**STM Lesson 117. NRF24L01. Information exchange**

Posted on [May 5, 2018](http://narodstream.ru/stm-urok-117-nrf24l01-obmen-informaciej/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Published in [SPI](http://narodstream.ru/spi/) , [Programming STM32](http://narodstream.ru/rub_stm32/)- [2 comments ↓](http://narodstream.ru/stm-urok-117-nrf24l01-obmen-informaciej/#comments)

[Earn remotely, working 2-3 hours.**Free** course "Specialist in Internet Advertising" for 10 days! Sign up before 10.09.18To learn moreprofiinternet.artur-grant.ruYandex.Direct](https://an.yandex.ru/count/Oo6vpnPOEXy50E01CUc-ari00000ECgs7402I09Wl0Xe173qmTYm0e01clpa0eW1YSEGWp6G0PAFlOyjc06sygd2Ag01b8-zZose0UZkgS8gk06qaQRQ6C010jW1YDIE4k01kgkO1kW19FW1lhtUlW680WEW0gpkd16v0gXtyN_cUSbay0AAmvht1FW2We20W83m4e03xQd0K8W3kgYVq0UO0uaXg0CIi0C4k0J_0UW4ZGBu1FhQ1OW5-je5a0NcbmQW1QNR0QW5wxa1i0NhkG6u1QV90S05zO0Oo0MWZGRG1T1Ju0K-y0K1c0Q0qApp3g06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00071n0000gGVuKKmx3Yp5IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0YMjyu8W0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mQg2n1AGDjO-54005GiqYjRTGK0m0k0emN82u3Kam7P2_XHJ3iEBCL8w0lwsWNm2mQ83Bwzthu1w0m2c0sLZ6Zo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Em8Gzi0u1eGy00000003mFwWF-AE1uTU1o-98?stat-id=3&test-tag=83013157281793&format-type=24&banner-test-tags=eyI2MDU2MDI3MDgxIjoiODMwMTMxMjc5Mjk4NTYifQ%3D%3D&)

[Programming for beginnersDownload the free book and become a real programmer. Get it now:To learn moresheremetev.infoYandex.Direct](https://an.yandex.ru/count/TUxcfwOYSPS50Dm1CUc-ari00000ECgs7402I09Wl0Xe172WmvR10O01vE3CgmU80VY6tP0fa06YakID99W1Zhs-tYEW0VwIp8qag06M_w_U8xW1XgMPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y084e0BUkAe2kGAeT_5_vdd9PF02ay3axGRu0eA0W820y1A00xUCwx44Y0FHWRVK0PW3YSK4i0C4k0J_0UW4sWRu18A_0OW5Why1a0NobmAW1QSvg0N8CB05o32u1PH7m0NmcGV81R3g0T05xHNW1Jxm1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XH3AmpBCL8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2A-SbG602W712W0000000F0_s0e2u0g0YNhu2i3y5OWB2AeB40If5uFuKG00WeZRArjr1G302u2Z1SWBWDIJ0TaB-54Ch3CinKZe2uA_0V0B1eWCjfNUlW7e30AO3VozPl8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_ue-6hhuRBuaW0?stat-id=4&test-tag=83013157281793&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY1IjoiODMwMTMxMjc5Mjk4NTYifQ%3D%3D&)

We continue to work with the wireless transceiver  **NRF24L01** . And today we will also be engaged in the transmission of information, but we will teach and receive and transfer each of our modules. I think that this will be an actual topic, since we often want to not just transfer some information to a remote device, but also get some response from it. That is, each of the modules must be able to be both a transmitter and a receiver. Thus, we organize a kind of half-duplex mode, which differs from full-duplex mode in that the latter can do this at the same time.

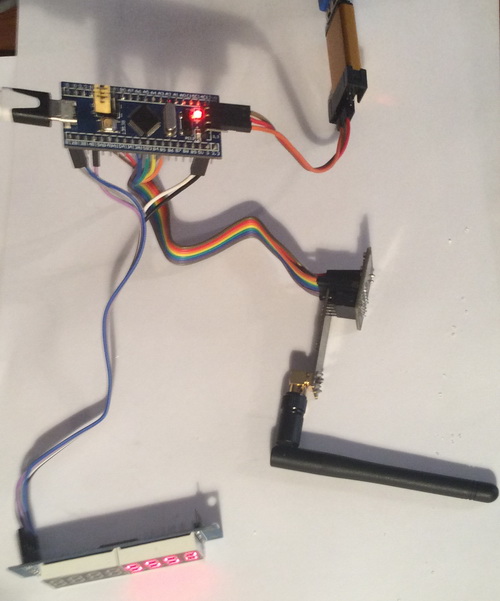
In principle, we already have in our library the functions of transferring the module either to the receiver mode or to the transmitter mode. So there is a beginning. But this does not mean that the code will be very easy. This is not so. But, feeling behind such a backlog of experience, we will definitely cope with all this. And we are not accustomed to difficulties.

The exchange will be organized as follows. We will have two communicating devices. After a certain time, one of the devices will transmit any packets. The other device will receive them and after a certain time it will respond to them also with a packet that the first device will also have to receive, plus every device on receiving and transmitting packets will have to report on the LED indicator.

As devices we will take inexpensive STM32F103 controllers located also on inexpensive debugging cards, especially since we already have projects for them and LED indicators are also connected to them.

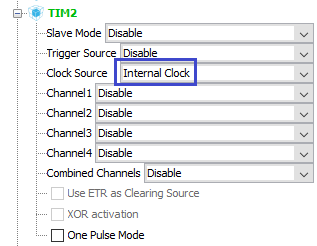
Also we will not disconnect the programmer from them any more. I remembered that I have an original ST-Link, and connected it to the 2nd device. And we will be constantly feeding both devices, so we will not connect with the cheap programmer, we will use only 3 wires, and on the original ST-Link we will either conduct 4 wires or connect it to the board with a 20-pin loop.

We connect for the time being only the scheme of the 1st device



We will create a project for this scheme from the [**lesson 115**](http://narodstream.ru/stm-urok-115-nrf24l01-neskolko-priemnikov-chast-1/) project for the first receiver **NRF24\_RX\_00**. We will **name** our new project **NRF24\_RXTX\_00** .

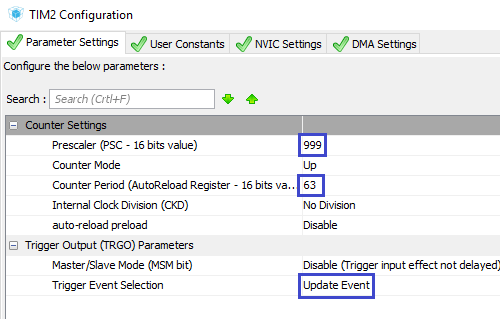
Run the project in Cube MX and turn on the timer, because we have it will also be a transmitter



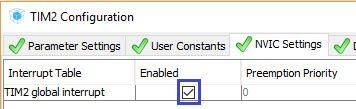
On the **PA2** foot, we now trace external interrupts from the module hardware, so we will translate it into the corresponding mode

http://narodstream.ru/wp-content/uploads/2018/04/stm117img05.png

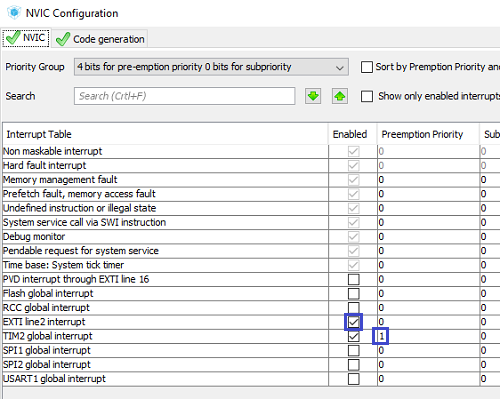
Go to **Configuration** and configure our timer



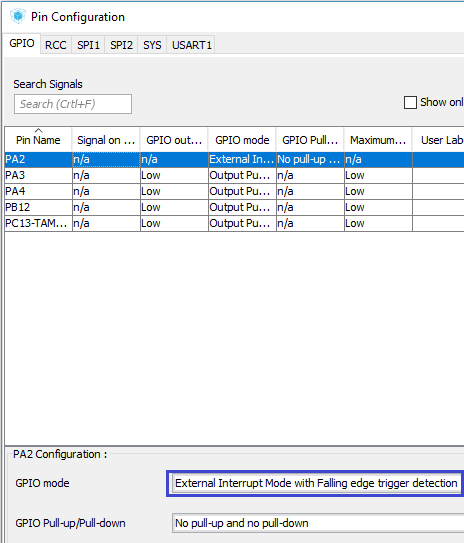
Enable interrupts



Go into the settings of NVIC and turn on there interruptions from the foot, and also reduce the priority of interrupts from the timer



Let's go to **GPIO** and set the interrupt trigger on the drop



Generate the project, run it in Keil, configure the programmer for autocut, enable the optimization level **1**, connect the files **NRF24.c** and  **max7219.c** .

Let's **go** to file  **NRF24.c** , **correct the** address of the transmitter a bit and add another one, because when it becomes a receiver, it will be necessary to change the addresses to its own

uint8\_t TX\_ADDRESS**0**[TX\_ADR\_WIDTH] = {**0xb7,0xb5,0xa1**};

**uint8\_t TX\_ADDRESS1[TX\_ADR\_WIDTH] = {0xb5,0xb5,0xa1};**

Let's add the size of the storage buffer

uint8\_t RX\_BUF[TX\_PLOAD\_WIDTH**+1**] = {0};

Also add flags for the received and transmitted packets

uint8\_t RX\_BUF[TX\_PLOAD\_WIDTH+1] = {0};

**volatile uint8\_t rx\_flag = 0, tx\_flag = 0;**

In the function of transferring to the receiver mode **NRF24L01\_RX\_Mode** we **will** add the address change code

NRF24\_WriteReg(CONFIG,regval);

**NRF24\_Write\_Buf(TX\_ADDR, TX\_ADDRESS1, TX\_ADR\_WIDTH);**

**NRF24\_Write\_Buf(RX\_ADDR\_P0, TX\_ADDRESS1, TX\_ADR\_WIDTH);**

And in the function of transferring to the mode of the transmitter **NRF24L01\_TX\_Mode,** we change the name of the address macro a little in the address change line, and also add the line for changing the address of the PIPE0 channel

NRF24\_Write\_Buf(TX\_ADDR, TX\_ADDRESS**0**, TX\_ADR\_WIDTH);

**NRF24\_Write\_Buf(RX\_ADDR\_P0, TX\_ADDRESS0, TX\_ADR\_WIDTH);**

In the initialization function, we change the name of the macro, adding to it a zero

NRF24\_Write\_Buf(TX\_ADDR, TX\_ADDRESS**0**, TX\_ADR\_WIDTH);

NRF24\_Write\_Buf(RX\_ADDR\_P0, TX\_ADDRESS**0**, TX\_ADR\_WIDTH);

After the initialization function, we add the function of interrupt processing from the stem. We wrote the same function for the receiver in [**lesson 115**](http://narodstream.ru/stm-urok-115-nrf24l01-neskolko-priemnikov-chast-1/)

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

**void IRQ\_Callback(void)**

**{**

**uint8\_t status=0x01;**

**uint8\_t pipe;**

**DelayMicro(10);**

**status = NRF24\_ReadReg(STATUS);**

**if(status & 0x40)**

**{**

**LED\_TGL;**

**pipe = (status>>1)&0x07;**

**NRF24\_Read\_Buf(RD\_RX\_PLOAD,RX\_BUF,TX\_PLOAD\_WIDTH);**

**\*(RX\_BUF+5) = pipe;**

**NRF24\_WriteReg(STATUS, 0x40);**

**rx\_flag = 1;**

**}**

**if(status&TX\_DS) //tx\_ds == 0x20**

**{**

**LED\_TGL;**

**NRF24\_WriteReg(STATUS, 0x20);**

**NRF24L01\_RX\_Mode();**

**tx\_flag = 1;**

**}**

**else if(status&MAX\_RT)**

**{**

**NRF24\_WriteReg(STATUS, 0x10);**

**NRF24\_FlushTX();**

**//Уходим в режим приёмника**

**NRF24L01\_RX\_Mode();**

**}**

**}**

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

Also, we'll add an interrupt function from the timer below, while empty

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

**void TIM1\_Callback(void)**

**{**

**}**

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

We add prototypes to our function-handlers and go to the file **main.c** , in which we will add an external interrupt handler from the stem

/\* USER CODE BEGIN 4 \*/

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

**void HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin)**

**{**

**if(GPIO\_Pin== GPIO\_PIN\_2)**

**{**

**IRQ\_Callback();**

**}**

**else**

**{**

**\_\_NOP();**

**}**

**}**

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

/\* USER CODE END 4 \*/

Below we add a function-handler from the timer, in which we simply call our function, which we added to our module library

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

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

**{**

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

**{**

**TIM1\_Callback();**

**}**

**}**

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

Also run our timer in **main ()**

NRF24\_ini();

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

Let's **go to the NRF24.h** file and delete the macros for the **IRQ** foot , since we already process this leg with hardware

~~#define IRQ\_GPIO\_PORT GPIOA~~

~~#define IRQ\_PIN GPIO\_PIN\_2~~

~~#define IRQ HAL\_GPIO\_ReadPin(IRQ\_GPIO\_PORT, IRQ\_PIN)~~

Let's **go** to file  **NRF24.c** and correct the function **NRF24L01\_Send** in the light of the hardware interrupt handling requirements. Practically, the code will remain the same, only with the treated interrupt processing

uint8\_t NRF24L01\_Send(uint8\_t \*pBuf)

{

**uint8\_t regval=0x00;**

**NRF24L01\_TX\_Mode(pBuf);**

**regval = NRF24\_ReadReg(CONFIG);**

**//если модуль ушел в спящий режим, то разбудим его, включив бит PWR\_UP и выключив PRIM\_RX**

**regval |= (1<<PWR\_UP);**

**regval &= ~(1<<PRIM\_RX);**

**NRF24\_WriteReg(CONFIG,regval);**

**DelayMicro(150); //Задержка минимум 130 мкс**

**//Отправим данные в воздух**

**NRF24\_Transmit(WR\_TX\_PLOAD, pBuf, TX\_PLOAD\_WIDTH);**

**CE\_SET;**

**DelayMicro(15); //minimum 10us high pulse (Page 21)**

**CE\_RESET;**

**return 0;**

}

In the file **max7219.c** we add the functions of clearing the left and right parts of the indicator separately (left and right 4 digits) after the clear indicator **Clear\_7219**

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

**void ClearL\_7219 (void)**

**{**

**uint8\_t i;**

**for(i=5;i<=8;i++)**

**{**

**Send\_7219(i,0xF);//символ пустоты**

**}**

**}**

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

**void ClearR\_7219 (void)**

**{**

**uint8\_t i;**

**for(i=1;i<=4;i++)**

**{**

**Send\_7219(i,0xF);//символ пустоты**

**}**

**}**

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

Let's create prototypes in the header file for these functions.

Let's return to the file  **NRF24.c** and fix the receive function as well. There will be practically only a display of information on the indicator

void NRF24L01\_Receive(void)

{

**if(rx\_flag==1)**

**{**

**ClearL\_7219();**

**NumberL\_7219(\*(int16\_t\*)RX\_BUF);**

**rx\_flag = 0;**

**}**

}

It remains to show the received packets on the indicator. We will do it in the timer handler, in the same place we will give commands to send response packets.

First, we'll create global timer counter variables and the counter of the sent packets, and also connect the buffer so that we do not waste unnecessary memory

volatile uint8\_t rx\_flag = 0, tx\_flag = 0;

**uint32\_t TIM1\_Count=0;**

**uint16\_t cnt1=0;**

**extern uint8\_t buf1[20];**

Now add the code to the timer handler

void TIM1\_Callback(void)

{

**if(TIM1\_Count%2000==0)**

**{**

**cnt1++;**

**memcpy(buf1,(uint8\_t\*)&cnt1,2);**

**NRF24L01\_Send(buf1);**

**ClearR\_7219();**

**Number\_7219(\*(uint16\_t\*)buf1);**

**if(cnt1>=9999) cnt1=0;**

**HAL\_Delay(1);**

**}**

**TIM1\_Count++;**

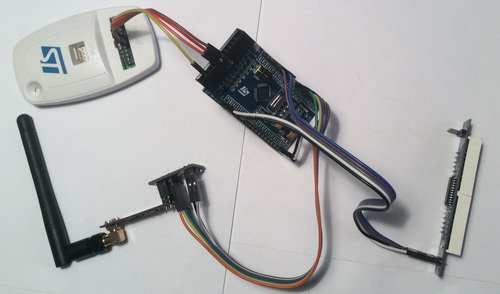
**if(TIM1\_Count>3000000) TIM1\_Count=0;**

}

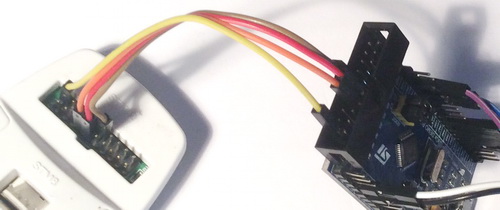
Once every 2 seconds we will send a packet with the counter value to the address of the second device and display this value on the right side of the indicator. Then we go into the receiver mode and wait for the packet from the second device and display its contents on the left side of the indicator.

We will collect the code, we will sew the controller. Only until we are unlikely to see any results, because the second device still does not know how to respond to packets. To accept it is likely to be able to, because the code remained from the previous session, and can not send yet. But for the present there is no reception, because we did not connect the second device. Turn off the ST-Link of the first device from the computer and take care of the second device. As a second device, we will have a card that we used in **lesson 115** for the second transmitter.

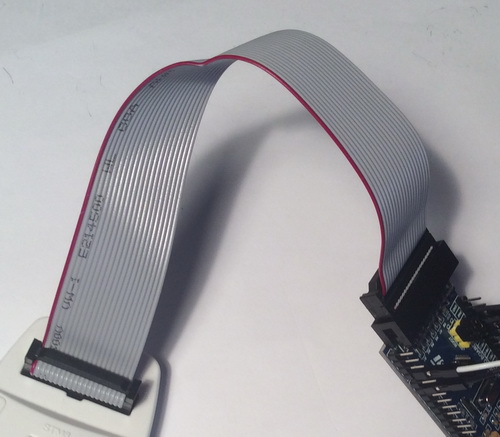
How to connect it cheap ST-Link, we already know. Therefore, today we connect to it the proprietary ST-Link (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2018/04/stm117img07.jpeg)

First connect the four wires. In this case, the power supply to the ST-Link is necessary, but the power wire is already playing a different role. The board is not powered by the programmer, but, on the contrary, ST-Link monitors that the board is powered. Consider the connection larger



Also we can connect ST-Link to such a card with the cable that comes with the programmer



In this case, we can use not only the SWD interface, but also JTAG for programming and debugging.

Connect the power to the board and connect the ST-Link to the PC.

The project we will make from the project of the first device and we will name it already **NRF24\_RXTX\_01** .

Run the project in Cube MX and, without touching it, open it in Keil. We will make traditional settings (auto-cutting and 1 level of optimization), add files  **NRF24.c** and **max7219.c** and try to assemble the project.

If everything is assembled without errors, then we will continue.

With this project it will not be so simple. We need not just change the address, but also teach the device to respond.

In the **main ()** function of the **main.c** file, we remove the start of the timer. We will launch it only when we accept the package

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

Let's **move** to file  **NRF24.c** and change places there for the addresses for receiving and transmitting

uint8\_t TX\_ADDRESS0[TX\_ADR\_WIDTH] = {**0xb5**,0xb5,0xa1};

uint8\_t TX\_ADDRESS1[TX\_ADR\_WIDTH] = {**0xb7**,0xb5,0xa1};

Connect the timer handle

extern SPI\_HandleTypeDef hspi1;

**extern TIM\_HandleTypeDef htim2;**

Correct condition body in function of **NRF24L01\_Receive**

if(rx\_flag==1)

{

**cnt1=\*(int16\_t\*)RX\_BUF;**

**ClearL\_7219();**

**NumberL\_7219(cnt1);**

**rx\_flag = 0;**

**TIM1\_Count=0;**

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

}

Here we start our timer, before doing this, resetting all the counters.

Now we also make a change to the event processing function of the timer

void TIM1\_Callback(void)

{

**if(TIM1\_Count>=1000)**

**{**

**cnt1 = 9999 - cnt1;**

**memcpy(buf1,(uint8\_t\*)&cnt1,2);**

**NRF24L01\_Send(buf1);**

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

**ClearR\_7219();**

**Number\_7219(\*(uint16\_t\*)buf1);**

**TIM1\_Count = 0;**

**}**

**TIM1\_Count++;**

}

Generally, a second after the timer starts, we will send the response packet to the first device. It will have 9999 minus the accepted value. Also in this function, we stop the timer until the next data packet arrives.

We will collect the package and sew the controller. We get this result

http://narodstream.ru/wp-content/uploads/2018/04/stm117img11.jpeg

Left module 2, and right - the first. Everything works perfectly. The first module transmits the data once a second, and the second one takes them perfectly and in return sends their data, which the first one successfully takes.

I tried to transmit quickly, as well as respond. I sent packets every 20 milliseconds and answered them in 1 millisecond. Everything works fine.

Thus, in this lesson, we learned how to quickly switch devices from reception to transmission, and then back, thereby teaching our devices to communicate among themselves by sending packets.

Thank you all for your attention!