**STM Lesson 100. LAN8720. LWIP. TCP. We connect two controllers. Part 1**

Posted on [December 14, 2017](http://narodstream.ru/stm-urok-100-lan8720-lwip-tcp-soedinyaem-dva-kontrollera-chast-1/)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-100-lan8720-lwip-tcp-soedinyaem-dva-kontrollera-chast-1/#respond)

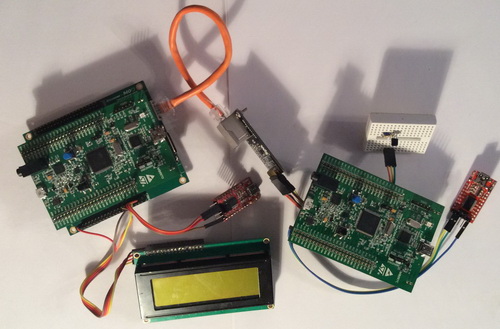
[Programming LessonsLearn all the in-house cooking **programming** training ! Get the book on the e-mail!To learn moresheremetev.info](https://an.yandex.ru/count/8eT7DjcrolO50Cq1COsyari00000ECgs7402I09Wl0Xe172WmvR10O01sE3CgmU80QE8tP0fa06cakID99W1eiM-tYEW0SAapOqag07iWR3U8xW1tAUPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y083e0BUkAe2kG8Lbl1JiG3dxV02wSB3jGdu0eA0W820q1EO0yy8g0CIi0C4k0J_0UW4qWJu1AzhY0MlQv05myW1e0MR8AW5fXkm1QORk0NzCi05y9a7o0MmwW7G1QGCu0K_y0K1c0Q0qApp3g06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00071n0000gGVuKP3zB9p4IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0Yld9K1W0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mQg2n2l6qGY-5400C_SsGTRTGK0m0k0emN82u3Kam7P2_XHaFqidCH8w0klQ_0B1eWCjfNUlW7e30AO3PMCQF8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_ueu7FnuMduaW0?stat-id=3&test-tag=86311692165121&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY3IjoiODYzMTE2NjI4MTMxODQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Unclear errors of parents,for which the child pays. Free test for parentsTo learn moreyurkovskaya.com](https://an.yandex.ru/count/D9lkdoGGFAS50Dm1COwyari00000ECgs7402I09Wl0Xe172Civx21u01cfZa1OW1cFwAt3IG0RIeoAGlc07mnlx_BQ01jgZ8f2-e0PJ6_lyjk06m-jNW6S010jW1_AsU2-01vehT2kW1TlW10OW21A02sA_P5Ra25PRmKx40v-t80gEaokYRwD2EPV02rR24uWBu0eA0W820q1E00_YDr4U80zEfbQS7i0C4k0J_0UW4jGBu1EZD0eW5wCq2a0M8_GAW1P9ag0NlLB05xrIu1V1Wm0MLpGd81S2u0j05c2RW1Gdm1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60019S740002f1_XHbBX9dCH8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2EpNx0g02W712W0000000F0_s0e2u0g0YNhu2i3y5OWB2AeB42GlQmRuKG00RWlX1rjr1G302u2Z1SWBWDIJ0TaB-56Kk4cSn4Ze2-ZD0l0B1eWChgZUlW7e30AO3VozPl8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_ue-6_Z8cduaW0?stat-id=4&test-tag=86311692165121&format-type=24&banner-test-tags=eyI2MzQ2NTc0MzYyIjoiODYzMTE2NjI4MTMxODQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)18+

We continue to work with the **LAN8720** LAN physical layer **chip** , as well as the **LWIP** protocol stack library .

We have already examined the operation of the TCP client and TCP server on this chip, using the **RAW API** , and also using the **STM32F4-Discovery** debugging card . Now we all, I think, will be interested in our server and the client to connect, that is, to use only controllers and LAN chips. This is what we are going to do today. We will completely disconnect from the PC as a result, that is, we will not even use USART. On the server, we will use the **LCD2004** symbol display on the **HD44780** controller to display information , which we are very familiar with from previous lessons. We connect it with the help of the adapter also known to us to the bus I2C. And we connect the temperature sensor **DS18B20** to the customerand we will send its temperature readings through the network to the server. Here such plans. At first I planned to send just some lines and take them also on the server and monitor using USART, but it was too simple and uninteresting. Therefore, I here and so decided to diversify the lesson and thereby bring it closer, so to speak, to real life conditions.

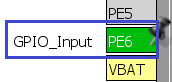
Our whole scheme in practice will look something like this:

[](http://narodstream.ru/wp-content/uploads/2017/11/stm100img00.jpeg)

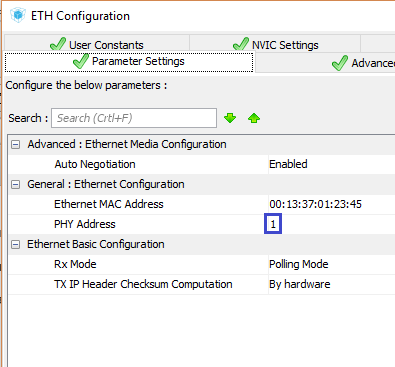
Only the circuit is not yet connected to the boards, so that the extra wires do not interfere, and so, if you connect, in principle, everything is already working and the temperature is transferred. Therefore, I will now begin to share my development with you. Only once we connect our boards will not, we take out this orange wire and bring to the mind first the client, and then the server.

Let's start with the client. Let's take the project from [**lesson 96**](http://narodstream.ru/stm-urok-96-lan8720-lwip-tcp-client-chast-1/) called **LAN8720\_TCP\_CLIENT** and rename it our cunning way to **LAN8720\_TCP\_CLIENT\_DS18B20** .

Open our project in **Cube MX** and turn on the foot of **PE6** port , we will connect our temperature sensor to it.



Go to **Configuration** in  **ETH** and check that there is a correct **PHI** address . At me for example on the client the unit from WaveShare, means should be **1**



Next, we generate the project, open it in the System Workbench and start working with it. As always, we will remove all debugger settings and set the optimization level to 1.

The LAN module is connected to the PC, since it is necessary to configure it. That is, in the role of the server while still there will be a PC, we will have time to connect. USART will also connect.

As always, run WireShark, the terminal program CoolTerm and command a line in which we try to ping our client.

If everything is normal, then back to the project.

From the project [**lesson 94**](http://narodstream.ru/stm-urok-94-ds18b20-podklyuchaem-neskolko-datchikov-na-provod-chast-1/) on the temperature sensor, take and copy the corresponding folders of our new project files **ds18b20.h** and **ds18b20.c** . We will update the project tree and fix the connected library in the file   **ds18b20.h** , since we have 4 series of controllers

#include "stm32f**4**xx\_hal.h"

**Go to the main.h** file and connect our library, as well as the library for working with strings

/\* USER CODE BEGIN Includes \*/

**#include "ds18b20.h"**

**#include <string.h>**

/\* USER CODE END Includes \*/

And in the **main.c** file in the **main ()** function, we call the initialization function of our sensor and port, and start the timer below all initializations. The code will now become

/\* USER CODE BEGIN 2 \*/

**net\_ini();**

**port\_init();**

**ds18b20\_init(SKIP\_ROM);**

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

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

/\* USER CODE END 2 \*/

Also in our main.c we will create and connect some global variables and arrays

extern char str[30];

**extern char str1[100];**

**uint8\_t Dev\_ID[8][8]={0};**

**uint8\_t Dev\_Cnt;**

**uint8\_t dt[8];**

**uint16\_t raw\_temper;**

**float temper;**

**char c;**

**volatile uint32\_t Tim2Cnt=0;**

**extern struct tcp\_pcb \*client\_pcb;**

/\* USER CODE END PV \*/

In an infinite loop, we will not poll the sensor readings, there is nothing to slow down the processes, we will do all this later in the procedure for processing interrupts from the timer, and in an infinite loop, we call the two functions into a call to one function

/\* USER CODE BEGIN 3 \*/

**MX\_LWIP\_Process();**

}

This was done not only to reduce the text of the code, but also for the fact that in the file where the body of this function lies, you can add your code, since there are appropriate comments for that.

Now we just need to go to the **ds18b20.c** file and make the corresponding changes there to the low-level code that uses the **CMSIS** library , since for the **STM32F407** controller such registers that we used there are not applicable because it simply does not. He has several other registers. I will not show you the Reference Manual, I think you've read it 100 times already. We will simply correct the same procedures using existing registers.

Let's start with the DelayMicro function and fix the line in which the divider I calculated experimentally

micros \*= SystemCoreClock / **4940000**;

Next, we fix the code in the function **port\_init** , at the same time with registers correcting the foot of the port, because it also changed

void port\_init(void)

{

**HAL\_GPIO\_DeInit(GPIOE, GPIO\_PIN\_6);**

**GPIOE->MODER &= ~GPIO\_MODER\_MODE6\_1;**

**GPIOE->MODER |= GPIO\_MODER\_MODE6\_0;**

**GPIOE->OSPEEDR |= GPIO\_OSPEEDR\_OSPEED6\_1;**

**GPIOE->OSPEEDR |= GPIO\_OSPEEDR\_OSPEED6\_0;**

**GPIOE->OTYPER |= GPIO\_OTYPER\_OT6;**

}

Next, we fix the code in the function **ds18b20\_Reset**

**GPIOE**->ODR &= ~GPIO\_ODR\_**OD6**;//низкий уровень

DelayMicro(485);//задержка как минимум на 480 микросекунд

**GPIOE**->ODR |= GPIO\_ODR\_**OD6**;//высокий уровень

DelayMicro(65);//задержка как минимум на 60 микросекунд

status = **GPIOE**->IDR & GPIO\_IDR\_**ID6**;//проверяем уровень

Then the function **ds18b20\_ReadBit**

**GPIOE**->ODR &= ~GPIO\_ODR\_**OD6**;//низкий уровень

DelayMicro(2);

**GPIOE**->ODR |= GPIO\_ODR\_**OD6**;//высокий уровень

DelayMicro(13);

bit = (**GPIOE**->IDR & GPIO\_IDR\_**ID6** ? 1 : 0);//проверяем уровень

And the function **ds18b20\_WriteBit**

**GPIOE**->ODR &= ~GPIO\_ODR\_OD6;

DelayMicro(bit ? 3 : 65);

**GPIOE**->ODR |= GPIO\_ODR\_OD6;

Now the project is likely to be assembled. Let's try to assemble the project, if everything is normal, then move on.

Let's **go to ds18b20.c** and fix the **ds18b20\_Convert** function **code** , removing all unnecessary ones.   
Otherwise, we have a negative temperature displayed incorrectly. I just did not have the opportunity   
to check, but now, when it's freezing in the street, such an opportunity has appeared. I just connected the sensor that   
was on the wire to the breadboard and threw it out the window, closing the window on the wire, there is an elastic band in the window,   
so nothing will happen to the wire. The code in the function will now be this

float ds18b20\_Convert(**int16\_t** dt)

{

  float t;

  t = **(float)dt / 16.0f;**

  return t;

}

In the prototype, also change the type of the input parameter.

Let's create a prototype for the function of sending a string to the network in the header file **net.h**

void UART6\_RxCpltCallback(void);

**void sendstring(char\* buf\_str);**

We now need to give the command in time to the sensor, so that he begins to read the temperature, then give the command on time, so that the sensor gives us this testimony. For this we go to the **main.c** file and add an interrupt handler from the timer

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

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

**{**

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

**{**

**Tim2Cnt++;**

**if(Tim2Cnt>10)**

**{**

**}**

**HAL\_GPIO\_TogglePin(GPIOD, GPIO\_PIN\_13);**

**}**

**}**

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

/\* USER CODE END 4 \*/

The timer, as we remember it works with a period of about half a second, so we first wait 5 seconds, I mean we'll give some time for our nodes to connect safely, and then we'll get into the body of the condition, the body of which we now start write

if(Tim2Cnt>10)

{

**if(client\_pcb->state == ESTABLISHED)**

**{**

**if(Tim2Cnt%4==0)**

**{**

**ds18b20\_MeasureTemperCmd(SKIP\_ROM, 0);**

**}**

**else if(Tim2Cnt%4==2)**

**{**

**ds18b20\_ReadStratcpad(SKIP\_ROM, dt, 0);**

**raw\_temper = ((int16\_t)dt[1]<<8)|dt[0];**

**temper = ds18b20\_Convert(raw\_temper);**

**if(temper>0) sprintf(str1,"+%.2f rn", temper);**

**else sprintf(str1,"%.2f rn", temper);**

**HAL\_UART\_Transmit(&huart6,(uint8\_t\*)str1,strlen(str1),0x1000);**

**sendstring(str1);**

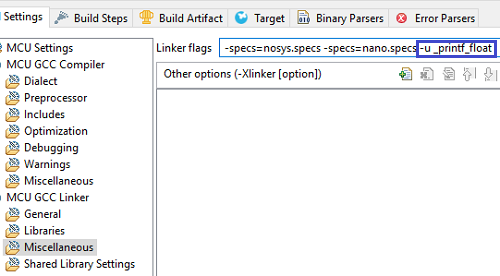
**}**

**}**

}

It turned out all very simple. We divide by 4 modulo, and when the result is 0, then we tell the sensor to measure the temperature, and when 2 - then we give it. And then we display it in USART in the terminal program and, most importantly, send it to our server.

Also, in order for the temperature to be correctly considered, we need to include a floating-point operation in the project. To do this, go to the properties of the project in the section **C / C ++ Build -> Settings -> MCU GCC Linker -> Miscellaneous -> Linker flags**  in this line, add " **-u \_printf\_float** " (do not forget about the space)



Only before these readings go to the server, we need to connect with it. So go to the file **net.c** and first add there another global variable, by which we will monitor the state of our connection

\_\_IO uint32\_t message\_count=0;

**\_\_IO uint8\_t net\_stat=0;**

Also add to the structure another kind of connection state

*ES\_NOT\_CONNECTED* = 0,

***ES\_CONNECTING*,**

*ES\_CONNECTED*,

Add the code to the **net\_ini** function **so** that the client immediately connects to the server

usartprop.is\_text=0;

**ipaddr\_dest[0] = 192;**

**ipaddr\_dest[1] = 168;**

**ipaddr\_dest[2] = 1;**

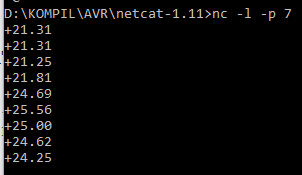
**ipaddr\_dest[3] = 87;**

**port\_dest = 7;**

**net\_stat = *ES\_CONNECTING*; //Инициализируем попытку соединения**

**tcp\_client\_connect();**

Now let's collect the code, let's tell our controller and see if our temperature comes to the server (while the PC). Before this, do not forget to start netcat and get it to listen on port 7



Excellent! The temperature began to be transmitted to the server.

Also I have one more idea. If suddenly something happens to the server and it becomes unavailable, and it makes no sense for us to keep the connection open and send out packages somewhere, then we will try to track this incident and disconnect from it. Well, rather, the correct separation will not happen, since it is not available, but at least we will do all our necessary procedures, free up memory under the structure, etc. and the packages we have nowhere to go will not. And the temperature will also stop reading, because we will not have the "Satisfied" satus and we just will not get into the corresponding bodies of conditions.

It was not so easy to track these things. In theory, it's simple. We will no longer receive confirmation for packages for some time and we can measure the maximum time between acknowledgments. But after all, the trouble is, these confirmations do not reach our handlers and are lost somewhere in the low-level functions of the protocol stack, and there it makes no sense for us to plug in with their stubs, as this is not provided and all this will be erased during the next regeneration of the project. But there is one interesting property of our structure, where there is a segment number of the last confirmation. And accordingly, if there is no such confirmation, then this number will remain constant, that is, it will not increase. This is what we will track down.

First, let's put in the function of disconnecting the connection, we will extinguish all the LEDs and light one green to see it somehow

  tcp\_close(tpcb);

**HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_13 | GPIO\_PIN\_14 | GPIO\_PIN\_15, *GPIO\_PIN\_RESET*);**

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

}

Connect the global variable of the timer counter

extern UART\_HandleTypeDef huart6;

**extern volatile uint32\_t Tim2Cnt;**

Let's **go** to the function  **tcp\_client\_poll** and add a static variable there

err\_t ret\_err;

**static uint32\_t cur\_lastack=0;**

Now before the cycle in which his body is empty, we add the following code

if (es != NULL)

{

**//Проверяем, что номер сегментов подтверждений от сервера наращивается**

**if(Tim2Cnt>20)**

**{**

**if(Tim2Cnt%40==0)**

**{**

**sprintf(str1,"%lu, %lurn", tpcb->lastack, Tim2Cnt);**

**HAL\_UART\_Transmit(&huart6,(uint8\_t\*)str1,strlen(str1),0x1000);**

**if(cur\_lastack==tpcb->lastack)**

**{**

**es->state = *ES\_CLOSING*;**

**tcp\_client\_connection\_close(tpcb, es);**

**net\_stat = *ES\_CONNECTING*;**

**Tim2Cnt = 0;**

**}**

**cur\_lastack = tpcb->lastack;**

**}**

**}**

  if (es->p\_tx != NULL)

Here everything is simple too. Again, using the operation of calculating the remainder of the division, we compare the saved value of the structure field with the current value approximately every 20 seconds, and if they are suddenly equal, then, of course, the packets did not go between the nodes, in which case we close the connection, corresponding statuses, and also reset the timer counter.

We will collect our code, we will sew the controller, let the data vary a little and we will then disconnect the network cable. After a while, the customer should turn off and the green LED on the board will light up.

Of course there was an idea that the client then began to request a connection from the server and when the latter appeared, it was created. But I was not able to achieve this. The connections were queued and when the server appeared they were all applied, that is, the server immediately installed 5 connections (this value is set in the LWIP properties). You already need to know how to work with virtual sockets in RAW API mode, but we have not reached it yet. At least we tracked something. Already what-no automation. That is, when resuming the server, we will need to restart the client.

Before we start working with the server project, we need to change the IP address in the client's project in the initialization function to the address of our future server

ipaddr\_dest[0] = 192;

ipaddr\_dest[1] = 168;

ipaddr\_dest[2] = 1;

ipaddr\_dest[3] = **191**;

In the [**next part of**](http://narodstream.ru/stm-urok-100-lan8720-lwip-tcp-soedinyaem-dva-kontrollera-chast-2/) this lesson we will work with the project for the server, then connect our motherboards on the LAN and check the operation of our code.

**STM Lesson 100. LAN8720. LWIP. TCP. We connect two controllers. Part 2**

Posted on [December 15, 2017](http://narodstream.ru/stm-urok-100-lan8720-lwip-tcp-soedinyaem-dva-kontrollera-chast-2/)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/)- [1 comment ↓](http://narodstream.ru/stm-urok-100-lan8720-lwip-tcp-soedinyaem-dva-kontrollera-chast-2/#comments)

[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/1mSLLYFDmi450D01CQkyari00000ECgs7402I09Wl0Xe172WmvR10O01sE3CgmU80QE8tP0fa06cakID99W1eiM-tYEW0SAapOqag07iWR3U8xW1tAUPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y083e0BUkAe2kG9FwD_TumkhuV02m-kXlW_u0eA0W820i0kO0wKVg0CIi0C4k0J_0UW4ymFu1FhT0OW5-jq1a0M_yW6W1Sj2g0MtEB05jpYu1TCzm0NmcGV81R3g0T05iXdW1J_m1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XHoDqzeyH8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2A-SbG602W712W0000000F0_s0e2u0g0YNhu2i3y5OWB1geB4BGy8UxuKG00qXIU2Ljr1G302u2Z1SWBWDIJ0TaB-578tJsZn4Ze2_hT0V0B1eWCjfNUlW7e30AO3PMCQF8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_uev6hqkUeuaW0?stat-id=3&test-tag=50027808448513&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY3IjoiNTAwMjc3NzkwOTY1NzYifQ%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/SWmjvHVVvjO50D41CQkyari00000ECgs7402I09Wl0Xe172AffoG1e01xCtW6OW1rBMagJMG0VZAnCelc07gZ-3LBg01zih4oY-e0UYEuDKkk07st_Rr6S010jW1ieMB5U01qBto4UW1OFW1elhUlW680WIW0g2-sHQv0a_etztZ2wlXy0BAZVwY1_W2We20W82m2u03uU7RcWc80wRI-B43i0C4w0Jp0VW4nlK2Y0N6zGAG1T_w0g05Y72e1VnUi0NyNhW5fM301R3x3CW5yAK3q0NUAk05Ml050PW6Zhkm6Q06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00071n0000gGVuKVX6NgF4IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0YGtwmBW0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mYg2n1LODF7-5400EukfWbRTGK0m0k0emN82u3Kam7P2_XH-4PUeyH8w0l6zGBm2mQ83ABwthu1w0m2c0tylMRo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Em8Gzi0u1eGy00000003mFwWF-AEHaTNhgE98?stat-id=4&test-tag=50027808448513&format-type=24&banner-test-tags=eyI2Mzg2Mzg5NjkyIjoiNTAwMjc3NzkwOTY1NzYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

In the [**previous part of**](http://narodstream.ru/stm-urok-100-lan8720-lwip-tcp-soedinyaem-dva-kontrollera-chast-1/) this lesson, we worked on a project for the client MK and tested this project, connecting to a server whose role was played by the computer.

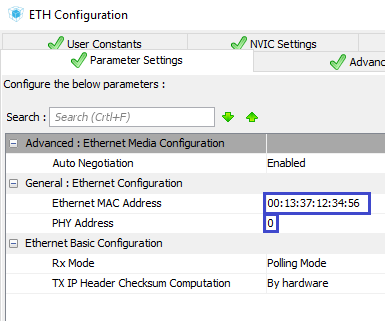
Now we move on to the server.

We will create a project for the server from the project of **lesson 98** named **LAN8720\_TCP\_SERVER**and call it **LAN8720\_TCP\_SERVER\_LCD** .

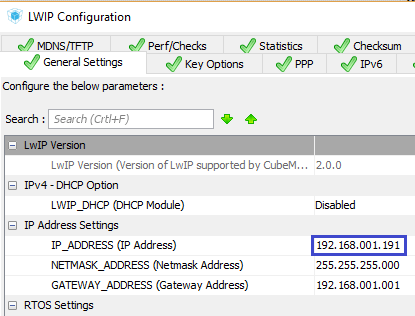
Open this project in **Cube MX** , turn on **I2C**

http://narodstream.ru/wp-content/uploads/2017/12/stm100img05.png

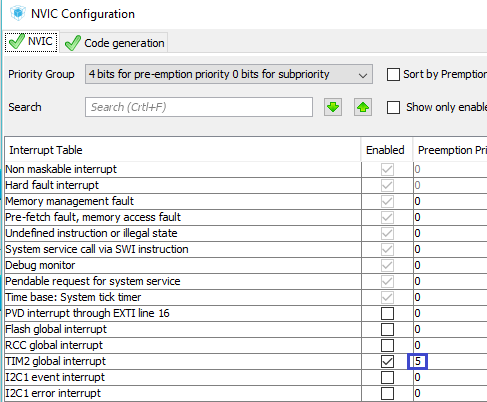
Then go to Configuration first in ETH. Here we need now not only to set PHY 0 address, as we have DIS-BB, but also to change the MAC address a little more, otherwise they have the same client and the network will not forgive



The same is with the IP address of the network. It should also be different for all participants of network connections, but in the same network mask. That is, if we have a mask of 255.255.255.0, then it turns out that the first 24 bits of them should be the same, and the last byte is different. Therefore, change this address by going to LWIP settings



Also, we'll go to NVIC and set interruptions from the timer to a smaller priority (the number on the contrary should be greater)



Generate the project, open it in System Workbench, configure it in the same way as the client and try to assemble it.

For now, also we will put a network cable into the server to communicate with the PC to debug it. Someone has this cable coming directly from the PC, someone has hubs, routers, it does not matter, the main thing is to see our PC with the server.

Now we start working with the project itself.

First, copy the files **lcd.h** and **lcd.c** from the project for HC-05 lesson 99 with the name HC\_05\_MASTER to the appropriate directories.

Let's **go to the lcd.h** file  and fix the HAL library connection

#include "stm32f**4**xx\_hal.h"

Go to the **main.c** file and connect our display library

#include "net.h"

**#include "lcd.h"**

/\* USER CODE END Includes \*/

We initialize our display in **main ()**

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

**LCD\_ini();**

**LCD\_Clear();**

net\_ini();

Start the timer switch below

tcp\_server\_init();

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

To test the display, add the following lines to the code

net\_ini();

**LCD\_StrBottom("String 1");**

**LCD\_StrBottom("String 2");**

**LCD\_StrBottom("String 3");**

**LCD\_StrBottom("String 4");**

We will collect the code, we will sew the controller and check the operation of our display



The display is tested, now you can delete the data of the line.

Just like in the case of a client in an infinite loop, we call the call of two functions into a call to one function

/\* USER CODE BEGIN 3 \*/

**MX\_LWIP\_Process();**

}

In net.h we also connect the display library

#include "lwip/tcp.h"

**#include "lcd.h"**

Go to the file net.c and add a couple of global variables

\_\_IO uint32\_t message\_count=0**, timeout\_count=0**;

**\_\_IO uint8\_t net\_stat=0;**

One variable for counting from the timer, and the second - to monitor the status of our connection with the client.

We also add in the enumeration one more kind of connection state

enum server\_states

{

*ES\_NONE* = 0,

*ES\_ACCEPTED*,

*ES\_RECEIVED*,

*ES\_CLOSING*,

***ES\_CLOSED***

};

Now in the function tcp\_server\_accept we will display the client and server addresses, also initialize the connection variable and extinguish all the LEDs

tcp\_poll(newpcb, tcp\_server\_poll, 0);

**sprintf(str1,"LOC %d.%d.%d.%d", (uint8\_t) newpcb->local\_ip.addr, (uint8\_t) (newpcb->local\_ip.addr>>8),**

**(uint8\_t) (newpcb->local\_ip.addr>>16), (uint8\_t) (newpcb->local\_ip.addr>>24));**

**//IP\_ADDR\_ANY[0],IP\_ADDR\_ANY[1],IP\_ADDR\_ANY[2],IP\_ADDR\_ANY[3]);**

**LCD\_StrBottom(str1);**

**sprintf(str1,"REM %d.%d.%d.%d", (uint8\_t) newpcb->remote\_ip.addr, (uint8\_t) (newpcb->remote\_ip.addr>>8),**

**(uint8\_t) (newpcb->remote\_ip.addr>>16), (uint8\_t) (newpcb->remote\_ip.addr>>24));**

**//IP\_ADDR\_ANY[0],IP\_ADDR\_ANY[1],IP\_ADDR\_ANY[2],IP\_ADDR\_ANY[3]);**

**LCD\_StrBottom(str1);**

**LCD\_StrBottom(" ");**

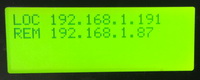
**LCD\_StrBottom(" ");**

**net\_stat=*ES\_ACCEPTED*;**

ret\_err = *ERR\_OK*;

**HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_12 | GPIO\_PIN\_13 | GPIO\_PIN\_14 | GPIO\_PIN\_15,*GPIO\_PIN\_RESET*);**

We will collect the code, let's try the controller and try to connect to our server using the Putty program, without forgetting to change the IP address. If the connection is successfully established, the IP addresses of our connection participants will be displayed



Excellent!

The next task is to display the lines coming from the client on the display.

Only lines we will display at the bottom of the display without scrolling the image so that the temperature readings from us simply change when they actually change, and do not run across the screen.

First, we in the **tcp\_server\_connection\_close** connection **termination** function **will** also assign the correct status of our variable

tcp\_close(tpcb);

**net\_stat=*ES\_CLOSED*;**

Also fill the body of the function **tcp\_server\_error** , since we completely forgot about it in the lesson on the server

static void tcp\_server\_error(void \*arg, err\_t err)

{

**struct server\_struct \*es;**

**LWIP\_UNUSED\_ARG(err);**

**es = (struct server\_struct \*)arg;**

**if (es != NULL)**

**{**

**/\* free es structure \*/**

**mem\_free(es);**

**}**

}

Basically, we release the memory of the structure in this body.

Now work with the function **tcp\_server\_recv** . First, change the state of the LED

es = (struct server\_struct \*)arg;

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

Now in the body of the condition in which we send the received string to USART, we will increase the counter of the received packets, since we have a variable for this, and we do not use it

tcp\_recved(tpcb, p->tot\_len);

**message\_count++;**

We'll clear the buffer, since without this I only received exactly 16 packets and the server no longer accepted

str1[p->len] = '\0';

**pbuf\_free(p);**

And after the transfer in USART, we will display the received string also on the display, and at the same time we will display the package number so that it can be seen that they are coming to us, we also reset the tick counter

HAL\_UART\_Transmit(&huart6, (uint8\_t\*)str1,strlen(str1),0x1000);

**str1[strlen(str1)-2] = '\0';**

**sprintf(str1,"%s %lu ",str1, message\_count);**

**LCD\_SetPos(0,3);**

**LCD\_String(str1);**

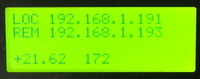
**timeout\_count = 0;**

ret\_err = *ERR\_OK*;

We'll collect our code and patch the controller.

Now let's check how the lines come to the server and are displayed. And, I hasten to please you, we will check it already, connecting our server with the real client - with the second Discovery card. The server board we leave to eat from the USB-port of the computer, as we will continue to write the code and we will need to flash it, but the client's power from the charger.

We will restart the server first, and then the client and if everything is fine, then we should see this picture on the display



Excellent! The temperature comes to us. I tried to heat the sensor with a touch, everything is displayed on the server.

You could of course finish this and finish it, but let's implement some vital idea on the server again.

We may well have a situation that we have a client will be powered by a battery and it will sit down, well, in general, our client can disappear and no packets will come from him. And then someday we'll replace the battery or find some other malfunction on the client and it suddenly appears on the network. In this case, the server will have to reboot, and then restart the client. To prevent this from happening, the server will need to catch the moment when packets from the client stop coming and breaking the connection with it, and then continue listening to the port. And, as soon as the client appears and asks for a connection, we will then connect with him.

Let's do it. We could use a timer, but I think it will sometime come in handy for more serious questions. We **already** have a wonderful timer - this is a handler function **tcp\_server\_poll** , which regularly counts down halves of seconds.

Let's move into this function and remove for the beginning this work with LEDs

~~HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_14,~~*~~GPIO\_PIN\_RESET~~*~~);~~

~~HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_12,~~*~~GPIO\_PIN\_SET~~*~~);~~

tcp\_server\_connection\_close(tpcb, es);

We will impute the tick counter

usartprop.is\_text=0;

}

**timeout\_count++;**

Then, if, for example, this meter reaches the level of 16, then close our connection, and at the same time and reset all our counters

timeout\_count++;

**if(timeout\_count>16)**

**{**

**HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_14, *GPIO\_PIN\_RESET*);**

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

**tcp\_server\_connection\_close(tpcb, es);**

**timeout\_count=0;**

**message\_count=0;**

**}**

In this body we get conditions when the buffer is empty. And as soon as it is empty for a long time, then nothing happens to us.

We will collect the code, we will sew our controller. Reboot the client, after a while it will start transmitting the temperature.

Then we disconnect the client from the power and after a while the server will disconnect, and the port, in principle, always listens. At us it is admissible up to 5 connections TCP. And when the connection is broken, I think the counter of open connections will increase again. Of course, I did not check this, but I think that this is implemented in LWIP. If we then supply power to the client, he will request a connection and the server will not refuse it and the temperature will again be fed to the server.

That's basically it.

Thus, today we realized a very interesting and, I think, long-awaited idea. We finally connected the two controllers via LAN.

I congratulate all on this event!

No wonder we have a lesson with such a jubilee number.

Thank you all for attention!

Wait for new interesting lessons.