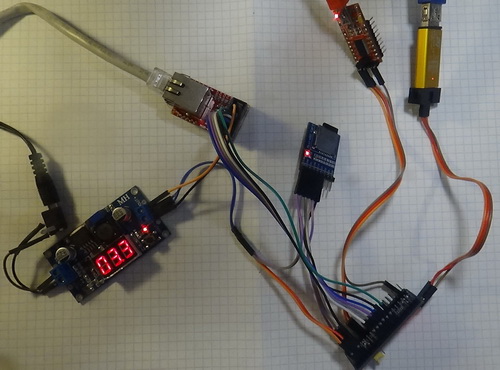
**Lesson 89**

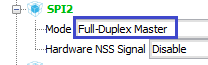
**LAN. ENC28J60. TCP WEB Server. Connect an SD card**

We continue to connect the **LAN ENC28J60** module   to the  **STM32F103 microcontroller** located on the inexpensive debugging board of the same name. We were able to respond to the client's request for a web page, but only we can not send large documents and images anyway, because we have approached to such a point that we will not have any memory for this, either operational or non-volatile. To help us we will get an old good  **SD card** , which we will connect via the **SPI** interface  . We learned how to connect the card via the SPI interface using the FATFS library in the [**previous lesson**](http://narodstream.ru/stm-urok-88-sd-spi-fatfs-chast-1/) , so today's task is to connect our [**last lesson**](http://narodstream.ru/stm-urok-87-lan-enc28j60-tcp-web-server-peredayom-stranicu-pobolshe-chast-1/) project [**on LAN**](http://narodstream.ru/stm-urok-87-lan-enc28j60-tcp-web-server-peredayom-stranicu-pobolshe-chast-1/) with the project[**the last lesson**](http://narodstream.ru/stm-urok-88-sd-spi-fatfs-chast-1/) .

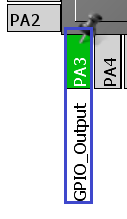
Our scheme remains the same. We connect the LAN module back to the controller, since when we were dealing with an SD card, we turned it off, and leave the SD card where it was. Now our scheme will turn out like this (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2017/08/image00-1.jpg)

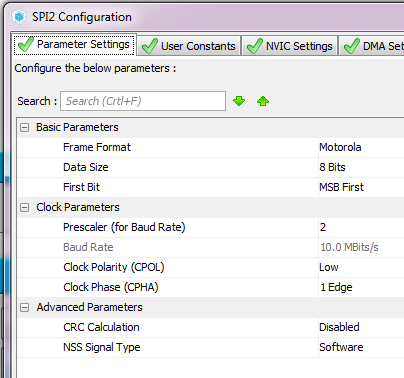
Now the project. The project is made from the project of the [**lesson 87**](http://narodstream.ru/stm-urok-87-lan-enc28j60-tcp-web-server-peredayom-stranicu-pobolshe-chast-1/) named **ENC28J60\_HTTPS\_LARGE** and was named **ENC28J60\_HTTPS\_SD** . The only difference is that I put it in the WB subfolder, since the project will be developed already in the **System Workbench for STM32** , since after connecting the **FATFS** library and writing several lines of code, the project exceeded the cherished 32 kilobytes of code, which Keil's free license to do categorically prohibits. And, as you know, we do not violate any rights, so we had to resort to a free development environment. I tried **CooCox**, but this is a separate story, and I abandoned this occupation. Yes, plus all, I still managed to defeat the insufficient heap size in the **System Workbench** and I finally resumed the work of the external assembler. If anyone is interested, as I did, write in the comments. In general, the choice of the compiler is your business, so you can use any. The difference between them in setting up and working is not so great. We open the project in **Cube MX** , include **SPI2** and **FATFS**

   index19

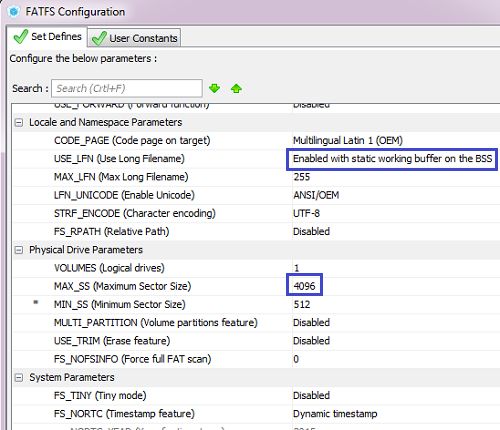
Also, we **'ll plug PA3** foot **onto the output** , since this is the **Chip Select** for **SD card**



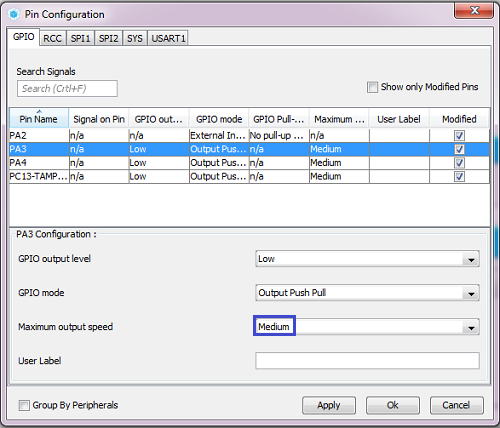
Go to the tab with the settings of the interfaces ( **Configuration** ) and first configure the **SPI** (the divider can now be turned on 2 and the speed at us will increase to 10 mbps, since the card at this speed works fine and we do not need to analyze anything with the logic analyzer), And such settings the SPI Cube automatically installs the bus, so do not touch anything and leave this dialog



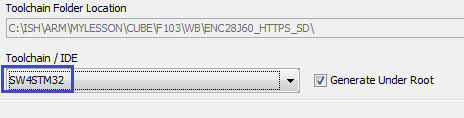
We will also configure FATFS, adding support for long names, suddenly come in handy, and also increasing the maximum sector size



Also, do not forget to enable the average speed for the **PA3** foot in the **GPIO**



Let's go into the project settings and change the development environment there



Save the settings, generate the project, go to System Workbench, connect our freshly generated project and in the properties make sure that we do not have ready-made debugger settings, and if there is, we will delete it, as always. We do not change the sinker to the inner. We open main.c and collect our project. If everything is fine, you can try to run it and check that everything still works for us, for example, by pinging our module.

In the **Inc** and **Sr** c folders of the project, we copy the files of our library for the **SD card** , which we created in the last lesson from the **SD\_FATFS** project - **sd.c** and **sd.h, respectively** . Update the project tree in the development environment ( **Refresh** ). Let's **go** to the file **sd.c** and the global string variable, add extern and remove its initialization

**extern** char str1[60]~~={0}~~;

We will also need to make some more corrections here. Some functions related to the SPI interface, or rather their names have already been used in our project for LAN, so let's change them a little, for example by adding the prefix **SD\_**

uint8\_t **SD\_**SPIx\_WriteRead(uint8\_t Byte)

{

void **SD\_**SPI\_SendByte(uint8\_t bt)

{

uint8\_t **SD\_**SPI\_ReceiveByte(void)

{

Accordingly, we will also need to add the same prefix in all locations of these function calls below.

We will also remove the **SD\_PowerOn** function and its prototype, since we will not need to wait for 20 milliseconds in the initialization of the card, they will already pass with interest, since the LAN module will be initialized first, then sometime we will receive the document request from the client, and then only the initial initialization of the card will take place.

Correspondingly, then remove the global variable for the timer

~~extern volatile uint16\_t Timer1;~~

Now go into the file **user\_diskio.c** , delete all the code there and paste it from the **SD\_FATFS** project file of the same name . In the body of the **USER\_initialize** function  **,** delete the function call **SD\_PowerOn** .

~~SD\_PowerOn();~~

if(sd\_ini()==0) {Stat &= ~STA\_NOINIT;} //сбросим статус STA\_NOINIT

So far with the connection cards like finished. Once again just in case, we'll compile the code to make sure that we have no errors.

Now go to the tcp.c file and start working on how we can now retrieve the requested documents from the SD card file system.

In the function **tcp\_read** comment out the output of information in the **USART** , so as not to slow down. If something does not work, then we can always uncomment it

**/\***

**sprintf(str1,"%d.%d.%d.%d-%d.%d.%d.%d %d tcp\r\n",**

**ip\_pkt->ipaddr\_src[0],ip\_pkt->ipaddr\_src[1],ip\_pkt->ipaddr\_src[2],ip\_pkt->ipaddr\_src[3],**

**ip\_pkt->ipaddr\_dst[0],ip\_pkt->ipaddr\_dst[1],ip\_pkt->ipaddr\_dst[2],ip\_pkt->ipaddr\_dst[3], len\_data);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**\*/**

**//HAL\_UART\_Transmit(&huart1,(uint8\_t\*)"ACK\r\n",5,0x1000);**

 In addition to the size of the segment, we need now also the window size, we are going to transfer large files. To do this, add the global variable in the same file

volatile uint16\_t tcp\_mss = 458;

**volatile uint16\_t tcp\_size\_wnd = 8192;**

In the **tcp\_header\_prepare** function **,** change the line

tcp\_pkt->size\_wnd = be16toword(**tcp\_size\_wnd**);

And now, to us when the number of transmitted data is approached to the window size in the **PCH** data packet  , we need a variable in the **tcp\_prop** structure  . To do this, go to the file  **tcp.h**  and add it

volatile uint16\_t cnt\_rem\_data\_part;//количество оставшихся частей данных для передачи

**volatile uint16\_t cnt\_size\_wnd;//количество переданных байтов окна**

Also in the same structure, since we are going to take data for transfer from the file, we will add an array for storing the filename

**char fname[20];//имя файла (документа)**

} tcp\_prop\_ptr;

We return to the **tcp.c** file   and initialize the field with the number of window bytes transferred in the **tcp\_read** function in the document request by the client

if (strncmp((char\*)tcp\_pkt->data,"GET /", 5) == 0)

{

**//инициализируем количество переданных байтов окна**

**tcpprop.cnt\_size\_wnd = 0;**

Now we need to find a place where we can exceed the size of the window and transfer the corresponding flag there.

The function of transferring a page in the size of one package does not interest us at all, there we will definitely not exceed anything.

Let's go into the function of transmitting the first packet of a multipack page and we will build up our counter on the size of the segment

tcpprop.cnt\_rem\_data\_part--;

**//добавим переданные байты в окно**

**tcpprop.cnt\_size\_wnd += tcp\_mss;**

Now let's move to the middle part of the page and, where we pass the **ACK** flag  , if we reach the maximum window size, we will also give the **PCH** flag

len=len\_tcp + tcp\_mss;

**//Узнаем, не подолши ли мы к предельному размеру окна**

**if ((tcp\_size\_wnd - tcpprop.cnt\_size\_wnd) > tcp\_mss)**

**{**

**tcp\_header\_prepare(tcp\_pkt, port, TCP\_ACK, len\_tcp, len);**

**}**

**else**

**{**

**tcp\_header\_prepare(tcp\_pkt, port, TCP\_PSH|TCP\_ACK, len\_tcp, len);**

**//инициализируем счётчик байтов окна заново, так как дальше будем передавать уже следующее окно**

**tcpprop.cnt\_size\_wnd = 0;**

**}**

len+=sizeof(ip\_pkt\_ptr);

We will advance on the function a little lower and also increase the counter

tcpprop.cnt\_rem\_data\_part--;

**//добавим переданные байты в окно**

**tcpprop.cnt\_size\_wnd += tcp\_mss;**

Let's **go** to the header file  **tcp.h**  and change the macros of the document variants a little, because now we will not only transfer the main page and the error page, but all the documents files that the client will ask us if they are present on our SD card

**//Варианты документов HTTP**

**#define EXISTING\_HTML 0**

**#define E404\_HTML 1**

**#define EXISTING\_JPG 2**

Now we will start working directly with the SD-card.

Let's return to the **tcp.c** file   and add three more local variables in the **tcp\_read** function

uint16\_t i=0;

**char \*ss1;**

**int ch1=' ';**

**int ch2='.';**

In the same function, where we filter out the main page's page from other queries, we completely rewrite the bodies of the condition (both true and opposite)

if((char)tcp\_pkt->data[5]==' ')

{

**strcpy(tcpprop.fname,"index.htm");**

**tcpprop.http\_doc = EXISTING\_HTML;**

}

else

{

**//скопируем 20 байтов из запроса после символа '/' в поле fname**

**memcpy((void\*)tcpprop.fname,(void\*)(tcp\_pkt->data+5),20);**

**//найдём пробел и заменим его нулём**

**ss1 = strchr(tcpprop.fname,ch1);**

**ss1[0] = 0;**

}

Further in all places file where the macro occurs  **index\_html** , replace it with a new macro  **EXISTING\_HTML** , so that when the assembly code, no errors occur.

In the file **tcp.h we** connect the **FATFS** library

#include "net.h"

**#include "fatfs.h"**

Let's return to the **tcp.c** file and add a few global variables to work with the file system

extern uint8\_t ipaddr[4];

**FATFS SDFatFs;//указатель на объект**

**extern char USER\_Path[4]; /\* logical drive path \*/**

**FIL MyFile;**

**FRESULT result; //результат выполнения**

**uint32\_t bytesread;**

We will also add one more global string array for the header of the transfer of the picture in **jpeg** format  , at the same time in other arrays we will include version  **1.0** , and we will forbid not to close the connection so that the client does not want too much from our server

const char http\_header[] = {"HTTP/**1.0** 200 OK\r\nServer: nginx\r\nContent-Type: text/html\r\nConnection: **close**\r\n\r\n"};

**const char jpg\_header[] = {"HTTP/1.0 200 OK\r\nServer: nginx\r\nContent-Type: image/jpeg\r\nConnection: close\r\n\r\n"};**

const char error\_header[] = {"HTTP/**1.0** 404 File not found\r\nServer: nginx\r\nContent-Type: text/html\r\nConnection: **close**\r\n\r\n"};

Then we continue to write the code for the **tcp\_read** function .

After a condition in which we just replaced the above, display the file name in the terminal program, mount the file system, try to open the requested file (or the main page if there was a space) and take its size from the structure

  ss1[0] = 0;

}

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)tcpprop.fname,strlen(tcpprop.fname),0x1000);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)"\r\n",2,0x1000);**

**result=f\_mount(&SDFatFs,(TCHAR const\*)USER\_Path,0);**

**sprintf(str1,"f\_mount: %d\r\n",result);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**result=f\_open(&MyFile,tcpprop.fname,FA\_READ); //Попытка открыть файл**

**sprintf(str1,"f\_open: %d\r\n",result);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**sprintf(str1,"f\_size: %lu\r\n",MyFile.fsize);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

tcpprop.cnt\_rem\_data\_part = tcpprop.data\_size / tcp\_mss + 1;

And after adding this code for some reason during the assembly, I got an error about the omission of the code for the realizations of functions related to long names

C: \ ISH \ ARM \ MYLESSON \ CUBE \ F103 \ WB \ ENC28J60\_HTTPS\_SD \ Debug /../ Middlewares / Third\_Party / FatFs / src / ff.c: 1943: undefined reference to ff\_convert'

C:\ISH\ARM\MYLESSON\CUBE\F103\WB\ENC28J60\_HTTPS\_SD\Debug/../Middlewares/Third\_Party/FatFs/src/ff.c:1997: undefined reference to ff\_convert '

Middlewares / Third\_Party / FatFs / src / ff.o: In function cmp\_lfn':

C:\ISH\ARM\MYLESSON\CUBE\F103\WB\ENC28J60\_HTTPS\_SD\Debug/../Middlewares/Third\_Party/FatFs/src/ff.c:1361: undefined reference to ff\_wtoupper '

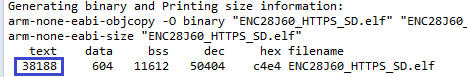
C: \ ISH \ ARM \ MYLESSON \ CUBE \ F103 \ WB \ ENC28J60\_HTTPS\_SD \ Debug /../ Middlewares / Third\_Party / FatFs / src / ff.c: 1362: undefined reference to `ff\_wtoupper '

We encountered in the past occupation with such a mistake and attached the file **ccsbcs.c** , which we generated, but was not attached. This time, for some reason, this file has not even been generated for me, so take it from the previous project and put it in the folder with " **Src** ", then make **refresh** in the project, open this file and remove the dots from the line with the connection of the header file

**#include "ff.h"**

After that, we update the project tree and the project will normally be collected.

That is, only now we have started to use the **FATFS** library , as evidenced by the sharply increased firmware size



That's why we are now working with the System WorkBench programming environment.

Then the case of the positive result of opening the file (if the file exists on the SD card we first study its extension and based on this we will use the corresponding header for the HTTP packet, otherwise we will pass the document with an error

HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);

**if (result==*FR\_OK*)**

**{**

**//изучим расширение файла**

**ss1 = strchr(tcpprop.fname,ch2);**

**ss1++;**

**if (strncmp(ss1,"jpg", 3) == 0)**

**{**

**tcpprop.http\_doc = EXISTING\_JPG;**

**//сначала включаем в размер размер заголовка**

**tcpprop.data\_size = strlen(jpg\_header);**

**}**

**else**

**{**

**tcpprop.http\_doc = EXISTING\_HTML;**

**//сначала включаем в размер размер заголовка**

**tcpprop.data\_size = strlen(http\_header);**

**}**

**//затем размер самого документа**

**tcpprop.data\_size += MyFile.fsize;**

**}**

**else**

**{**

**tcpprop.http\_doc = E404\_HTML;**

**//сначала включаем в размер размер заголовка**

**tcpprop.data\_size = strlen(error\_header);**

**//затем размер самого документа**

**tcpprop.data\_size += sizeof(e404\_htm);**

**}**

tcpprop.cnt\_rem\_data\_part = tcpprop.data\_size / tcp\_mss + 1;

With this function finished, now go to the transfer functions. start with the function  **tcp\_send\_http\_one** .

In this function, we replace the code in the place where we fill the field of the structure intended for data

//Отправляем страницу

**if ((tcpprop.http\_doc==EXISTING\_HTML)||(tcpprop.http\_doc==EXISTING\_JPG))**

**{**

**result=f\_mount(&SDFatFs,(TCHAR const\*)USER\_Path,0);**

**sprintf(str1,"f\_mount: %d\r\n",result);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**result=f\_open(&MyFile,tcpprop.fname,FA\_READ); //Попытка открыть файл**

**sprintf(str1,"f\_open: %d\r\n",result);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**sprintf(str1,"f\_size: %lu\r\n",MyFile.fsize);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**result=f\_lseek(&MyFile,0); //Установим курсор чтения на 0 в файле**

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

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**if (tcpprop.http\_doc==EXISTING\_HTML)**

**{**

**strcpy((char\*)tcp\_pkt->data,http\_header);**

**result=f\_read(&MyFile,(void\*)(tcp\_pkt->data+strlen(http\_header)),(uint16\_t)MyFile.fsize,(UINT \*)&bytesread);**

**}**

**else**

**{**

**strcpy((char\*)tcp\_pkt->data,jpg\_header);**

**result=f\_read(&MyFile,(void\*)(tcp\_pkt->data+strlen(jpg\_header)),(uint16\_t)MyFile.fsize,(UINT \*)&bytesread);**

**}**

**}**

else

The code here is very simple, we open the file, set the pointer to it and then read the data in the amount that is declared in the corresponding field of our structure. There is more service output to the terminal program, which can then be deleted if the result is positive.

Similar actions we will do in the following function - **tcp\_send\_http\_first**

//Отправляем первую часть страницы

**if ((tcpprop.http\_doc==EXISTING\_HTML)||(tcpprop.http\_doc==EXISTING\_JPG))**

**{**

**strcpy((char\*)tcp\_pkt->data,http\_header);**

**result=f\_mount(&SDFatFs,(TCHAR const\*)USER\_Path,0);**

**result=f\_open(&MyFile,tcpprop.fname,FA\_READ); //Попытка открыть файл**

**result=f\_lseek(&MyFile,0); //Установим курсор чтения на 0 в файле**

**if (tcpprop.http\_doc==EXISTING\_HTML)**

**{**

**strcpy((char\*)tcp\_pkt->data,http\_header);**

**result=f\_read(&MyFile,(void\*)(tcp\_pkt->data+strlen(http\_header)),tcp\_mss-strlen(http\_header),(UINT \*)&bytesread);**

**}**

**else**

**{**

**strcpy((char\*)tcp\_pkt->data,jpg\_header);**

**result=f\_read(&MyFile,(void\*)(tcp\_pkt->data+strlen(jpg\_header)),tcp\_mss-strlen(jpg\_header),(UINT \*)&bytesread);**

**}**

**}**

else

The next function is  **tcp\_send\_http\_middle**

**if ((tcpprop.http\_doc==EXISTING\_HTML)||(tcpprop.http\_doc==EXISTING\_JPG))**

**{**

**if (tcpprop.http\_doc==EXISTING\_HTML)**

**{**

**result=f\_lseek(&MyFile,((uint32\_t)tcp\_mss\*(tcpprop.cnt\_data\_part-tcpprop.cnt\_rem\_data\_part))-strlen(http\_header)); //Установим курсор чтения в файле**

**}**

**else**

**{**

**result=f\_lseek(&MyFile,((uint32\_t)tcp\_mss\*(tcpprop.cnt\_data\_part-tcpprop.cnt\_rem\_data\_part))-strlen(jpg\_header)); //Установим курсор чтения в файле**

**}**

**result=f\_read(&MyFile,(void\*)tcp\_pkt->data,tcp\_mss,(UINT \*)&bytesread);**

**}**

else

It's almost the same here, only we set the pointer not to 0, but to the appropriate place in the file, and before that we do not mount the file system and open the file, as we have already done.

Well, the last function is  **tcp\_send\_http\_last**

**if ((tcpprop.http\_doc==EXISTING\_HTML)||(tcpprop.http\_doc==EXISTING\_JPG))**

**{**

**if (tcpprop.http\_doc==EXISTING\_HTML)**

**{**

**result=f\_lseek(&MyFile,(tcp\_mss\*(tcpprop.cnt\_data\_part-1))-strlen(http\_header)); //Установим курсор чтения в файле**

**}**

**else**

**{**

**result=f\_lseek(&MyFile,(tcp\_mss\*(tcpprop.cnt\_data\_part-1))-strlen(jpg\_header)); //Установим курсор чтения в файле**

**}**

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

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**result=f\_read(&MyFile,(void\*)tcp\_pkt->data,tcpprop.last\_data\_part\_size,(UINT\*)&bytesread);**

**}**

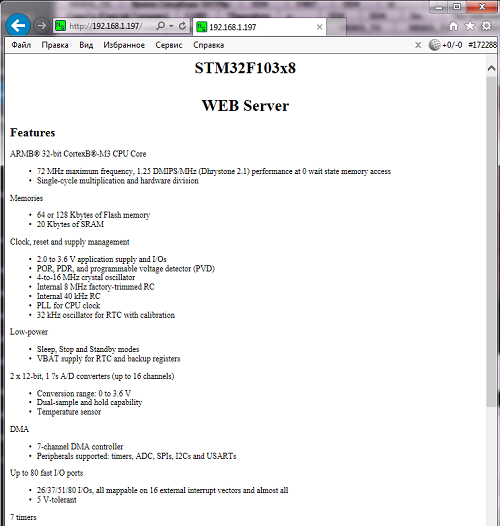
else

An array with the main page can now be deleted

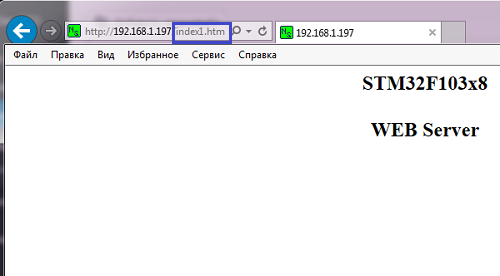
~~const uint8\_t index\_htm[] = {0x3c,~~

We will collect the code, we will tell the controller and try to request various documents from the server.

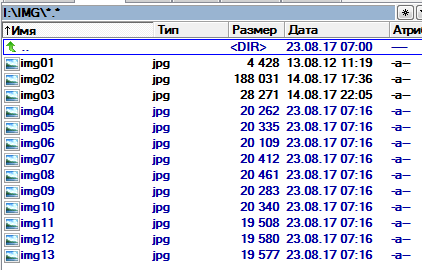
Request the home page (click on the image to enlarge)

[](http://narodstream.ru/wp-content/uploads/2017/08/Image06.png)

Request another file, for example  **index1.htm**



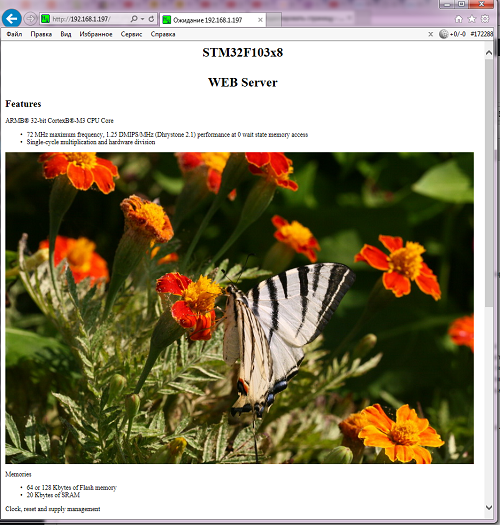
Download several files in **jpeg** format to the IMG folder on the SD card



Also open the file **index.htm** and add the image from the file somewhere in the text of the page

</ul></p>  
**<p>  
  <img src="/IMG/img02.jpg" />  
</p>**  
<p>Memories</p>

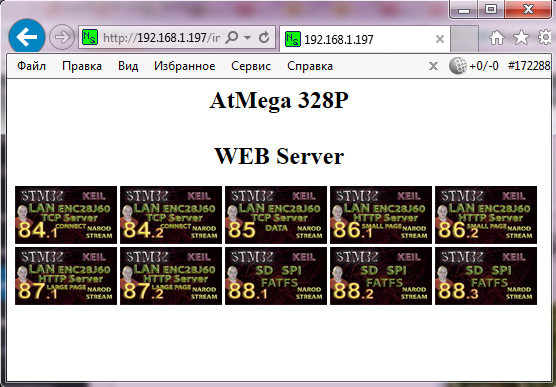
We will insert the card back into our homemade card reader, connect the power and ask for the main page again (click on the picture to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2017/08/Image09.png)

You can insert a smaller picture and more but only one, otherwise the browser tries to open several connections, but we do not have such support. I tried to add virtual sockets, but apparently because the free memory we are already tending to zero, the socket number, as well as some variables, although with the **volatile** attribute   start without my knowledge to change the value, so I abandoned this occupation. Here you need a controller more powerful. But even so, I think not bad. if we want to add some pictures anyway, although small ones, we can bypass this nuance by introducing the image data straight into the html code, that is, into the page itself. To do this, there are several online services with which you can make these conversions. The code looks something like this

**<img width="102" height="58" title="" alt="" src="data:image/jpeg;base64,/9j/4RSXRXhpZgAASUkqA... и т.д.**

So I got a lot of images on the page



Thus, with the help of today's lesson, we were able to improve our HTTP server by expanding its memory for storing files by connecting an **SD card** and file system. Although of course the speed leaves much to be desired, since the card is connected via the **SPI** interface , which does not allow us to communicate with it at a speed exceeding 1 megabit per second, nevertheless we again worked on the HTTP protocol, having also learned to take into account both the window size and transfer other documents except the main page, and other formats.

Thank you for attention!