**STM Lesson 102. LAN8720. LWIP. HTTP RAW. Part 1**

Posted on [December 26, 2017](http://narodstream.ru/stm-urok-102-lan8720-lwip-http-raw-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-102-lan8720-lwip-http-raw-chast-1/#respond)

[Programming for beginnersDownload the free book and become a real programmer. Get it now:To learn moresheremetev.info](https://an.yandex.ru/count/JJEtOT7CFPK50Dy1CGMzari00000ECgs7402I09Wl0Xe172WmvR10O01vE3CgmU80VY6tP0fa06YakID99W1Zhs-tYEW0VwIp8qag06M_w_U8xW1XgMPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y083e0BUkAe2kG94Hz2y6UbE8l02fekZYWtu0eA0W820q1E00__Fl_qBY0ElXgkW2fW3YSK4g0CIi0C4k0J_0UW4f0Vu1AJR0OW5fDi1a0MQcmAW1UH1g0NWDx05u3Uu1O18m0NmcGV81R3g0T05YndW1Lhm1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XHB4I_kCH8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2A-SbG602W712W0000000F0_s0e2u0g0YNhu2i3y5OWB1geB40IEyu_uKG00YMx-3bjr1G302u2Z1SWBWDIJ0TaB-54iHB-un4Ze2wJR0V0B1eWCjfNUlW7e30AO3PMCQF8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_uey7ZwvUkuaW0?stat-id=3&test-tag=86311692165121&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY1IjoiODYzMTE2NjI4MTMxODQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

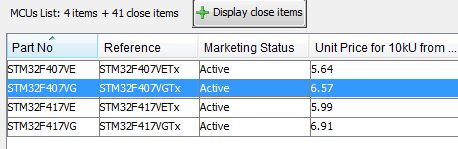
[Programming kitRobot, mobile application and study guide. Learn now is interesting!To learn moredorado-uno.ru](https://an.yandex.ru/count/QSCYEWXfO8050Di1CGQzari00000ECgs7402I09Wl0Xe172AffoG1e01d_tFcvogwEbHY06ZjgIfDP01Yih4oY-O0SIGuDKke06CoiJABwW1me_WrIwu0T3WzlKPm042s06oXOiLu07GlV8Hw05W-06Y-jw-0OW21A02eBxP5ha2H4VGl1dfJYBm0fw7uhW8-0A2W820WD0JW0FjgkwX3eW3tEoP-mAO0vlD0h031EW4zWFu1Fwc0OW5_gO1a0MEwGAW1Omog0MwAh05kYgu1UTRm0Mm-mp81V2b0z05ZHFW1Lhm1G6O1ewxi1cW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000Aa7-54C5-sun4Ym1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08aD-i2u0A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i8gWiGQFACTFXH000fumaFMtK50C0BWAC5o0k0r9C1sGluKGmNxRZ4IEWB_gO1y0i6Y0oY-jw-0UWC0fWD_BrcyWq0-Wq0WWu0YGu00000003mFv0Em8Gzc0x6rypKbOdWkXEW3i24FR0E0Q4F00000000y3-e3_YZuRl3dQxYI000?stat-id=4&test-tag=86311692165121&format-type=24&banner-test-tags=eyI2Mzg2Mzg5NjM3IjoiODYzMTE2NjI4MTMxODQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

We continue to work with the LWIP protocol stack and the LAN8720 chip.

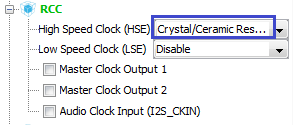
And now we will try to create a small, but quite functional and useful **HTTP** server based on our chip and controller. We will use the same **STM32F4-Discovery** debug card and **STM32F4DIS-BB** expansion **card** . Who does not have such a board, then it can use a module from WaveShare. Just me with this board is much easier and more comfortable to work with and if that, then do not have to complain about bad connections with the module because of poor-quality wires.

We also work with the **RAW API** , but we will better create the project again, since we now do not need a display and create a new project is not so difficult.

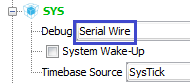
Run Cube MX and create a new project by selecting our controller



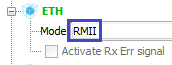
Let us configure RCC



Let's choose a debugger



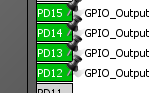
Enable Ethernet



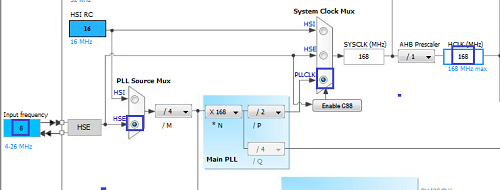
Then LWIP

http://narodstream.ru/wp-content/uploads/2017/12/stm102img04.png

We plug the legs of the LEDs to the output, they will also be required

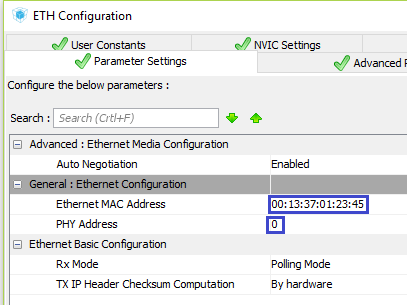


Let's go to  **Clock Configuration**  and adjust the dividers and multipliers to set the required frequencies (click on the image to enlarge the image)

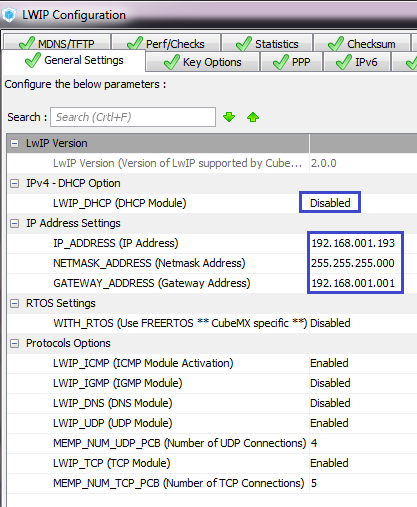
[](http://narodstream.ru/wp-content/uploads/2017/12/stm102img06.png)

Go to **Configuration** .

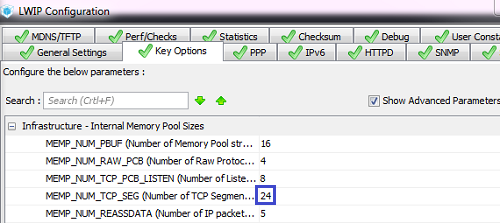
First, set up **ETH** . Here, as usual, the MAC address and address of PHY

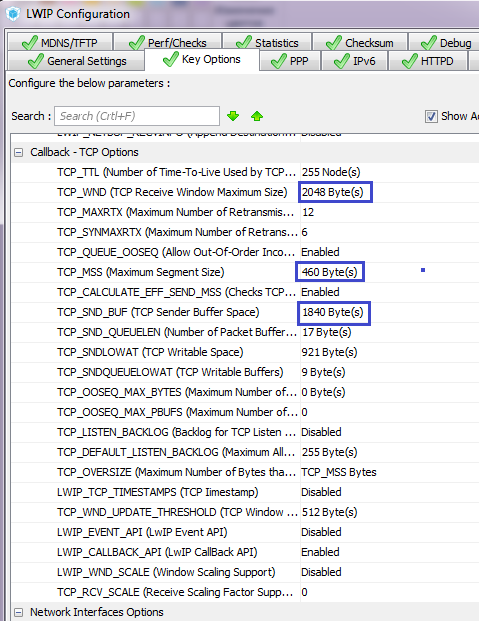


Further **LWIP** . Fill in first the General Settings



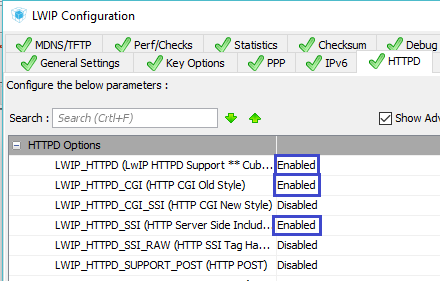
Then Key Options



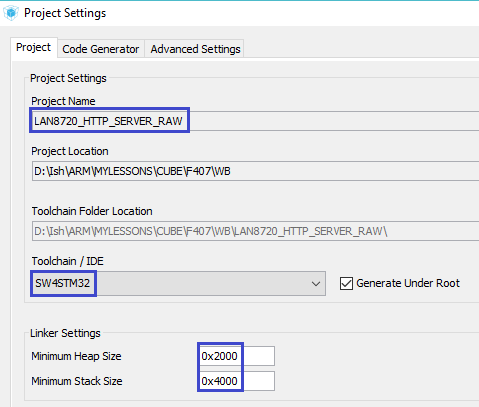


The settings are all the same, as in the lesson on TCP.

Now go to the **HTTPD** tab and **enable** it, and also enable **CGI** and **SSI** support



Let's go into the project settings, increase the stack and the heap and configure the generation of the project for **System Workbench** , well, give our project a name



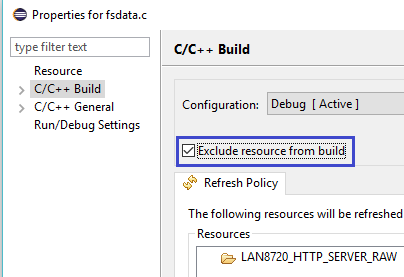
Apply the settings, generate the project, go to System Workbench, connect our generated project there, go into its settings, remove the entire debug configuration, if any, and set the optimization level as usual to  **1** .

Let's try to collect our project.

Only it is unlikely to gather, as it will require the presence of the file **fsdata.c** . This is because the HTTP server is working with documents, and therefore with the files. And we did not connect any media and file systems. It turns out, for this purpose it is provided storage of image files in the flash memory of the controller just like this through such a file. How this file is formed, we most likely remember from previous sessions on HTTP. And it is formed using the utility **makefsdata** , which is easy to find, it is only an executable file. It can also require the existence of a msvcr runtime library, which can also be provided to the utility as a dynamic link library file, which can be put into the utility folder.

Open the project folder, and in it the folder on the path " **Project folder / Middlewares / Third\_Party / LwIP / src / apps / httpd /** " and copy the utility and, if required, the **msvcr100d.dll** file into it. We will also create a folder named " **fs** " in this stick and copy all the server files we need into it-web pages, pictures, and so on. Just make sure that all this together does not take up too much size, since the flash memory of the controller is not rubber. Now **run the makefsdata.exe** file and we will **create the** file **fsdata.c** we **need**. And how to connect real files, using the media and file system in the RAW API, I, unfortunately, have not figured it out yet. But I think we will have enough of this to sort out the HTTP server a little, and not very much.

Let's return to the project, we will do Refresh, you can do this by selecting the project name in the project tree and pressing the " **F5** " key . Only now we have a project the more will not get together. It is important to know that the fsdata.c file should be seen, but should not be compiled. To do this, we call the context menu on it by right-clicking on it and entering its properties. Select **C / C ++ Build** and check the box next to " **Exclude resource from build** "



We will keep the changes and now we are likely to get the project together. Only if we project our project, then our server will not even ping.

Therefore, we'll go to main.c and start by connecting the necessary libraries

/\* USER CODE BEGIN Includes \*/

**#include "lwip/apps/httpd.h"**

**#include <string.h>**

/\* USER CODE END Includes \*/

In the **main ()** function, we initialize the server

/\* USER CODE BEGIN 2 \*/

**httpd\_init();**

/\* USER CODE END 2 \*/

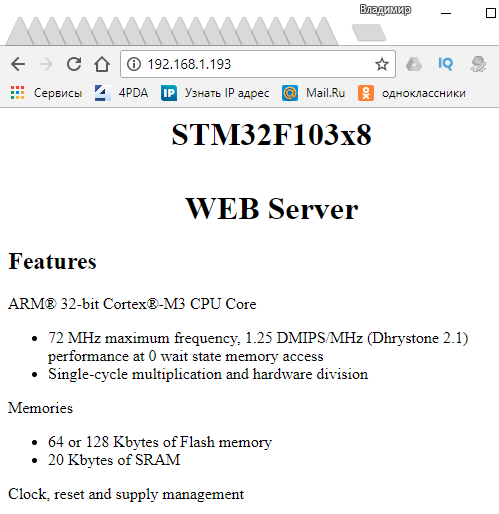
And in an infinite loop, we call the corresponding function

  /\* USER CODE BEGIN 3 \*/

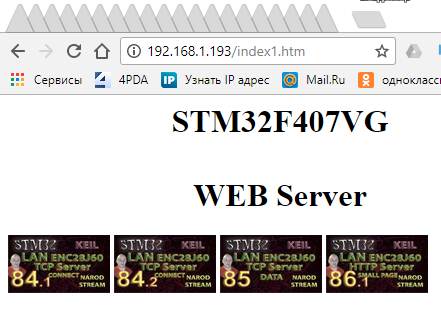
**MX\_LWIP\_Process();**

}

We will collect the code, we will tell the controller, in the browser in the address bar we will enter the IP-address of our server, press "Enter" and we will open the main page



Also try to open a page with links to several pictures. I inserted only 4. since we only have 5 sockets. Well, I think that's enough for us



That's how it all turns out, just when we use LWIP.

It would seem that this could be calmed down, but we are not the kind to get rid of us so easily. It turns out that our server can also work with passing parameters to the browser and from the browser, that is, it supports **CGI** and **SSI** . Well, how can you get around this?

First we will try to send some parameters from our server to the browser. Parameters can be any number. In the example in the repository, the parameter was only one, so uninteresting for us. We of course want to send a few. To do this, we created a special page, in which the data will come to certain places. These specific places in the web document are marked specifically in the form of a comment and enclosed in them by some character string or even one letter. Here on this marker server will fill the page with data and then send it. The disadvantage of CGI and SSI is that every time the data comes, the server has to send the entire page. Our pages will be small, without pictures, so it's uncritical. And if the pages are large, heaped up with styles and text, then there is for this technology AJAX, which we can not use in RAW, but in Netcon we will probably try it someday. Although it provides for PHP support on the server, but we'll somehow try to get around this. Well, while CGI and SSI. Ah, I almost forgot. The extension of documents for SSI should be**shtml** otherwise nothing will work. This is how the code for our page **counter.shtml**

**counter.shtml**

I selected a tag in which we set the period through which the page will be updated, and hence the data, we will have 1 second. It is possible less often, more often it is impossible.

comments with markers. That's where the parameter values ​​will be inserted. We will organize a regular counter that will be incremented each time a parameter is received to send a client to the handler.

Let's return to the code and add some global arrays and variables

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

**char const\* TAGCHAR[]={"p","r","s","t"};**

**char const\*\* TAGS=TAGCHAR;**

**uint32\_t n=0;**

/\* USER CODE END PV \*/

We have added an array with a string of markers, which we have only one letter at a time, and several, but enough for one, also added a pointer to this array. Well, also added a variable for the counter.

Let's add a handler for **SSI**

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

**uint16\_t SSI\_Handler(int iIndex, char \*pcInsert, int iInsertLen)**

**{**

**return 0;**

**}**

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

Let's write the body of this handler

uint16\_t SSI\_Handler(int iIndex, char \*pcInsert, int iInsertLen)

{

**if (iIndex ==0)**

**{**

**n++;**

**sprintf(pcInsert,"%lu",n);**

**return strlen(pcInsert);**

**}**

**else if (iIndex ==1)**

**{**

**sprintf(pcInsert,"%lu",n+5);**

**return strlen(pcInsert);**

**}**

**else if (iIndex ==2)**

**{**

**sprintf(pcInsert,"%lu",n+10);**

**return strlen(pcInsert);**

**}**

**else if (iIndex ==3)**

**{**

**sprintf(pcInsert,"%lu",n+15);**

**return strlen(pcInsert);**

**}**

  return 0;

We check the input parameter **iIndex** , which contains the marker index, which are considered in the order they are located in the global string array. And we process each index by returning a string with an account number. Since each page update will generate as many occurrences in this handler as we have markers, we increment the counter in only one of them, and then we convert the value of the account variable to a string, and in the rest the value of the variable is increased by a certain amount. And then we have to return the length of our string.

The input parameters, I hope, are understandable.

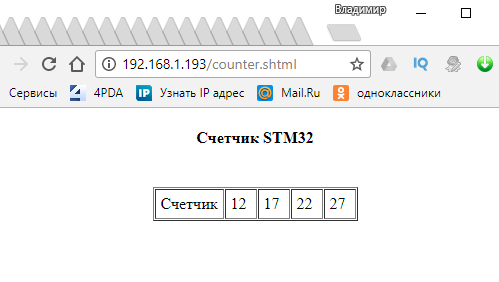
The handler is ready.

Now in **main ()** we declare a pointer to our handler using a special function

httpd\_init();

**http\_set\_ssi\_handler(SSI\_Handler, (char const \*\*)TAGS, 4);**

Now we can collect the code, flash the controller and admire the result



Our counter counts a second synchronously in each cell of the table, that is, they are transferred and placed where necessary in the HTML document (or rather SHTML) all our 4 parameters transmitted from the program.

Excellent!

In the [**next part of the**](http://narodstream.ru/stm-urok-102-lan8720-lwip-http-raw-chast-2/) lesson, we will study and try the CGI technology in practice, and then also try their common use to achieve some goals.

**STM Lesson 102. LAN8720. LWIP. HTTP RAW. Part 2**

Posted on [December 27, 2017](http://narodstream.ru/stm-urok-102-lan8720-lwip-http-raw-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/)- [No Comments ↓](http://narodstream.ru/stm-urok-102-lan8720-lwip-http-raw-chast-2/#respond)

[Programming for beginnersDownload the free book and become a real programmer. Get it now:To learn moresheremetev.info](https://an.yandex.ru/count/RWIDXe5LR2C50Du1CHMzari00000ECgs7402I09Wl0Xe172WmvR10O01vE3CgmU80VY6tP0fa06YakID99W1Zhs-tYEW0VwIp8qag06M_w_U8xW1XgMPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y083e0BUkAe2kG9x0XI9yC-hBV02cUkBbmRu0eA0W820a1I00y-UfQuDY0Ezp-qMc0E9nGIe0nAm0mIu1Fy1w0IF1_W4XvG1Y0M7b06G1QUJ0g05jIoe1T8bi0NI9RW5_qN01V2P1yW5iEe1q0Nu4E05Ml050PW6WD2iymwW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000Aa7-568Pukyn4Ym1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08hvoL0O0A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i6gWiGC0NRzVXH001kT_4FMtK50C0BWAC5o0k0r9C1sGluKOXdYxp4IEWBXvG1y0i6Y0osbTw-0UWC0fWDbOneyWq0-Wq0WWu0YGu00000003mFv0Em8Gzc0x6rypKbOdWkXEW3i24FR0E0Q4F00000000y3-e3_YZWSAjaQ_YI000?stat-id=3&test-tag=88510715420673&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY1IjoiODg1MTA2ODYwNjg3MzYifQ%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/4o0ARb4F2UC50D81CHQzari00000ECgs7402I09Wl0Xe173MZO-53e01q8gZjGE80UMTcjGqa06KnkkWBvW1wBlk-IsW0Oh6ww0lg06Gk-xvBRW1w9tytXd00GBO0OgrwnJW0TIzXWVe0QO2-07czzw-0OW21A02hCMi3xa2Um8KYV3Fgotm0iJLcPK9-0A2W820W90Kc0EHAB031BW4_m7e1DO2-0IsbmI81RQN1905Wxq6e0NGe06e1P280R05a8W1k0NYqW701Pkq6iW5iDi6q0MWFU05F_050PW6hesqW0wW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000Aa7-56G1Cgyn4Ym1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08rgBM1u0A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i8gWiG1uV6aFXH002p7W4GMtK50C0BWAC5o0k0r9C1sGluKP04ohp4IEWBjfS4y0i6Y0pczzw-0UWC0fWD_BrcyWq0-Wq0WWu0YGu00000003mFv0Em8Gzc0x6rypKbOdWkXEW3ioe3B0E0Q4F00000000y3-e3_YZmPQKcQ_YI000?stat-id=4&test-tag=88510715420673&format-type=24&banner-test-tags=eyI2MzQyNjY4NjgyIjoiODg1MTA2ODYwNjg3MzYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)18+

In the [**previous part of the**](http://narodstream.ru/stm-urok-102-lan8720-lwip-http-raw-chast-1/) lesson we created and configured the project, wrote and checked the WEB server code, and also wrote and checked the code for using SSI.

Now, on the contrary. We need to transfer data from the client browser to the server. This is already CGI. We will, as in the example from the repository, manage the LEDs, including certain ones, depending on the checkboxes in which we will tick.

For this we already have another document **led.html**

**led.html**

In this document there is a form with four checkboxes and a button. When the button is clicked, the parameters described in each input, in which the daw is installed, will go to the server from the browser.

To make this work, go to the project and add some more global variables and a prototype

uint32\_t n=0;

**const char \* LEDS\_CGI\_Handler(int iIndex, int iNumParams, char \*pcParam[], char\*pcValue[]);**

**const tCGI LEDS\_CGI={"/leds.cgi", LEDS\_CGI\_Handler};**

**tCGI CGI\_TAB[1];**

**uint8\_t ledstate=0;**

We added a prototype of the parameter handler from the browser, which also contains a line with a file that is not actually there, just this will be the beginning of the browser's GET request, as well as the address of the function in which this request will be processed. Also a special array of one element and a variable for storing the status of the LEDs.

Add a request handler after the SSI handler

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

**const char \* LEDS\_CGI\_Handler(int iIndex, int iNumParams, char \*pcParam[], char\*pcValue[])**

**{**

**uint32\_t i=0;**

**if (iIndex==0)**

**{**

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

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

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

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

**ledstate = 0;**

**for (i=0; i<iNumParams; i++)**

**{**

**if (strcmp(pcParam[i] , "led")==0)**

**{**

**if(strcmp(pcValue[i], "1") ==0)**

**{**

**ledstate |= 1;**

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

**}**

**else if(strcmp(pcValue[i], "2") ==0)**

**{**

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

**ledstate |= 2;**

**}**

**else if(strcmp(pcValue[i], "3") ==0)**

**{**

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

**ledstate |= 4;**

**}**

**else if(strcmp(pcValue[i], "4") ==0)**

**{**

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

**ledstate |= 8;**

**}**

**}**

**}**

**}**

**return "/led.html";**

**}**

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

In this handler, we check the **iIndex** parameter to zero, but we will not have another one, since we only have one parameter, and the values ​​can be different. And in the values ​​we already have a line, part of which we will check at once, and then we will check the figure and if one of the four comes, then we light the corresponding LED and also turn on the definite bit in the status variable. We do not particularly care about this status, it will be later. And in the end, we return the result as a page.

But that's not all. It is necessary to do something else in **main ()**

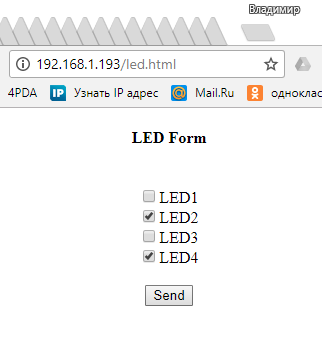
http\_set\_ssi\_handler(SSI\_Handler, (char const \*\*)TAGS, 4);

**CGI\_TAB[0] = LEDS\_CGI;**

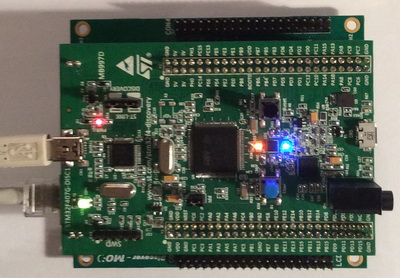
**http\_set\_cgi\_handlers(CGI\_TAB, 1);**

We initialized the array element and the handler.

Now you can collect the code, flash the controller and test the code



Install the daws in any checkboxes and click the "Send" button. We will have Dodge to light up the corresponding LEDs



Only I did not like one thing. After clicking the button and restarting the document, the daws are removed, I would like them to remain in place. I have long thought about how to implement this. But I found a way out. I've used SSI here, which will return all we need as a parameter. For this, the status was invented.

Call in the browser we will already have another document - led.shtml with the following code

**<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "**<http://www.w3.org/TR/html4/strict.dtd>**">  
<html><head><title>LED</title>**

**<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">  
<meta content="MSHTML 6.00.2800.1561" name="GENERATOR">  
<style ="font-weight: normal; font-family: Verdana;"></style></head>  
<body>  
<h4  style="text-align: center;">LED Form</h4>  
<br>  
<form align="center" method="get" action="/leds.cgi">  
<!--#p--> LED1<br>  
<!--#r--> LED2<br>  
<!--#s--> LED3<br>  
<!--#t--> LED4<br>  
<br>  
<input value="Send" type="submit"> </form>  
</body></html>**

In this document, we already otstvuyut tags and instead they are comments with markers.

Only markers with comments in the body of the tags could not be added, so we will transfer the entire tag.

The point is that in order for the checkbox to be a dummy, we must insert the property into the input tag as an attribute without a parameter. This is a **checked** attribute .

Let's return to the project and add two more global string arrays that represent the parts of the line with the tag that will not change

uint8\_t ledstate=0;

**char htmstr1[] = "<input value=\"";**

**char htmstr2[] = "\" name=\"led\" type=\"checkbox\"";**

In the SSI **SSI\_Handler** handler, comment out all the code except the last line with a return of zero length and add another code

**SSI\_Handler**

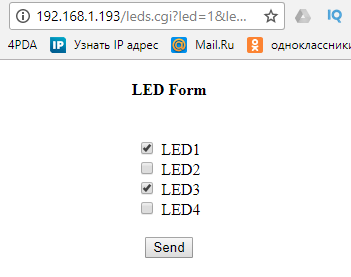
The code here, though large, but fairly simple. We first enter the beginning of the tag in the line, then go into the condition corresponding to the query index, add to the line in the body of the condition the index corresponding to the index, then add the next part of the tag. Then, depending on the state of the bit assigned to the LED, we will end the tag. If the bit is one, that is, the LED is lit, then we fall into the true body and add the **checked** attribute corresponding to the installed daw and close the tag, and if at zero, we simply close the tag. That's all.

But no, actually not quite everything. In function **LEDS\_CGI\_Handler we** do not forget to change the document extension when returning, otherwise we will return the wrong document

  return "/led.**s**html";

}

We collect the code, flush the controller and look at the result, invoking the browser of course **led.shtml**



The code works fine!

We also manage LEDs and daws already from the checkbox when reloading the page does not disappear.

Thus, today we have learned how to create a simple **HTTP** server, and also learned how to use the functionality of this server to work with **CGI** and **SSI** , and also simultaneously with both.

Thank you all for attention!