take the files for display **lcd.c** and **lcd.h** and copy them into the appropriate folders of our new project.

We will update the project tree.

We connect the library in **main.c**

#include "net.h"

**#include "lcd.h"**

Also, in the **main ()** function, we initialize the display, wait a short time before it, and also start the timer to the last place, so that we do not call the packet transfer function before the network connection is initialized. It will be in main () here such code

/\* USER CODE BEGIN 2 \*/

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

**HAL\_Delay(200);**

**LCD\_ini();**

**HAL\_TIM\_Base\_Start\_IT(&htim2);**

/\* USER CODE END 2 \*/

In the **lcd.c** file **,** change the bus number I2C

extern I2C\_HandleTypeDef hi2c**1**;

In function **LCD\_WriteByteI2CLCD**  too

HAL\_I2C\_Master\_Transmit(&hi2c**1**,(uint16\_t) 0x4E,buf,1,1000);

In the file **net.h** also connect the library for the display

#include "lwip/udp.h"

**#include "lcd.h"**

Go to the file **net.c**  and the **udp\_client\_connect** function **,** change the IP address of the server

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

Also change the port number

err= udp\_connect(upcb, &DestIPaddr, **7**);

Let's go to the function **udp\_client\_send** and add two local variables

struct pbuf \*p;

**uint32\_t gt;**

**uint16\_t port;**

Remove the conversion function in a string

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

Find out our local port

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

**port = upcb->local\_port;**

We learn the number of system ticks that have passed

if (p != NULL)

{

**gt = HAL\_GetTick();**

Put them in the send buffer of the package, changing the arguments in the corresponding called function

pbuf\_take(p, **(void \*) &gt, 4**);

Display this number on the display

pbuf\_take(p, (void \*) &gt, 4);

**sprintf(str1,"%10lu", gt);**

**LCD\_SetPos(0,0);**

**LCD\_String(str1);**

Let's pass to the function of receiving the package  **udp\_receive\_callback** and remove the formation of the string from there

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

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

Form it in a new way and send it to the display

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

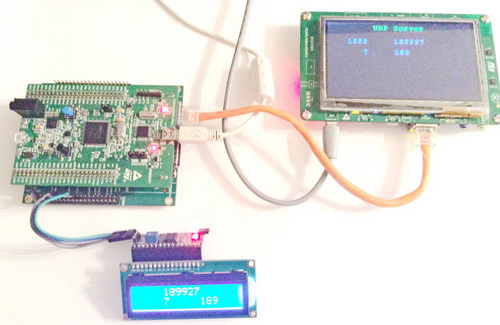
{

**sprintf(str1,"%5u %7lu", port, \*(uint32\_t\*) p->payload);**

**LCD\_SetPos(0,1);**

**LCD\_String(str1);**

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

[](http://narodstream.ru/wp-content/uploads/2018/05/stm122img06.jpg)

Everything at us is perfectly transferred.

Thus, this lesson allowed us to connect a client and a server working on the UDP protocol through two debugging cards, on which the LWIP protocol stack with various interfaces was applied. This in the future can allow a decent speed to transfer any data from the controller to the controller.

Thank you all for attention!

s