**STM Lesson 115. NRF24L01. Several receivers. Part 1**

Posted on [April 10, 2018](http://narodstream.ru/stm-urok-115-nrf24l01-neskolko-priemnikov-chast-1/)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 by STM32](http://narodstream.ru/rub_stm32/)- [No Comments ↓](http://narodstream.ru/stm-urok-115-nrf24l01-neskolko-priemnikov-chast-1/#respond)

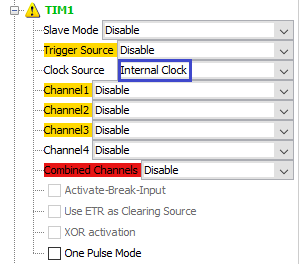
[Programming for beginnersDownload the free book and become a real programmer. Get it now:To learn moresheremetev.info](https://an.yandex.ru/count/JNuRuytykl450Dy1CUI-ari00000ECgs7402I09Wl0Xe172WmvR10O01vE3CgmU80VY6tP0fa06YakID99W1Zhs-tYEW0VwIp8qag06M_w_U8xW1XgMPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y083e0BUkAe2kG8MMP_nASeTMF02uS6V-mhu0eA0W820i1A00__Fl_qBY0ElXgkW2fW3YSK4g0CIi0C4k0J_0UW4gmRu19_m0OW5d_01a0NSc0AW1Ob8g0MAFR05YZsu1Qz7m0NmcGV81R3g0T05lXlW1Lhm1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XH99q1AyL8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2A-SbG602W712W0000000F0_s0e2u0g0YNhu2i3y5OWB1geB47T83JNuKG00wmIFArjr1G302u2Z1SWBWDIJ0TaB-54adG4hnKZe2v_m0V0B1eWCjfNUlW7e30AO3PMCQF8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_ueu6MaE3AuaW0?stat-id=3&test-tag=80814134026241&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY1IjoiODA4MTQxMDQ2NzQzMDQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Looking for Python courses?**Python programming** courses from scratch. Theory and practice. Start learning!To learn moreshultais.education](https://an.yandex.ru/count/NdCJsJtdRtm50Dq1CUM-ari00000ECgs7402I09Wl0Xe173soUR12e01ZREU68W1iVwJcZAG0RBUk-ujc06-flosAw01zjkuxYse0Poc_BOhk07wyTk46S010jW1YCk-4E01tll01-W1fW7u0OIlthu1Y084e0AKz8qJkG8MMP_nASeTMF02_vpMsWVu0eA0W820i1A00yxjkC88Y0FPtFFd0PW3Zj89i0C4k0J_0UW4cGFu1Fde0eW5-UW2a0MzfGIW1QHig0NYMx05ubku1TcB0S05uBCEo0NWr0FG1QKfu0K-y0K1c0QWuexd0g06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00071n0000gGVuKV0oA2l5IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0YA-iO9W0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mYg2n2vUwq9-54003kgc2jRTGK0m0k0emN82u3Kam7P2_XHy38eAyL8w0lvw0Bm2mQ838Ilthu1w0m2c0tylMRo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Efhysi0u1eGy00000003mFwWF-AEHj_Raok98?stat-id=4&test-tag=80814134026241&format-type=24&banner-test-tags=eyI2MTU1NjMwNDg5IjoiODA4MTQxMDQ2NzQzMDQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

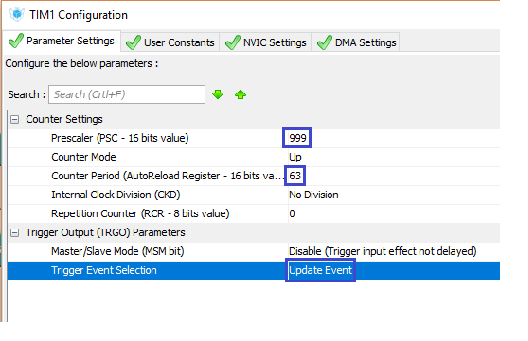
We continue to work with the wireless transceiver  **NRF24L01** . And today the board, which we acted as a receiver, will act as a transmitter. And we will solve the problem again. To transfer packets from the module connected to this card, on the contrary, to several receivers. We will have three receivers and our task will be to transfer the packet to any of these three receivers at any time from the transmitter. This task turned out to be doable, to dig deeper, of course it was necessary, but the goal was achieved, so I immediately decided to share with you my work.

The project for our transmitter on the **Nucleo F401RE board** was created from the receiver's project of the [**lesson 113**](http://narodstream.ru/stm-urok-113-nrf24l01-neskolko-peredatchikov-chast-1/) **NRF24\_RX\_LCD** and was named, respectively, **NRF24\_TX\_LCD** .

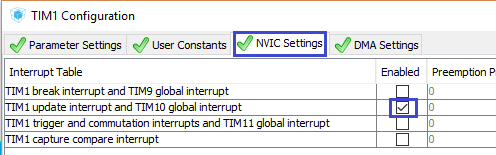
Let's run our project in Cube MX and turn on some timer, I think it will come in handy today.



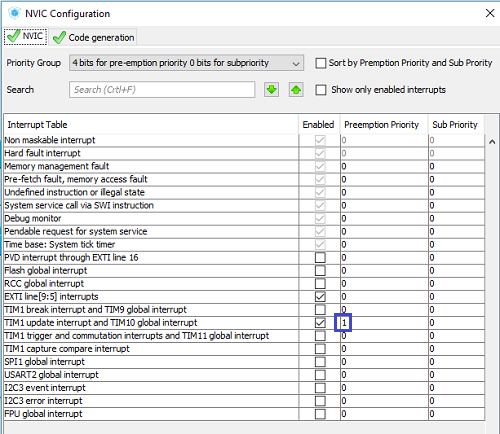
Let's go into the **configuration** and configure there the period of our timer for about 1 millisecond



Also set the timer interrupt



Now go into **NVIC** settings and reduce the priority of interrupts from the timer. The highest priority for us should be for interrupts from the module foot



Save the settings, generate the project for Keil, run it, configure the programmer for autorestart, and enable the optimization level in **1** . Also we will connect **NRF24.c** and **lcd.c** files to the project tree  , and then we will try to collect our project.

If everything is normal, then we move on.

We remove in the **main ()** function from an infinite loop the call to the receive function of the package

/\* USER CODE BEGIN 3 \*/

~~NRF24L01\_Receive();~~

Do not forget to run our timer

NRF24\_ini();

**HAL\_TIM\_Base\_Start\_IT(&htim1);**

We pass to file **NRF24.c** and at the very bottom we shall add function- **output agent** for events of the timer

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

**void TIM1\_Callback(void)**

**{**

**}**

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

Let's create a prototype for this function, go back to the **main.c** file and create a global variable to count the ticks of our timer

uint8\_t buf1[20]={0};

**uint32\_t TIM1\_Count=0;**

Let's create also in this file also an interrupt handler from the timer after the function of the external interrupt handler. In this function, we will count the ticks of our timer and call the handler from the library

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

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

**{**

**if(htim==&htim1)**

**{**

**TIM1\_Callback();**

**TIM1\_Count++;**

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

**}**

**}**

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

The restriction to 3,000,000 was due to the fact that we will track different time periods, the number has been chosen such that for our periods it is shared without a trace.

We return now to our library file **NRF24.c** . The rest of the code will now only be there.

Let's add a flag for the transfer

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

Reduce the size of the buffer, because we will not be able to transfer the number of non-transmitted packets. We played with it, made sure that everything is normally transmitted by us

#define TX\_PLOAD\_WIDTH **5**

Since we will send packets to different receivers, the transmission address will change in a certain place, and we will have to change the addresses in the receiving channels, since the **pipe** address must always contain the transmission address. So let's add some more address macros

#define TX\_PLOAD\_WIDTH 5

**#define TX\_ADDRESS0\_MSB 0xb5**

**#define TX\_ADDRESS1\_MSB 0xb7**

#define TX\_ADDRESS2 0xb6

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

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

**uint8\_t TX\_ADDRESS2\_FULL[TX\_ADR\_WIDTH] = {0xb6,0xb5,0xa1};**

We also changed the address of the zero transmitter, since now it will not always be zero, and now it will have to play under a common tune, that is, it also has all the bytes except the youngest to be the same as the other addresses.

And we will change the addresses in a function that includes the transmitter mode. I think this is the right place for this. Therefore, we add one more argument to this function - the number of the receiver we are going to transfer packets to. After that, the function will get the following form

void NRF24L01\_TX\_Mode(**uint8\_t TX\_num,** uint8\_t \*pBuf)

{

**switch(TX\_num)**

**{**

**case 1:**

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

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

**NRF24\_Write\_Buf(RX\_ADDR\_P1, TX\_ADDRESS1, TX\_ADR\_WIDTH);**

**NRF24\_WriteReg(RX\_ADDR\_P2, TX\_ADDRESS2);**

**break;**

**case 2:**

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

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

**NRF24\_Write\_Buf(RX\_ADDR\_P1, TX\_ADDRESS0, TX\_ADR\_WIDTH);**

**NRF24\_WriteReg(RX\_ADDR\_P2, TX\_ADDRESS2);**

**break;**

**case 3:**

**NRF24\_Write\_Buf(TX\_ADDR, TX\_ADDRESS2\_FULL, TX\_ADR\_WIDTH);**

**NRF24\_Write\_Buf(RX\_ADDR\_P0, TX\_ADDRESS2\_FULL, TX\_ADR\_WIDTH);**

**NRF24\_Write\_Buf(RX\_ADDR\_P1, TX\_ADDRESS0, TX\_ADR\_WIDTH);**

**NRF24\_WriteReg(RX\_ADDR\_P2, TX\_ADDRESS1\_MSB);**

**break;**

**default:**

**break;**

**}**

  CE\_RESET;

  // Flush buffers

  NRF24\_FlushRX();

  NRF24\_FlushTX();

}

I think there is nothing complicated here. We just swap addresses depending on which address we are going to send packets to.

As we remember, we removed all code from the send function of the package. Now we will add it anew, only taking into account the fact that on the foot of interrupts level changes we will already track hardware. Also in our function one more input argument will be added - the number of the receiver to which the packet will be sent

uint8\_t NRF24L01\_Send(**uint8\_t TX\_num,** uint8\_t \*pBuf)

{

**uint8\_t regval=0x00;**

**NRF24L01\_TX\_Mode(TX\_num, 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 prototype, we'll also add a new argument.

We practically have all the code left, only as long as we waited for failures in the condition of zeroing the level on the interrupt leg, now this will be processed in a special handler function.

Also in some places, where we addressed to addresses more than the 5th, we will correct the code, so that there are not even warnings when assembling.

First, we will correct this in the function **NRF24L01\_Receive** , and also remove the extra argument in the **sprintf** function call

LCD\_SetPos(0, \*(RX\_BUF+**5**));

**sprintf(str1,"%5u %5u ", \*(int16\_t\*)RX\_BUF, \*(int16\_t\*)(RX\_BUF+2));**

Now in function **IRQ\_Callback**

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

After the function NRF24L01\_Receive we add the function of outputting the number of the packet to be sent to the corresponding line. Moreover, we will send data to the display only when the flag is on

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

**void NRF24L01\_LCD(uint16\_t num, uint16\_t pos)**

**{**

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

**{**

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

**sprintf(str1,"%5u ", num);**

**LCD\_String(str1);**

**tx\_flag = 0;**

**}**

**}**

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

Now we go to the **IRQ\_Callback** interrupt **routine** . First of all, remove the unclaimed variable from there

~~uint16\_t dt=0;~~

We will remove the flashing LED, we will blink it now only when certain interrupts occur

~~LED\_TGL;~~

Flashing the LED in the event of an interrupt in the event of receiving a packet

if(status & 0x40)

{

**LED\_TGL;**

Add to the conditions of transmission interrupt and packet errors

    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();**

**}**

}

We connect the global timer counting timer ticker, also the buffer to prepare the sent packet, and also add three separate counters for our three receivers in the form of global variables

uint8\_t ErrCnt\_Fl = 0 ;

**extern uint32\_t TIM1\_Count;**

**extern uint8\_t buf1[20];**

**uint16\_t cnt1=0, cnt2=0, cnt3=0;**

Well and it remains to us only to send packets on our receivers in function-output agent from the timer

void TIM1\_Callback(void)

{

**if(TIM1\_Count%1000==0)**

**{**

**cnt1++;**

**NRF24L01\_LCD(cnt1, 0);**

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

**NRF24L01\_Send(1, buf1);**

**HAL\_Delay(1);**

**}**

**if(TIM1\_Count%600==0)**

**{**

**cnt2++;**

**NRF24L01\_LCD(cnt2, 1);**

**memcpy(buf1,(uint8\_t\*)&cnt2,2);**

**NRF24L01\_Send(2, buf1);**

**HAL\_Delay(1);**

**}**

**if(TIM1\_Count%1500==0)**

**{**

**cnt3++;**

**NRF24L01\_LCD(cnt3, 2);**

**memcpy(buf1,(uint8\_t\*)&cnt3,2);**

**NRF24L01\_Send(3, buf1);**

**HAL\_Delay(1);**

**}**

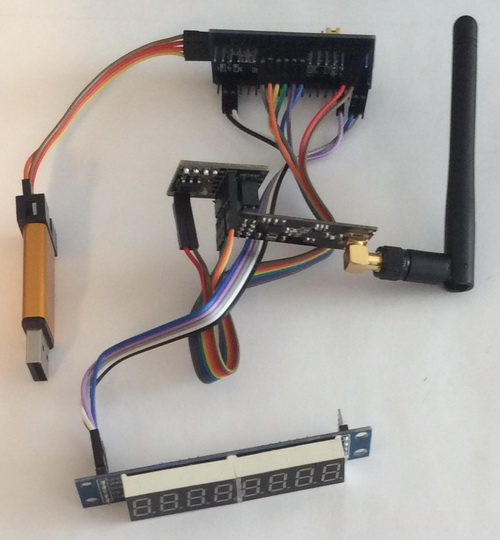
}

Here we determine which receiver is the right time for us to send the packet. We determine this by achieving a certain number of timer ticks. On the first receiver we send the packet approximately once in one second, on the second one - every 600 milliseconds, on the third one - once a half seconds. The delay in conditions of 1 millisecond is necessary for the case of coincidence of conditions, since without delay we will not transmit packets to several transmitters at the same time. I tried. One millisecond was chosen approximately, perhaps, there will be less delay, so you can play with delays in microseconds, creating a prototype for the delay function in the lcd.h file.

We will collect the project, we will impose the controller. The result is, most likely, no one will see, since we do not yet have a single receiver. Therefore, we will disconnect our transmitter from the PC and connect it to an independent power source.

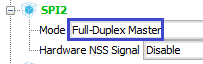
As the first receiver we will use the scheme of the first transmitter on the controller **F103** , located on an inexpensive board. Only to everything else we connect to it for clarity 8-bit indicator on the chip MAX7219.

Look our first receiver will be as follows

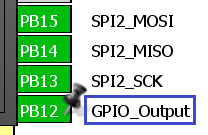


The project for it we will do from the project of the [**lesson**](http://narodstream.ru/stm-urok-113-nrf24l01-neskolko-peredatchikov-chast-1/) transmitter [**113**](http://narodstream.ru/stm-urok-113-nrf24l01-neskolko-peredatchikov-chast-1/)**NRF24\_TX\_PIPE0** and call it  **NRF24\_RX\_00** .

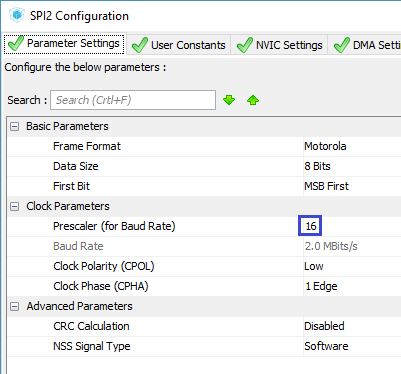
Run our project in the projector Cube MX and **turn** on **SPI2** for the 8-bit indicator



We also include the foot **PB12** for the foot **CS**



Let's go into **Configuration** and configure our **SPI2**



Save the settings, we will co-create the project for Keil, open it in it, configure the programmer for auto-cutting, set the optimization level to **1** .

The files **max7219.c** and **max7219.h can be** copied to our project from the [**lesson 107**](http://narodstream.ru/stm-urok-107-datchik-vlazhnosti-i-temperatury-dht22-chast-1/) project by **DHT22**temperature **sensor** with the same name.

Let's return to the project and connect the files  **NRF24.c** and  **max7219.c**  to the project tree.

Let's try to collect our project.

If everything is fine, we will connect the library for the indicator in the **main.c** file

#include "NRF24.h"

**#include "max7219.h"**

From the **main ()** function, delete all local variables

~~//uint8\_t dt\_reg=0;~~

~~uint8\_t dt;~~

~~uint16\_t i=1,retr\_cnt\_full=0, cnt\_lost=0;~~

In the same function, we initialize our indicator

/\* USER CODE BEGIN 2 \*/

**HAL\_Delay(100);**

**Init\_7219();**

NRF24\_ini();

From an infinite loop, we completely remove all the code.

We go to file  **NRF24.h** and connect also the indicator library

#include "stm32f1xx\_hal.h"

**#include "max7219.h"**

Now go to the file  **NRF24.c** .

We will change there the number of bytes in the packet, as well as the address, since we changed it in the transmitter

#define TX\_PLOAD\_WIDTH **5**

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

We add after the function of transferring the packet  **NRF24L01\_Send the** function of receiving the packet, for us it is already standard

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

**void NRF24L01\_Receive(void)**

**{**

**uint8\_t status=0x01;**

**while((GPIO\_PinState)IRQ == *GPIO\_PIN\_SET*) {}**

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

**DelayMicro(10);**

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

**if(status & 0x40)**

**{**

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

**LED\_TGL;**

**Clear\_7219();**

**Number\_7219(\*(uint16\_t\*)RX\_BUF);**

**NRF24\_WriteReg(STATUS, 0x40);**

**}**

**}**

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

Add a prototype for this function in the header file.

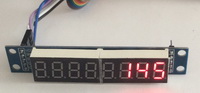
Back in the **main.c** file and in the **main ()** function in an infinite loop, we call the receive function of the package

/\* USER CODE BEGIN 3 \*/

**NRF24L01\_Receive();**

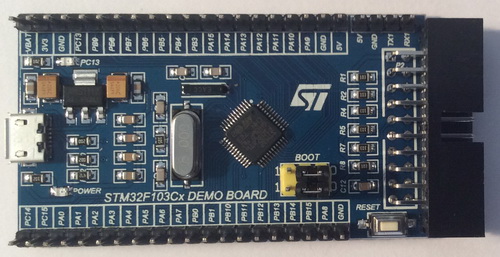
}

We will collect the code, we will tell the controller and see that the packets from the transmitter come to us normally

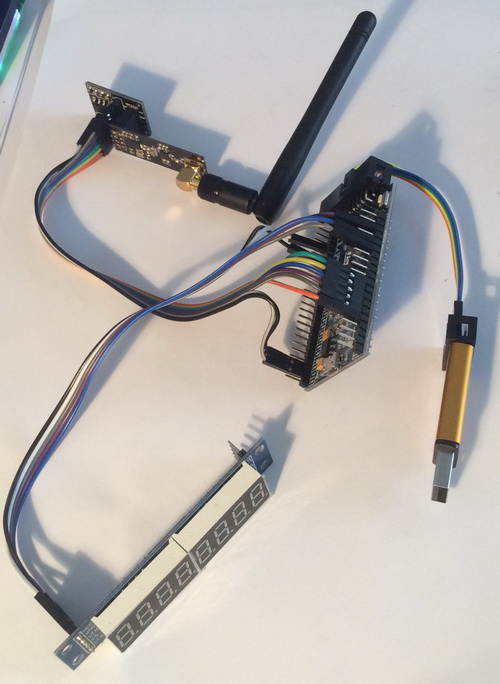


On the transmitter display, while the packets are not on that line, but I think when we connect all the receivers, everything will be fine.

Now connect our first receiver from an independent power source, and via the USB OTG connector, since we need ST-LINK for the second transmitter. I have two more F103 boards, but there are no ST-Link-cheap, through which you can power the boards. Here's one of these boards we'll take as the second receiver. It looks a little different than our usual, but the meaning is the same, and this is even more convenient



We connect the NRF module and the indicator to the same legs as the first transmitter, and also connect the ST-Link



Connect ST-Link to the PC.

And we will create the project in the [**next part of**](http://narodstream.ru/stm-urok-115-nrf24l01-neskolko-priemnikov-chast-2/) our lesson, in which we will also create and write a project for the third receiver, and also check how the address transmission works for several receivers.

**STM Lesson 115. NRF24L01. Several receivers. Part 2**

Posted on [April 12, 2018](http://narodstream.ru/stm-urok-115-nrf24l01-neskolko-priemnikov-chast-2/)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/)- [1 comment ↓](http://narodstream.ru/stm-urok-115-nrf24l01-neskolko-priemnikov-chast-2/#comments)

[Programming for beginnersDownload the free book and become a real programmer. Get it now:To learn moresheremetev.infoYandex.Direct](https://an.yandex.ru/count/DMTHRnOJW7m50Dm1CGU_ari00000ECgs7402I09Wl0Xe172WmvR10O01vE3CgmU80VY6tP0fa06YakID99W1Zhs-tYEW0VwIp8qag06M_w_U8xW1XgMPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y083e0BUkAe2kG9cg2X6BsckdV02kAAIx0pu0eA0W820y1A00xZiZ_KCY0F4dz3T0vW3YSK4g0CIi0C4k0J_0UW4w0Ru1BHbY0MqPP05Xve2e0Ms7gW5v1cm1UGPk0NQHy05y9a7o0MmwW7G1SqBu0LQy0K1c0Q0qApp3g06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00071n0000gGVuKKpnEZF5IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0Yld9K1W0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mQg2n2wfMcP-54009zPdIrRTGK0m0k0emN82u3Kam7P2_XHJF4wCyL8w0kqPV0B1eWCjfNUlW7e30AO3VYSPV8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_uew7_d-VCuaW0?stat-id=3&test-tag=83013157281793&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY1IjoiODMwMTMxMjc5Mjk4NTYifQ%3D%3D&)

[PHP courses from scratchWeb server and data server, PHP and SQL, typical backend tasks. Personal ProjectTo learn morehtmlacademy.ruYandex.Direct18+](https://an.yandex.ru/count/NmisBXjxA_i50Dm1CGU_ari00000ECgs7402I09Wl0Xe173MZO-53e01oexaomU80V6TcjGqa06GnkkWBvW1vhlk-IsW0Pp6ww0lg06Ek-xvBRW1w9tytXd00GBO0OgrwnJW0TIzXWVe0QO2-06uhzw-0OW21A02hCMi3xa2PgWeHYzfhftm0jUgnfK6-0A2W820WF0IW0FdwUV-2eW3qCQsq0Qm0mIu1Fy1w0Ij0VW4fUS4Y0MbvmIG1RJy2A05oxW1g0Mtd06m1RUS0RW5_AC2m0MRj1h81R3R1j05iaRW1Jxm1G6O1lArsFuDe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XHHErPCyL8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2DQYrWU02W712W0000000F0_s0e2u0g0YNhu2i3y5OWB2AeB4AHun0xuKG00lHYbBLjr1G302u2Z1SWBWDIJ0TaB-554xLapnKZe2wNd1F0B1eWCkA_UlW7e30AO3VozPl8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0xCg0om3W6X3m0000000F0_g0_ueu6oa-lCuaW0?stat-id=4&test-tag=83013157281793&format-type=24&banner-test-tags=eyI2MzQyNjY4NjgwIjoiODMwMTMxMjc5Mjk4NTYifQ%3D%3D&)

In the [**previous part of**](http://narodstream.ru/stm-urok-115-nrf24l01-neskolko-priemnikov-chast-1/) our lesson we created and wrote a project for the receiver, as well as a project for the first transmitter.

We will do the project completely from the project of the first transmitter  **NRF24\_RX\_00** and call it  **NRF24\_RX\_01** with the purpose of least time to write the source code.

Let's open the project in Cube MX, do not touch anything there, generate the project for Keil, open it, configure the programmer for auto-cutting, set the optimization level to **1** , and also connect the files **NRF24.c** and  **max7219.c** to the project tree .

Let's try to assemble the project.

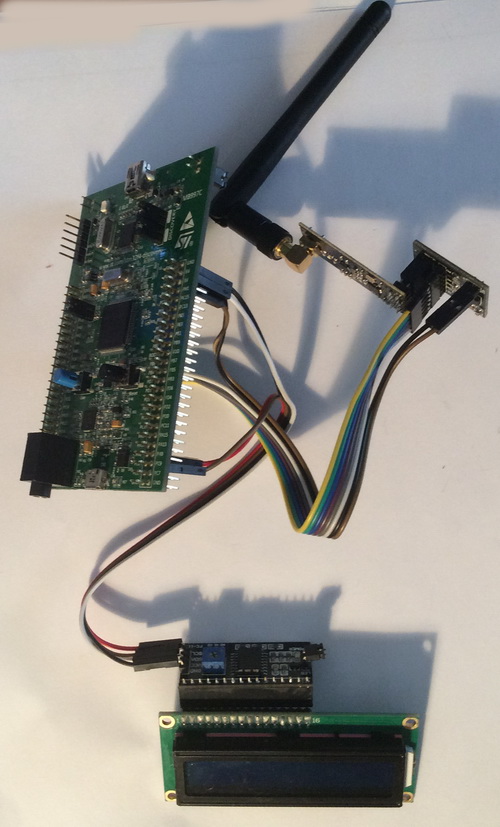
If everything is fine, then go to the file  **NRF24.c** and change there only the address, or rather, only its lower byte

uint8\_t TX\_ADDRESS[TX\_ADR\_WIDTH] = {0x**b7**,0xb5,0xa1};

We will collect the code, we will sew the controller. Now both of our receivers perfectly accept all packets from the transmitter



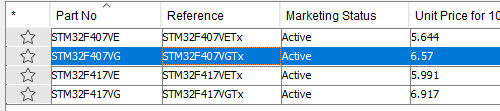
As a third transmitter, I wanted to take a similar charge, but for some reason I had a third 8-bit indicator refused to work. So I decided to use the display on the **HD44780** controller . The display is small in size **16 × 2** . We have already worked with this. And we will connect it as well as the display on the transmitter, via the adapter via the **I2C** bus . On this bus it with the F103 controller also refused to work together with the NRF module. It works normally without it. Therefore, as a control board, I decided to take the good old **STM32F4-DISCOVERY**, with whom we have gone through many lessons and know about it deeply not by hearsay. We connect the NRF module and the display (which legs and what to connect, we'll see later when we configure the project in the Cube MX). We will get here is such a scheme



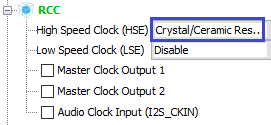
We connect our third transmitter to the PC, and the second one is fed from an independent source.

We will create a new project for our third transmitter, since we have not worked with this board for a long time.

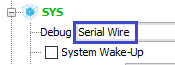
Let's choose the controller



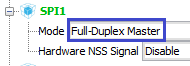
Turn on the quartz resonator



Turn on the debugger



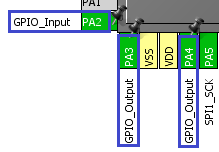
Enable **SPI1** for the NRF module



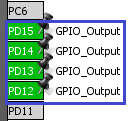
For the display, **turn on I2C3**

http://narodstream.ru/wp-content/uploads/2018/03/stm115img18.png

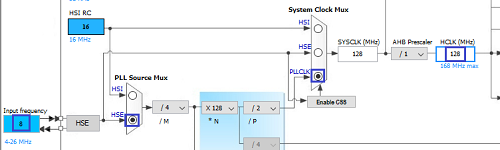
The foot **PA2 is** switched on to the input - this will be the **IRQ** foot for the NRF module. And legs **PA3**and **PA4** - on the output. This will accordingly be **CE** and **CS**



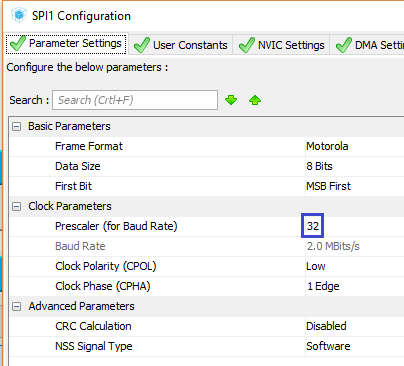
Also we will include the legs connected to the LEDs, also to the output



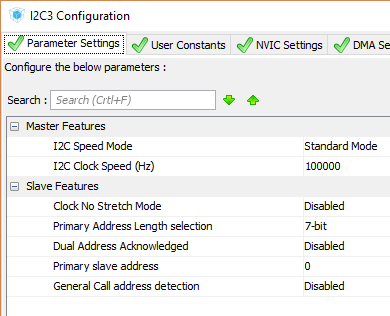
Go to **Clock Configuration** and adjust our frequencies (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2018/03/stm115img21.png)

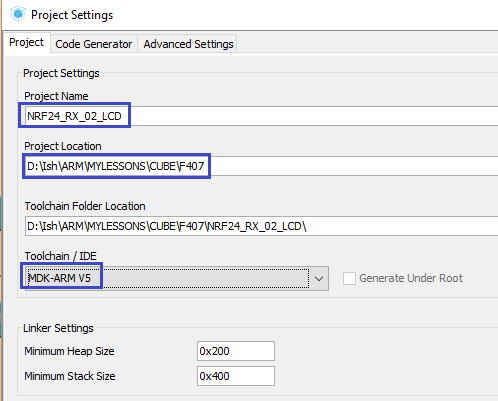
Go to **Configuration** , configure the **SPI** bus



Check the **I2C** settings



Let's go into the project settings, set the project directory, name and select the programming environment



Save the settings, generate the project in Keil, open it, set up the programmer for autorestart and enable the optimization level in **1** .

Let's try to assemble the project.

If everything is ok going, some of the project 1 receiver copy files **NRF24.c** and **NRF24.h** , a transmitter of the project - files **lcd.c** and **lcd.h** . These files are quite suitable, there will be almost no corrections for the display of this dimension, except for the positioning function.

Connect the **NRF24.c** and  **lcd.c** files to the project .

In the **main.c** file we connect our libraries

/\* USER CODE BEGIN Includes \*/

**#include "NRF24.h"**

**#include "lcd.h"**

**#include <string.h>**

/\* USER CODE END Includes \*/

Initialize the display and the transceiver in **main ()**

/\* USER CODE BEGIN 2 \*/

**LCD\_ini();**

**NRF24\_ini();**

/\* USER CODE END 2 \*/

Call the function of receiving a packet in an infinite loop

/\* USER CODE BEGIN 3 \*/

**NRF24L01\_Receive();**

}

In the **lcd.c** file **,** remove the extra lines from the positioning function, since there are only 2 lines in our display. The function now takes on the following form

**void LCD\_SetPos(uint8\_t x, uint8\_t y)**

**{**

**switch(y)**

**{**

**case 0:**

**sendbyte(x|0x80,0);**

**break;**

**case 1:**

**sendbyte((0x40+x)|0x80,0);**

**break;**

**}**

**}**

Go to file **NRF24.h** , fix the library, also remove the connection of the indicator library, and instead we connect the library for the display

#include "stm32f**4**xx\_hal.h"

~~#include "max7219.h"~~

**#include "lcd.h"**

Also, we fix the foot for the LED and the polarity of its on and off

#define LED\_GPIO\_PORT GPIO**D**

#define LED\_PIN GPIO\_PIN\_**12**

#define LED\_ON HAL\_GPIO\_WritePin(LED\_GPIO\_PORT, LED\_PIN, GPIO\_PIN\_**SET**)

#define LED\_OFF HAL\_GPIO\_WritePin(LED\_GPIO\_PORT, LED\_PIN, GPIO\_PIN\_**RESET**)

Let's **go** to file  **NRF24.c** and replace the lower byte of the address

uint8\_t TX\_ADDRESS[TX\_ADR\_WIDTH] = {**0xb6**,0xb5,0xa1};

Let's connect a string array

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

**extern char str1[100];**

In the function of receiving the **NRF24L01\_Receive** package,  **we** replace the lines of code responsible for displaying information on the indicator on lines that will display information on the display

~~Clear\_7219();~~

~~Number\_7219(\*(uint16\_t\*)RX\_BUF);~~

**LCD\_SetPos(0, 0);**

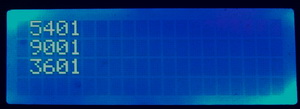
**sprintf(str1,"%5u ", \*(int16\_t\*)RX\_BUF);**

**LCD\_String(str1);**

We will collect the code, we will sew the controller. After that we see that all the packages are fine and on time delivered and information is displayed on the display and the indicators of all three receivers (click on the picture to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2018/03/stm115img25.jpeg)

Also, at the transmitter, the display is all fine and on time displays



Here and so it looks like our entire overall scheme in practice (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2018/03/stm115img00.jpeg)

Thus, this lesson showed us that it is quite realistic to work immediately with three receivers assembled on NRF modules. We also learned how to change the transmission address of the transmitter and send packets to any of the three receivers, sometimes almost simultaneously.

Even today, we once again became convinced of the fact that the use of the HAL library makes it easier to write the code. We, almost without straining, transfer the code from one controller to another.

Thank you all for attention!