**Lesson 93**

**Part 1**

**LAN. W5500. HTTP Server. Sockets**

Today we  will try to organize sockets for our **W5500** module .

Sockets are virtual connections that are used to organize multiple TCP connections that work simultaneously.

That is, if the client with the server wants to create another connection without closing the already created one, then a socket will be used for this. In other words, if we want to have up to 8 unbroken connections between the client and the server at the same time, then we need to organize 8 sockets. This is necessary for the fact that sometimes many programs create several simultaneous connections for faster transmission. For example, the FTP protocol generally uses 2 open connections. One is for commands and control, and the other is for data transfer.

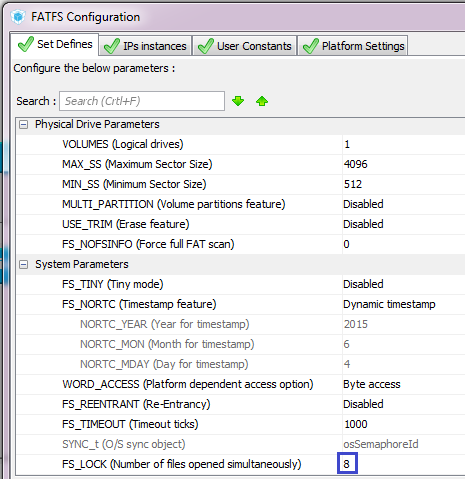
Therefore, we need somehow to ensure this.

And for this, our chip offers us a ready-made hardware approach. When we got to know her in [**lesson 91**](http://narodstream.ru/stm-urok-91-lan-w5500-http-server-chast-1/) , we saw that there are two types of registers in it. as well as memory. Designed for general use, that is independent of any connection, as well as registers and memory for sockets, which are each responsible for their connection. We initialized, and subsequently used only one of them. And today we will try to involve others.

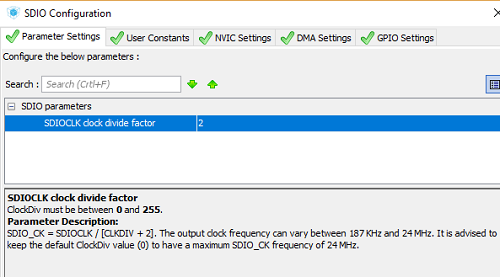
In order to sort it all out, let's create a project from the [lesson](http://narodstream.ru/stm-urok-91-lan-w5500-http-server-chast-1/) project [91](http://narodstream.ru/stm-urok-91-lan-w5500-http-server-chast-1/)**W5500\_HTTPS** and call it **W5500\_HTTPS\_SOCKETS** .

Run our project in the Cube MX.

Go to Configuration and in the FATFS parameters, change the number of simultaneously opened files to "8"

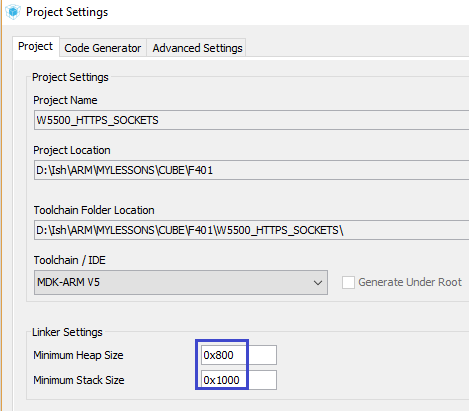


We will also make some adjustments to the settings of the SDIO peripherals. There we will set the divisor **2 there** , so that the controller does not hang up when copying through a given bus a large amount of information



If you stand on the line with the divider, then below we will see how the frequency of the SDIO bus is calculated. That is, if we leave the divisor by default to  **0** , then the frequency will be obtained according to the formula. That is, if we have a bus clock of 48 megahertz (no longer possible), then the actual bus frequency will be 24, if the divisor is set to **1** , then we get a frequency of 18 megahertz. We set the divider **2** , hence the bus frequency will be 12 megahertz.

In the settings, just in case, we'll increase the size of the stack and the heap fourfold, since there will be a lot of connections, structures and variables will also get a lot, so it may not be standard enough. Although I tested everything was enough. But how many pages will fall into the future, and I did not have 8 connections at once



Save the changes, generate the project, open it in Keil, connect our libraries **net.c** , **httpd.c** and **w5500.c** , configure the programmer for autocut, set the optimization level 1 and try to assemble.

If everything is going well, then we'll start thinking about how we can organize work with sockets.

Add to the file  **httpd.h** prtotipy two functions (for some reason they were not there, but still it worked)

void http\_request(void);

**void tcp\_send\_http\_middle(void);**

**void tcp\_send\_http\_last(void);**

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

Let's **go** to file   **w5500.c** and write in it the port initialization function in the socket after the function **w5500\_readSockBuf**

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

**void SetSockPort(uint8\_t sock\_num, uint16\_t port)**

**{**

**uint8\_t opcode;**

**opcode = (((sock\_num<<2)|BSB\_S0)<<3)|OM\_FDM1;**

**w5500\_writeReg(opcode, Sn\_PORT0,port>>8);**

**w5500\_writeReg(opcode, Sn\_PORT1,port);**

**}**

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

We did this in the initialization, so there's nothing to explain. Simply we now need a function, since this action is done for each socket and not the fact that they will have the same ports.

Let's move to the initialization function of the microcircuit in the **w5500.c** file and remove this portion of the code from it

~~//Настраиваем сокет 0~~

~~opcode = (BSB\_S0<<3)|OM\_FDM1;~~

~~w5500\_writeReg(opcode, Sn\_PORT0,local\_port>>8);~~

~~w5500\_writeReg(opcode, Sn\_PORT1,local\_port);~~

~~//инициализируем активный сокет~~

~~tcpprop.cur\_sock = 0;~~

And instead of all this, insert the following code

**//Настраиваем сокеты**

**SetSockPort(0, local\_port);**

While we also work only with a null socket, it is already more universal. We need to work until one socket, but with an emphasis on its operational change at any time.

We also delete from the initialization this line

//Посмотрим статусы

~~opcode = (BSB\_S0<<3)|OM\_FDM1;~~

And then we'll fix the code a bit, calling a higher-level function

//Посмотрим статусы

**dtt = GetSocketStatus(0);**

**sprintf(str1,"First Status Sn%d: 0x%02X\r\n",0,dtt);**

HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);

We will collect the code and check that the initialization is working.

Image01

Everything is working.

In the function of receiving the packet, we add the input argument of the socket number

void w5500\_packetReceive(**uint8\_t sn**)

Do not forget also about the prototype.

Also we pass the parameter and in the call of this function in the file **net.c**

w5500\_packetReceive(**0**);

Let's return to the **w5500** file and replace the line **tcpprop.cur\_sock**   with **sn** in the **w5500\_packetReceive** function . Since this line is found only in this function, you can use the replacement procedure, calling it the same as the procedure for finding the key combination **the Ctrl + the F** .

Also, change the called parameter where we measure the size of the incoming packet. Also in the same place, we will correct the display in the terminal program. First, we will also display the socket number there, and display only if it is not empty, that is, move the display below the output in the case of a zero packet

if(httpsockprop[sn].data\_stat == DATA\_COMPLETED)

{

  len = GetSizeRX(**sn**);

  //Если пришел пустой пакет, то уходим из функции

  if(!len) return;

  //Отобразим размер принятых данных

  sprintf(str1,"**S%d**len buf:0x%04X\r\n",**sn,**len);

  HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);

  //здесь обмениваемся информацией: на запрос документа от клиента отправляем ему запрошенный документ

The display of a pointer to data in the buffer can be deleted

~~sprintf(str1,"Sn\_RX\_RD:0x%04X\r\n",point);~~

~~HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);~~

Also closer to the end of this function, we will display in the terminal the number of the closed socket

if(httpsockprop[sn].prt\_tp == PRT\_TCP\_HTTP)

{

  tcp\_send\_http\_last(sn);

  DisconnectSocket(sn); //Разъединяемся

  SocketClosedWait(sn);

**sprintf(str1,"S%d closed\r\n",sn);**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

We pass to the file **httpd.c** and also **insert** a similar input parameter into the function void **http\_request** , as well as in its prototype in the header file

void http\_request(**uint8\_t sn**)

We return to **w5500.c** and add a parameter in the call to this function in the function of receiving packets

http\_request(**sn**);

Again, go to the file **httpd.c** and send all buffers to the same parameter, and to which of them there are prototypes, then there too

void tcp\_send\_http\_one(**uint8\_t sn**)

void tcp\_send\_http\_first(**uint8\_t sn**)

void tcp\_send\_http\_middle(**uint8\_t sn**)

void tcp\_send\_http\_last(**uint8\_t sn**)

Also, in all calls to these functions in the files **httpd.c** and **w5500.c** , we also add an argument

tcp\_send\_http\_one(**sn**);

tcp\_send\_http\_first(**sn**);

tcp\_send\_http\_middle(**sn**);

tcp\_send\_http\_last(**sn**);

In the **httpd.c** file,   **we** now replace the occurrence of  **tcpprop.cur\_sock**   with **sn** .

I got 109 replacements.

In the functions tcp\_send\_http\_first and tcp\_send\_http\_last we will remove the output to the terminal program of the fact of setting the pointer in the file

~~sprintf(str1,"f\_lseek: %d\r\n",result);~~

~~HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);~~

And this is also in the file  **w5500.c**

~~HAL\_UART\_Transmit(&huart2,(uint8\_t\*)"HTTP\r\n",6,0x1000);~~

We can now remove the structure of  **tcp\_prop** from the file w5500.h, and also remove the declarations of all variables of its type.

In theory, we are no longer tied to the null socket as much as it was before.

Let's check first the working capacity on the null socket, collecting the code, flashing the controller and requesting a document in the browser.

If everything works, then we will try to change the null socket to the third one.

To do this, it is enough to change the called parameter in the following lines in the files **net.c** and **w5500.c** in the following lines

net: c:

w5500\_packetReceive(**3**);

**w5500.c** :

//Настраиваем сокеты

SetSockPort(**3**, local\_port);

//Открываем сокет

OpenSocket(**3**,Mode\_TCP);

SocketInitWait(**3**);

//Начинаем слушать сокет

ListenSocket(**3**);

SocketListenWait(**3**);

HAL\_Delay(500);

//Посмотрим статусы

dtt = GetSocketStatus(**3**);

sprintf(str1,"First Status Sn%d: 0x%02X\r\n",**3**,dtt);

Also check the work after installation as a working socket 3.

If everything works well, then it remains for us to use several sockets at once. It will also be very easy.

To do this, create a local variable in the initialization function in the file **w5500.c**

void w5500\_ini(void)

{

**uint8\_t i;**

  uint8\_t dtt=0;

And rebuild some code at the end of the function

  w5500\_writeReg(opcode, SIPR3,ipaddr[3]);

  //Настраиваем сокеты

**for(i=3;i<8;i++)**

**{**

    SetSockPort(**i**, local\_port);

    //Открываем сокет

    OpenSocket(**i**,Mode\_TCP);

    SocketInitWait(**i**);

    //Начинаем слушать сокет

    ListenSocket(**i**);

    SocketListenWait(**i**);

**}**

  HAL\_Delay(500);

  //Посмотрим статусы

**for(i=3;i<8;i++)**

**{**

    dtt = GetSocketStatus(**i**);

    sprintf(str1,"First Status Sn%d: 0x%02X\r\n",**i**,dtt);

    HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);

**}**

}

That is, we initialized 6 sockets with numbers 3-7.

It remains also in the file net.c also to correct the call of the function of receiving and processing packets, making it also in the cycle

void packet\_receive(void)

{

**uint8\_t i;**

**for(i=3;i<8;i++)**

**{**

    w5500\_packetReceive(**i**);

**}**

}

Also, in the **httpd.c** file, **we** fix in the global array the number of elements

http\_sock\_prop\_ptr httpsockprop[**8**];

The same is **done** in the file **w5500.c**

extern http\_sock\_prop\_ptr httpsockprop[**8**];

All this is very good, but there is another interesting ambush. I often forget about it.

We transfer files, sockets work in turn, and we have one variable for the file, but there must be one for each file. That is, each socket works with its file, and it happens simultaneously with when other sockets work with other files.

Therefore, we will turn the variable for the file in the file httpd.c into an array

**static** FIL MyFile**[8]**;

Also in this file, the occurrence of the string  **MyFile is replaced** by the string **MyFile [sn]** .

In the [**next part of the**](http://narodstream.ru/stm-urok-93-lan-w5500-http-server-sokety-chast-2/) lesson, we will correct the code for the function of receiving and processing HTTP packets and check our code in practice.

**Lesson 93**

**Part 2**

**LAN. W5500. HTTP Server. Sockets**

In the [**previous part of the**](http://narodstream.ru/stm-urok-93-lan-w5500-http-server-sokety-chast-1/) lesson we got acquainted with the concept of sockets, and also made some corrections to the code for working with them.

In the **http\_request** function, **we** change the output of the file name in USART a little, adding also the socket number

  strcpy(httpsockprop[sn].fname,tmpbuf);

}

**sprintf(str1,"S%d %srn",sn, httpsockprop[sn].fname);**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

f\_close(&httpsockprop[sn].MyFile);

Here we also display the number of the socket

sprintf(str1,"**S%d**f\_open: %drn",**sn,**result);

HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);

sprintf(str1,"**S%d**f\_size: %lurn",**sn,**MyFile[sn].fsize);

HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);

And here

sprintf(str1,"**S%d**data size:%lu; cnt data part:%u; last\_data\_part\_size:%urn",**sn,**

(unsigned long)httpsockprop[sn].data\_size, httpsockprop[sn].cnt\_rem\_data\_part, httpsockprop[sn].last\_data\_part\_size);

And here we will confirm in the terminal program the closing of the socket in case of transfer of a single-buffer page. We will display a little differently in order to distinguish

DisconnectSocket(sn); //Разъединяемся

SocketClosedWait(sn);

**sprintf(str1,"S%d (one) closedrn",sn);**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

In the function  **tcp\_send\_http\_one** in the body of the condition in the case of a page with an error, we will display this information with the socket number

else

{

**sprintf(str1,"S%d file not foundrn",sn);**

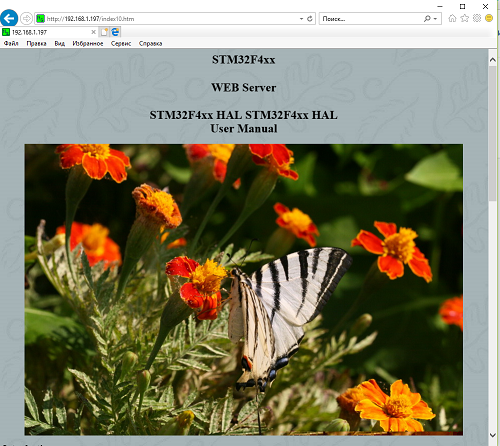
**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

  header\_len = strlen(error\_header);

We will collect the code, we will sew the controller and try to request some document heavier. I also added a background to the page in the form of a repeating picture using the design style

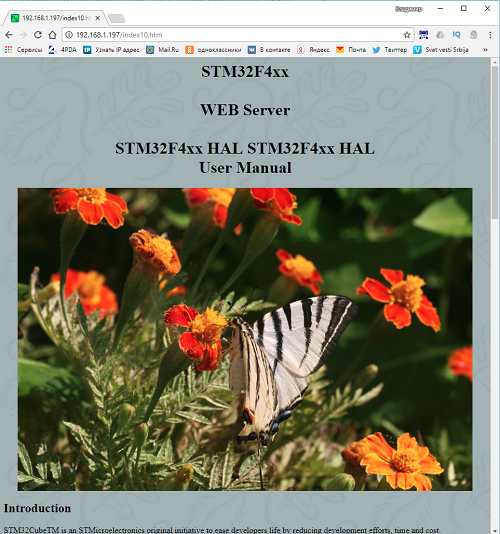
**<body style="background-image:url(IMG/img01.jpg)">**

We should get this result in browsers



We see that everything is displayed. And why in browsers, but because now we can request pages from our mini-server in any server, we are not afraid now that the browser will want to open several connections.

Here is the result in the Google Crome browser



Let's try to open this file with several pictures

**<html><body>**

**<html><body><h1 style="text-align: center;">STM32F103x8<br><br>WEB Server</h1>**

**<p>**

**<img src="/IMG/img04.jpg" />**

**<img src="/IMG/img05.jpg" />**

**<img src="/IMG/img06.jpg" />**

**<img src="/IMG/img07.jpg" />**

**<img src="/IMG/img08.jpg" />**

**<img src="/IMG/img09.jpg" />**

**<img src="/IMG/img10.jpg" />**

**<img src="/IMG/img11.jpg" />**

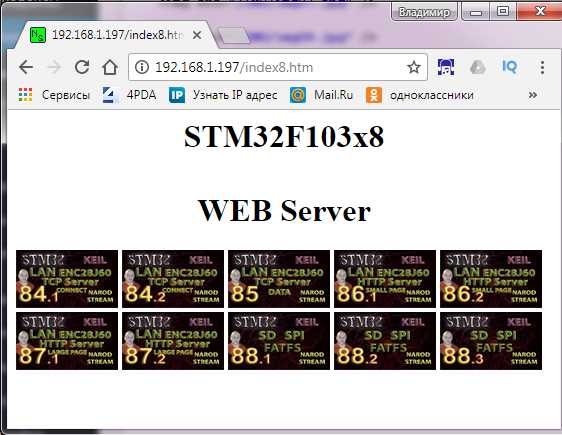
**<img src="/IMG/img12.jpg" />**

**<img src="/IMG/img13.jpg" />**

**</p>**

**</body></html>**

Open it in Google Chrome



Let's see how the sockets switch in the terminal program

**Terminal out**

We see that all 6 sockets are involved in the work. Excellent! So everything is correct.

Thank you all for your attention!