**STM Lesson 101. HC-05. Transfer the temperature**

Posted on [December 19, 2017](http://narodstream.ru/stm-urok-101-hc-05-peredayom-temperaturu/)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-101-hc-05-peredayom-temperaturu/#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/D7M29VjsoYq50Dy1CTgyari00000ECgs7402I09Wl0Xe172WmvR10O01sE3CgmU80QE8tP0fa06cakID99W1eiM-tYEW0SAapOqag07iWR3U8xW1tAUPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y083e0BUkAe2kGAu3Lbb1XSIll02tvNCoGdu0eA0W820a1E00-RYgP81Y0EIhhwt29W3ky06g0CIi0C4k0J_0UW4lGFu1B_M0OW5lzO1a0NVnGAW1Qn0g0N4Dh05n3Qu1UXIm0NmcGV81R3g0T05nHZW1GRm1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XHe8LphiH8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2A-SbG602W712W0000000F0_s0e2u0g0YNhu2i3y5OWB1geB42EHSeNuKG00-Nvh35jr1G302u2Z1SWBWDIJ0TaB-56WXNEkn4Ze2x_M0V0B1eWCjfNUlW7e30AO3PMCQF8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_ue-6bfywhuaW0?stat-id=3&test-tag=84112668909569&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY3IjoiODQxMTI2Mzk1NTc2MzIifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Buy a robot carTeach your child to program. 6900 for a **set** of shares. OrderTo learn moredorado-uno.ru](https://an.yandex.ru/count/RXcP55XN7zC50DO1CTgyari00000ECgs7402I09Wl0Xe172AffoG1e01-SAy88W1phMagJMG0OZBnCelc07WZ-3LBg01Xil4oY-e0TwEuDKkk07it_Rr6S010jW1ieMB5U01qBto4UW1OFW1elhUlW680WIW0g2-sHQv0hWDMMK65nA-y0BAfAlc3_W2We20W82G4u03cUUvmGE80wU0d8y6c0Flem6m0mJe1CW2-0JHZWA81T6E0f05cl02e0MOKQW5vqIm1UT4k0NENS05iFiCo0NmfGFG1ViUu0LQy0K1c0QEkx0Pe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XHJF2JhiH8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW293Vh0k02W712W0000000F0_s0e2u0g0YNhu2i3y5OWB2AeB4Bqrrc3uKG00bPbp35jr1G302u2Z1SWBWDIJ0TaB-55Cy9Ekn4Ze2z6E0l0B1eWCelhUlW7e30AO3VozPl8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_ue_7khzAhuaW0?stat-id=4&test-tag=84112668909569&format-type=24&banner-test-tags=eyI2Mzg2Mzg5NzAwIjoiODQxMTI2Mzk1NTc2MzIifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

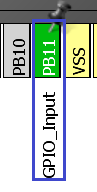
We continue to work with the module **HC-05** , which provides data transfer without wires via Bluetooth.

We have already learned how to transfer data from one controller to another with the help of two such modules, and today we will try to transfer more conscious data. We have already become a tradition to transfer the temperature from the sensor DS18B20 and we will not break it yet. And the library at us was written under the same payment, what is used at us on the slave device. But since this task, I think, will not make much work for us, then, so that the lesson was not too short and simple, I had another idea. We will try on the host device from time to time to know the status of our connection. Maybe this is not very important information for us - the status of the connection, but we will learn how to send commands and receive answers to them, and then process these responses, not from the terminal program, as in previous studies, namely from the controller.

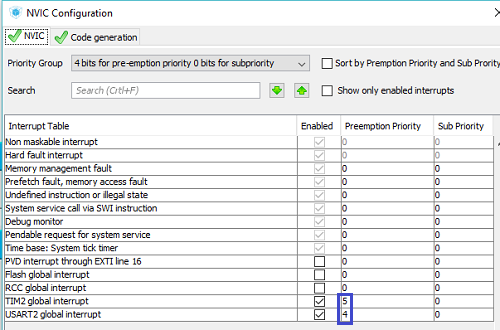
So let's not wear out ourselves with anticipation and finally get down to business.

Let's start with the slave device. The project we will borrow the former from [**lesson 97**](http://narodstream.ru/stm-urok-97-podklyuchaem-bluetooth-modul-hc-05/) and assign it the name **HC\_05\_SLAVE\_DS18B20** , since we will connect to it exactly such a sensor as we have attached by means of a suffix.

Open our project in the **Cube MX** and first of course we will turn on the foot to which we will attach our temperature sensor. We remember from the [**lesson on the sensor**](http://narodstream.ru/stm-urok-92-datchik-temperatury-ds18b20-chast-1/) that we do not care about the direction, we need to have at least some initialization, and that the output will be involved, we will see and we will not forget about it afterwards. Include, for example, the leg PB11



Let's go into Configuration, open the NVIC section and distribute the priorities of the interrupts from USART and the timer, they are so much better "agree" with each other



Generate our project, open it in Keil, configure the programmer for auto-reload, and also enable optimization level 1 and try to assemble it.

If everything was assembled without errors, then we will copy the files from the project  [**lesson 94**](http://narodstream.ru/stm-urok-94-ds18b20-podklyuchaem-neskolko-datchikov-na-provod-chast-1/)  on the temperature sensor **ds18b20.h**  and  **ds18b20.c**  into the corresponding folders of our new project.

We connect the file **ds18b20.c** to the project  **tree** and also connect our library in the **main.h** file

/\* USER CODE BEGIN Includes \*/

**#include "ds18b20.h"**

/\* USER CODE END Includes \*/

Also for work with the sensor we will add a number of global variables and arrays in the file **main.c**

  "String5\r\n"

};

**uint8\_t Dev\_ID[8][8]={0};**

**uint8\_t Dev\_Cnt;**

**uint8\_t dt[8];**

**uint16\_t raw\_temper;**

**float temper;**

**char c;**

Now the project will at least be assembled.

Initialize our sensor in **main ()**

/\* USER CODE BEGIN 2 \*/

**port\_init();**

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

HAL\_UART\_Receive\_IT(&huart2,(uint8\_t\*)str1,1);

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;

}

That is, we now consider the readings in the register as a significant result, 16 times larger than in fact, and that's it.

In the prototype, we also correct the type of the input argument.

**We go to main.c** In the procedure of interrupting the timer, the code also changes

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

{

  if(htim==&htim2)

  {

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

**i++;**

**if(i>10)**

**{**

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

**{**

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

**}**

**else if(i%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 \r\n", temper);**

**else sprintf(str1,"%.2f \r\n", temper);**

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

**}**

**if(i==252) i=12;**

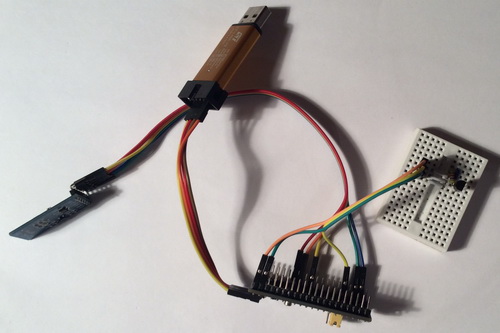
**}**

  }

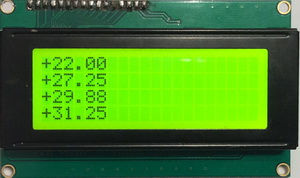
}

Here we have the following. We at a certain moment of time instruct the sensor to start the temperature conversion, and in about a second we take the readings from it, process it appropriately in a readable form, and immediately in a legible form send it to the USART, and therefore into the air. The temperature display code itself has also changed, since we no longer use the sign detection function, since the temperature already comes from the **ds18b20\_Convert** function  with the sign. The only thing, if there is a positive value, then we substitute a plus sign and that's it.

Our scheme has not changed in practice, only the only thing, we connected to it a temperature sensor, which has long been used to our board. While we connect a conventional unprotected sensor for convenience, and if we need to measure for example a negative temperature outside the window and pass it through the air into the room, then perhaps you should connect the sensor on the wire and in the metal screen. This I also have if that. Here is our scheme



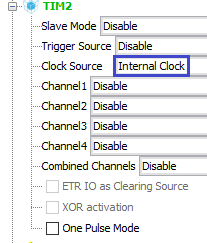
We connect the programmer of this circuit to the USB port of the computer, collect and sew our project. Also, we connect the circuit with the master module and the display, feeding it in turn from an independent source (power supply with USB-out or from the POWER-bank, now the display should show the temperature coming from about 2 seconds from the slave device. that it is updated by touching the temperature sensor.The temperature will increase at the same time



Everything works great. It is now possible to connect the slave's ST-Link to an independent scribe, and the main module to the USB port of the computer, and to implement the second task of monitoring the status of the connection, and some more aesthetic ideas.

The project will be created on the basis of project **lesson 99** for the host device and is named **HC\_05\_MASTER\_AT** .

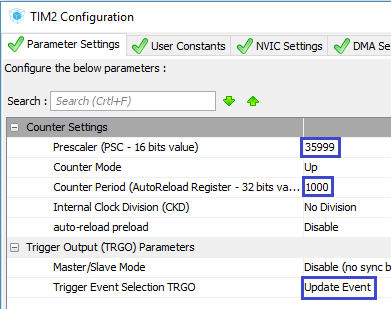
Open our project in the Cube MX and first turn on our favorite timer 2



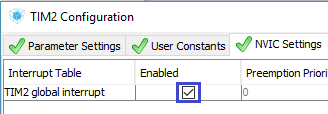
Also, turn on the **PA12** foot to which we connect the **KEY** foot of the HC-05 module to control the mode of receiving and sending **AT** commands



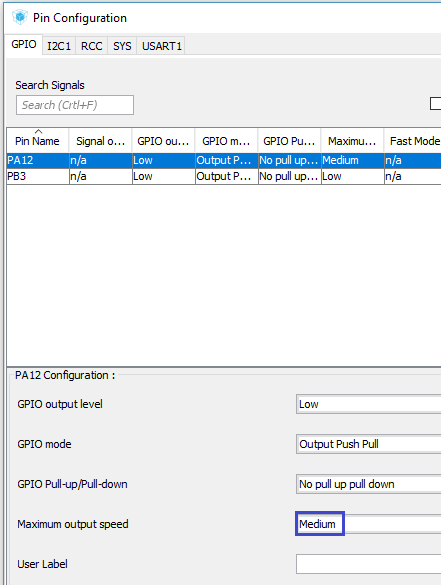
Now go to **Configuration** and start setting up the timer



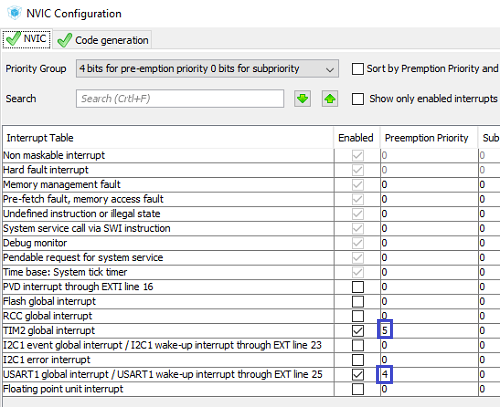
Enable interrupts



Let's add a bit of speed to the port foot



Then, configure the priority of the timer interrupts and the USART port of the alias with the slave



Generate the project, open it in Keil, configure the programmer for auto- **reload** , set the optimization level to 1 and connect the file **lcd.c** to the project. Let's try to assemble the project. If the project is properly assembled, then we start to compose the code.

First, remove the output of test strings from **main ()**

~~sprintf(str1,"String 1");~~

~~LCD\_StrBottom(str1);~~

~~sprintf(str1,"String 2");~~

~~LCD\_StrBottom(str1);~~

~~sprintf(str1,"String 3");~~

~~LCD\_StrBottom(str1);~~

~~sprintf(str1,"String 4");~~

~~LCD\_StrBottom(str1);~~

Let's go to the file lcd.c and add one more function that will output the string not to a specific place in the line, but just to a certain line, it will also blank the remaining characters in the line to erase the information from the line that was before that . We will add this function above the function **LCD\_StrBottom**

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

**void LCD\_StringAtLine(uint8\_t y, char\* st)**

**{**

**uint8\_t i=0, str\_cnt=0;**

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

**while((st[str\_cnt]!=0)&&(st[str\_cnt]!=0x0D)&&(st[str\_cnt]!=0x0A))**

**{**

**sendbyte(st[str\_cnt],1);**

**str\_cnt++;**

**}**

**//остальное - пробелы**

**for(i=str\_cnt;i<20;i++)**

**{**

**sendbyte((uint8\_t)' ',1);**

**}**

**}**

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

In general, the code here is simple and in the explanation, I think, does not need.

Create a prototype for this function in the header file and return to the **main.c** file .

We will increase the buffer for USART a bit, because the answers to the commands may be longer than desired, we will not deduce them entirely, but we are obliged to accept them

uint8\_t usart\_buf[**40**];

We also add some more global variables, an array and an enumeration for the connection status, as well as for timer tick counting

USART\_prop\_ptr usartprop;

**enum bt\_states**

**{**

***BT\_NONE* = 0,**

***BT\_INITIALIZED*,**

***BT\_READY*,**

***BT\_PAIRABLE*,**

***BT\_PAIRED*,**

***BT\_INQUIRING*,**

***BT\_CONNECTING*,**

***BT\_CONNECTED*,**

***BT\_DISCONNECTED***

**};**

**\_\_IO uint8\_t bt\_state = *BT\_NONE*;**

**char str\_state[14]="NONE";**

**volatile uint32\_t Tim2Cnt=0;**

Run the timer in **main ()**

HAL\_UART\_Receive\_IT(&huart1,(uint8\_t\*)str1,1);

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

Add an event handler from the timer

    UART1\_RxCpltCallback();

  }

}

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

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

**{**

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

**{**

**if(Tim2Cnt>9)**

**{**

**if(Tim2Cnt%10==0)**

**{**

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

**}**

**if(Tim2Cnt%10==1)**

**{**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)"AT+STATE?\r\n",11,0x1000);**

**}**

**if(Tim2Cnt%10==3)**

**{**

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

**}**

**}**

**Tim2Cnt++;**

**}**

**}**

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

/\* USER CODE END 4 \*/

The code here, though it seems complicated at first glance, but in reality everything here is simple. Count ten ticks, so let's say the modules agree, then every ten ticks (or 5 seconds approximately), we feed the plus potential on the KEY foot, thereby causing the module to accept commands. When the modules are connected, it is not enough simply to supply the potential and immediately remove it. In order for the module to respond to commands, it is necessary to keep the power level on this leg at this time. And already the data coming to the module in this state will not enter the port, but they will be stored in the buffer, and when we finish working with the commands and set the level 0 on the KEY foot, they all will normally arrive safely from the buffer to the port , so do not be afraid of losing the data coming through the air. Then we wait half a second and send the status request command,

Now let's process the answer.

To do this, we will first slightly redesign the code in the **UART1\_RxCpltCallback** function . Well, rather, not for this, but due to the fact that we have increased the buffer and that we all accept it, but then do not send everything to the LCD line. First of all, we increase the maximum length of the buffer

if (usartprop.usart\_cnt>**39**)

And that the excess is not displayed on the display, at the end of the reception, if it turned out more than necessary, put zero in the appropriate place, thereby denoting the end of the line

string\_parse((char\*)usartprop.usart\_buf);

**if (usartprop.usart\_cnt>19)//у нас строка на дисплее 20 символов**

**{**

**usartprop.usart\_buf[20] = 0;**

**}**

usartprop.usart\_cnt=0;

Well, accordingly, the parsing function of the string has increased seriously

void string\_parse(char\* buf\_str)

{

**if(!strncmp(buf\_str,"+ST",3))**

**{**

**if(!strncmp(buf\_str+7,"CONNECTE",8))**

**{**

**bt\_state=BT\_CONNECTED;**

**strcpy(str\_state,"CONNECTED");**

**HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_3, *GPIO\_PIN\_SET*);**

**}**

**else**

**{**

**if(!strncmp(buf\_str+7,"INI",3))**

**{**

**bt\_state=BT\_INITIALIZED;**

**strcpy(str\_state,"INITIALIZED");**

**}**

**else if(!strncmp(buf\_str+7,"RE",2))**

**{**

**bt\_state=BT\_READY;**

**strcpy(str\_state,"READY");**

**}**

**else if(!strncmp(buf\_str+7,"PAIRA",5))**

**{**

**bt\_state=BT\_PAIRABLE;**

**strcpy(str\_state,"PAIRABLE");**

**}**

**else if(!strncmp(buf\_str+7,"PAIRE",5))**

**{**

**bt\_state=BT\_PAIRED;**

**strcpy(str\_state,"PAIRED");**

**}**

**else if(!strncmp(buf\_str+7,"INQ",3))**

**{**

**bt\_state=BT\_INQUIRING;**

**strcpy(str\_state,"INQUIRING");**

**}**

**else if(!strncmp(buf\_str+7,"CONNECTI",8))**

**{**

**bt\_state=BT\_CONNECTING;**

**strcpy(str\_state,"CONNECTING");**

**}**

**else if(!strncmp(buf\_str+7,"DI",2))**

**{**

**bt\_state=BT\_DISCONNECTED;**

**strcpy(str\_state,"DISCONNECTED");**

**}**

**else**

**{**

**bt\_state=BT\_NONE;**

**strcpy(str\_state,"NONE");**

**}**

**HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_3, *GPIO\_PIN\_RESET*);**

**}**

**}**

**else if(buf\_str[0]!='O')**

**{**

**LCD\_StringAtLine(3,buf\_str);**

**LCD\_StringAtLine(0,str\_state);**

**}**

**else**

**{**

**}**

}

We first found out that it was not an ordinary line. but the response to the status request, and then in the body of this condition, we already parse the string starting from the character where the status information comes exactly and take the required number of characters from this line and compare them with the model ones, and if it coincides with some status , we already assign the same status to the variable and prescribe in the string buffer a string with the status that we will display on the display when we display the incoming line with the temperature sensor readings. If we display a row with the status at once, then we risk not having time to process the next line. You can check if there are glitches. If this is not an answer with status, then check further. Let's see that there was no ' **O** ' to not show the line with the answer **OK**, since it also comes, and if it is not it, then we simply display the temperature readings in the bottom line, and the status line in the upper line.

We will collect the code, we will sew the crystal and see the result of our work. When the connection state is normal, it should be like this

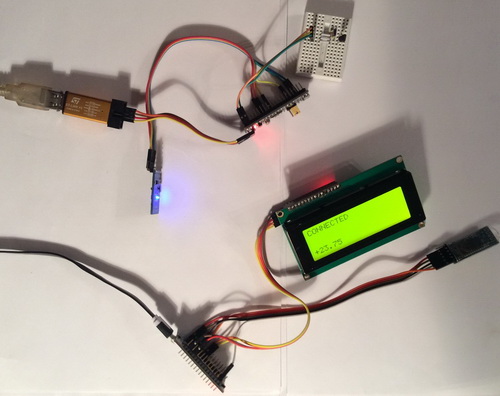


You can work with other statuses by turning off the slave sometimes. There may be some glitches. Not always then there will be a normal return to normal after reconnection, but basically everything is normal.

Also I will show how our thermometer measures the negative temperature



This is how our entire receiving and transmitting complex looks now (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2017/12/stm101img10.jpeg)

Most importantly, we now, firstly, transfer not any abstract information but information that is aware of the temperature readings, and secondly, we are able during normal work to obtain information about the state of our connection and other information from the module, implementing the ability to programmatically work with sending AT commands and receiving and processing responses to them.

Thank you for your attention!