**STM Lesson 113. NRF24L01. Several transmitters. Part 1**

Posted on [March 27, 2018](http://narodstream.ru/stm-urok-113-nrf24l01-neskolko-peredatchikov-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-113-nrf24l01-neskolko-peredatchikov-chast-1/#respond)

[Fundamentals of Programming LanguagesDownload the free book and become a real programmer. Get it now:To learn moresheremetev.info](https://an.yandex.ru/count/Q5diOHn9Qfq50Du1COA-ari00000ECgs7402I09Wl0Xe172WmvR10O01ZSt-9uW1jeRTa2cG0VYKhOyac07okhxU8w01lfFCZIIe0PB-hzuZk062f9dq5S010jW1bh2j4E01vfJJ3kW15lW1fkdUlW680WEW0jwugWAv0ZKEeSckSQ_iy0ACpFxi3VW2We20W82W5803YVo9_Gs80uR2xf84c0FoqmMe0nAm0mIu1Fy1w0IR1FW4jPW1Y0Mrc06G1PkM0g05tIse1U0ci0NW9hW5tKR01V2P1yW5iEe1q0Mt4U05Fl050PW6WD2iymwW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000Aa7-54yzwCJnKYm1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08hvoL0O0A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i6gWiGwXNTM_XH003rchSbMtK50C0BWAC5o0k0r9C1sGluKJptenF5IEWBjPW1y0i6Y0ocwTw-0UWC0fWDbOneyWq0-Wq0WWu0YGu00000003mFv0Em8Gzc0x6rypKbOdWkXEW3i24FR0E0Q4F00000000y3-e3_YZaP2nzCJYI000?stat-id=3&test-tag=89060471234561&format-type=24&banner-test-tags=eyI0ODQ3OTM2ODI4IjoiODkwNjA0NDE4ODI2MjQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Buy robot constructorTeach your child to program. 6900 for a set of shares. OrderTo learn moredorado-uno.ru](https://an.yandex.ru/count/EwrwcfOS6sm50D41COE-ari00000ECgs7402I09Wl0Xe172AffoG1e01xCtW6OW1rBMagJMG0VZAnCelc07gZ-3LBg01zih4oY-e0UYEuDKkk07st_Rr6S010jW1ieMB5U01qBto4UW1OFW1elhUlW680WIW0g2-sHQv0ZKEeSckSQ_iy0BKg-3n2FW2We20W82W5803cuYDb0-80_p-_FuFi0C4w0It0lW4b8q2Y0MKZGAG1SFC0w05u52e1RT4i0MtHBW5YdN01R3x3CW5yAK3q0Nb7k05Ml050PW6tgcE6Q06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00071n0000gGVuKQ3NoXF5IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0YGtwmBW0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mYg2n2xDKOO-54007PJmILRTGK0m0k0emN82u3Kam7P2_XHeDVA4yL8w0kKZGBm2mQ83ABwthu1w0m2c0tylMRo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Em8Gzi0u1eGy00000003mFwWF-AFngwRvnE98?stat-id=4&test-tag=89060471234561&format-type=24&banner-test-tags=eyI2Mzg2Mzg5NjkyIjoiODkwNjA0NDE4ODI2MjQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

We continue to work with an interesting wireless transceiver **NRF24L01** . And today, continuing to develop our topic of programming data transmission over wireless channels, we will try to accept the packets simultaneously from three NRF modules on the board, acting as a receiver. I did it, and therefore I want to share this with you immediately.

First, in order to visually analyze the reception of packets on the receiving device, there are not enough 8 bits of our favorite indicator. I even bought two more, but I could not connect them in a cascading way, maybe the modules are clumsy, maybe even that, in general, the voltage at the input of each module was much higher than the output voltage. Well, okay, I've already figured out where to use all three indicators in the future.

Therefore, as a means of displaying information, I decided to use our favorite character display 20 × 4 with an I2C adapter, which had a lot of lessons, and besides, another such display came to me and I decided to use it. He is already blue. but not green and the characters it displays in white, there will be at least some variety. But that's not the point. The adapter was already soldered into this display. This display for some reason did not start in all my projects. I tried to run it using the Arduino library, and it started. I began to carefully study this library and found there a lot of interesting and undocumented data on the display. Therefore, I decided to share these corrections with you, too, since, judging by many comments, many also do not launch displays, maybe at the same time and my lesson will help.

This is the plan we have here.

Therefore, we will begin to implement it somehow.

However, here is another thing I came across. It is impossible, it turns out, to use simultaneously I2C and SPI of one number, that is, I2C1 and SPI1 or I2C3 and SPI3. I did not understand why. I warn you in advance. It is necessary to choose the data of the bus differently. That is, if we use SPI1, then we can use I2C2 or I2C3, but not I2C1.

Let's connect one of our transmitters, the one that we had, but we do not need to connect the temperature sensor, but we'll fill it with firmware from [**lesson 105**](http://narodstream.ru/stm-urok-105-nrf24l01-peredayom-dannye-chast-1/) . Then we connect the transmitter from an independent source, and connect a receiver to the PC, that is, a Nucleo F401 board.

While we will all look like this. On the display of the squares



To which feet to connect the display, we'll see when we set up the project in the Cube MX.

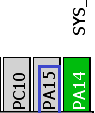
The project for the receiver will be created from the [lesson](http://narodstream.ru/stm-urok-109-nrf24l01-peredayom-vlazhnost-i-temperaturu-s-datchika-dht22/) project [109](http://narodstream.ru/stm-urok-109-nrf24l01-peredayom-vlazhnost-i-temperaturu-s-datchika-dht22/)**NRF24\_RX\_DHT22** and call it **NRF24\_RX\_LCD** .

The files **max7219.c** and **max7219.h** from the folders of the new project can be deleted.

Let's open our project at Cube MX, disable bus **SPI3**

http://narodstream.ru/wp-content/uploads/2018/03/stm113img01.png

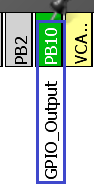
The PA15 leg, which we used to work as a CS leg for SPI3, can also be disabled



The port of the PA8 port, which we were responsible for contacting the NRF **CE** module , is also disabled, otherwise we will not turn on the I2C



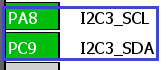
Instead of it we include a foot PB10, as it is not far (do not forget also to re-commute and postings)



Now we can turn on the **I2C3** bus

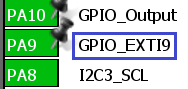
http://narodstream.ru/wp-content/uploads/2018/03/stm113img04.png

Let's see what we have included legs, to connect them to the adapter display

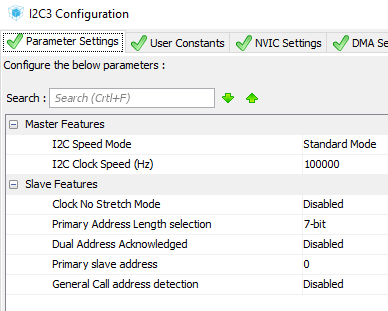


Power supply to the display will be supplied from the contact 5 volts, otherwise there will not be enough contrast.

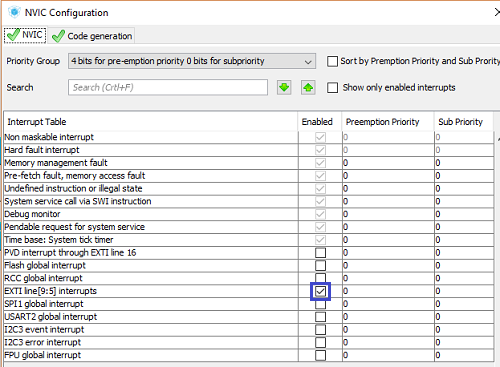
Also, signals on the interrupt leg from the NRF module will be captured in hardware by using external interrupts on this leg



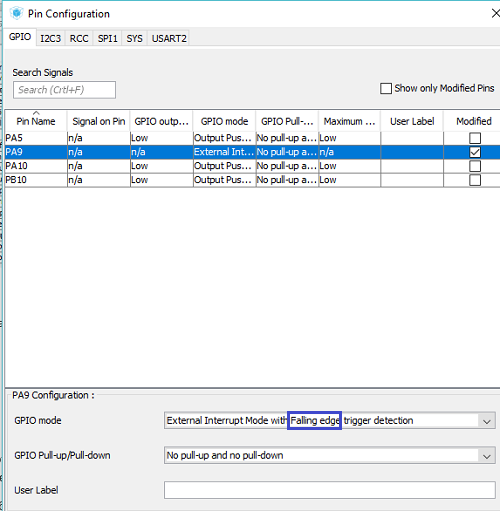
Go to **Configuration** , open the **I2C** settings to make sure that everything is fine there, and then everything happens



Also we'll go into **NVIC** and turn on external interrupts



Also the most important thing that many people can forget is to switch the front of interrupts. At us they are traced precisely on a negative front (on recession). Therefore, we'll go into the settings of the **GPIO** and for our feet we will configure such an interrupt mode



We will collect the project and open it in Keil, configure the programmer for auto- **reload, enable the**optimization level in **1** , connect the file **NRF24.c**  and try to assemble our project. He most likely will not meet, referring to the lack of a library for the indicator. And okay, we do not need it, we will work with the LCD display. Therefore, from the [**lesson 22**](http://narodstream.ru/stm-urok-22-hal-i2c-i2c-to-lcd2004/) project **I2CLCD80 we** copy the files **lcd.c** and **lcd.h** into our project .

We connect the file **lcd.c** to the project, as well as in the **main.c** file, remove the library connection for the indicator, and connect the file lcd.h

~~#includ~~

**#include "lcd.h"**

The same is **done** in the file **NRF24.h** .

Since we have come to this file, we will correct the foot for CE

#define CE\_GPIO\_PORT GPIO**B**

#define CE\_PIN GPIO\_PIN\_**10**

We will also remove in this file everything related to the IRQ foot, as the interrupts will now be processed by us hardware.

~~#define IRQ\_GPIO\_PORT GPIOA~~

~~#define IRQ\_PIN GPIO\_PIN\_9~~

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

Go to file  **NRF24.c** and remove all the code from the function **NRF24L01\_Receive** . There will be another code later.

Also, in the transfer function  **NRF24L01\_Send,**  we will also remove all the code until future times, otherwise we risk forgetting about interrupts. When we work with this card in the transmitter mode, then the code will be added.

Change the size of the global string array

extern char str1[**100**];

Create the function at the very bottom of the file to handle the interrupt

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

**void IRQ\_Callback(void)**

**{**

**}**

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

Let's create a prototype for this function in the header file and go to the **main.c** file , where we will also create an interrupt handling function, in which we trace the interrupt from our feet, calling in the body the conditions just added by us

/\* USER CODE BEGIN 4 \*/

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

**{**

**if(GPIO\_Pin== GPIO\_PIN\_9)**

**{**

**IRQ\_Callback();**

**}**

**else**

**{**

**\_\_NOP();**

**}**

**}**

/\* USER CODE END 4 \*/

We also delete a string array in this file, it is declared now in another place

~~char str1[150];~~

Also, in the main () function, we change a little initialization, since we now have an indication of the other. It remains only now that

/\* USER CODE BEGIN 2 \*/

**LCD\_ini();**

**NRF24\_ini();**

/\* USER CODE END 2 \*/

Now we need to play **around** with the display, so first go to the **lcd.c** file and start making corrections there.

Let's fix the bus number I2C

extern I2C\_HandleTypeDef hi2c**3**;

Let's fix here too

HAL\_I2C\_Master\_Transmit(&hi2c**3**,(uint16\_t) 0x4E,buf,1,1000);

Now our project will have to be assembled unmistakably.

If we do it, then most likely we will remove the boxes on the display.

We make some changes to the function of sending a half byte to the display controller

void sendhalfbyte(uint8\_t c)

{

  c<<=4;

  LCD\_WriteByteI2CLCD(portlcd|c);

**LCD\_WriteByteI2CLCD((portlcd|=0x04)|c);**//включаем линию E

  DelayMicro(**1**);

**LCD\_WriteByteI2CLCD((portlcd&=~0x04)|c);//**выключаем линию E

  DelayMicro(50);

}

Now here a few things happen differently. We first send the data, turn on the line, then wait for the microsecond and then disconnect the line.

From the positioning function, we will remove all delays

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

{

  switch(y)

  {

    case 0:

      sendbyte(x|0x80,0);

~~HAL\_Delay(1);~~

      break;

    case 1:

      sendbyte((0x40+x)|0x80,0);

~~HAL\_Delay(1);~~

      break;

    case 2:

      sendbyte((0x14+x)|0x80,0);

~~HAL\_Delay(1);~~

      break;

    case 3:

      sendbyte((0x54+x)|0x80,0);

~~HAL\_Delay(1);~~

      break;

  }

}

The initialization function is also slightly rewritten

void LCD\_ini(void)

{

  HAL\_Delay(**50**);

**LCD\_WriteByteI2CLCD(0);**

HAL\_Delay**(100);**

  sendhalfbyte(0x03);

**DelayMicro(4500);**

  sendhalfbyte(0x03);

**DelayMicro(4500);**

  sendhalfbyte(0x03);

**DelayMicro(200);**

  sendhalfbyte(0x02);

  sendbyte(0x28,0);// mode 4 bits, 2 lines (for our large display it's 4 lines, font 5х8

  sendbyte(0x0C,0);// the display is turned on (D = 1), the cursors are not needed

  HAL\_Delay(1);

  sendbyte(0x01,0);// уберем мусор

  HAL\_Delay(2);

  sendbyte(0x06,0);// пишем влево

**HAL\_Delay(1);**

**sendbyte(0x02,0);//курсор на место**

**HAL\_Delay(2);**

  setled();//подсветка

  setwrite();//запись

}

With a display like everything.

We go to file **NRF24.c** and start compose, as we will catch the packets in the interrupt handler.

Our task is to ensure that the code in the handler is as small as possible, otherwise we run the risk of missing packets. We will process the received data in the reception function, which is still called in our system cyclically in an infinite loop.

In the function  **IRQ\_Callback** we add several local variables

void IRQ\_Callback(void)

{

**uint8\_t status=0x01;**

**uint8\_t pipe;**

**uint16\_t dt=0;**

10 microseconds, we still have to wait

uint16\_t dt=0;

**DelayMicro(10);**

Flashing LED

DelayMicro(10);

**LED\_TGL;**

We recognize the status, and if it's the one we need, we'll count the buffer in the array and reset the flag

LED\_TGL;

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

**if(status & 0x40)**

**{**

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

**NRF24\_WriteReg(STATUS, 0x40);**

**}**

We will collect the project and we will sew the controller. The green LED should begin to flash approximately once per second in sync with the transmitter's LED.

If this is the case, then now we will need to somehow display our data, which we received from the receiver, on the display. Only we will produce all this mapping in the function of the other, not in the handler. Only we will need to talk about this somehow this function. We will do this with the help of a flag. To do this, add a global variable

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

**volatile uint8\_t rx\_flag = 0;**

Let's return to our **IRQ\_Callback** handler  and **turn on the** flag

  NRF24\_WriteReg(STATUS, 0x40);

**rx\_flag = 1;**

}

Now we go to function **NRF24L01\_Receive** , trace the included flag, display the information from the buffer on the display and reset the flag, so as not to get stuck

void NRF24L01\_Receive(void)

{

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

**{**

**LCD\_SetPos(0, 0);**

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

**LCD\_String(str1);**

**rx\_flag = 0;**

**}**

}

We collect the project, we flash the controller. Also, the controller of the transmitter will be reset, so that attempts to transmit errors are dropped in order to track that everything is transmitted unerringly



Only we will accept today also the number of erroneously transmitted packets from our transmitters, so we will slightly increase our buffer

#define TX\_PLOAD\_WIDTH **7**

Also, the **NRF24L01\_Receive** function  **displays the** output to show all three types of data - the number of the account, the number of attempts at errors and the number of unrequested packets

sprintf(str1,"%5u %5u **%5u**", \*(int16\_t\*)RX\_BUF, \*(int16\_t\*)(RX\_BUF+2)**, \*(int16\_t\*)(RX\_BUF+4));**

We will collect the code, we will sew the controller. It is clear that now we will not have the correct reception of packets, since we need to fix the size of the buffer in the transmitter project.

In the [**next part of the**](http://narodstream.ru/stm-urok-113-nrf24l01-neskolko-peredatchikov-chast-2/) lesson we will prepare the projects for two transmitters, also in the receiver project we will add the ability to receive packets from two transmitters and display them in different lines of the display and check our code in practice.

**STM Lesson 113. NRF24L01. Several transmitters. Part 2**

Posted on [March 29, 2018](http://narodstream.ru/stm-urok-113-nrf24l01-neskolko-peredatchikov-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 by STM32](http://narodstream.ru/rub_stm32/)- [No Comments ↓](http://narodstream.ru/stm-urok-113-nrf24l01-neskolko-peredatchikov-chast-2/#respond)

[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/BiBWUdqgn5i50D01CPI-ari00000ECgs7402I09Wl0Xe172WmvR10O01sE3CgmU80QE8tP0fa06cakID99W1eiM-tYEW0SAapOqag07iWR3U8xW1tAUPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y083e0BUkAe2kG8nU-_3cu8YR_02kgZIv0Bu0eA0W820m1EO0wKVg0CIi0C4k0J_0UW4gmJu18J70OW5XCS1a0NctW6W1Tixg0NDCh05pJAu1T8um0NmcGV81R3g0T05uXRW1J_m1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XH3Adc5yL8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2A-SbG602W712W0000000F0_s0e2u0g0YNhu2i3y5OWB1geB4F1oonluKG00baV89bjr1G302u2Z1SWBWDIJ0TaB-54CgUONnKZe2uJ70V0B1eWCjfNUlW7e30AO3PMCQF8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_ue-7Aulp5uaW0?stat-id=3&test-tag=85761936351233&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY3IjoiODU3NjE5MDY5OTkyOTYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Buy robot constructorTeach your child to program. 6900 for a set of shares. OrderTo learn moredorado-uno.ru](https://an.yandex.ru/count/Aj1TnXXARYa50D41CPM-ari00000ECgs7402I09Wl0Xe172AffoG1e01xCtW6OW1rBMagJMG0VZAnCelc07gZ-3LBg01zih4oY-e0UYEuDKkk07st_Rr6S010jW1ieMB5U01qBto4UW1OFW1elhUlW680WIW0g2-sHQv0Z5xxyERWY9ly0BXZwgr2FW2We20W8304u03xuV0-WM80vs3l_qAi0C4w0I10_W4vEi1Y0Nawm6G1RlN0g05taQe1Vaxi0NvExW5gLV01R3x3CW5yAK3q0Ny6k05Ml050PW6tgcE6Q06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00071n0000gGVuKMn831Z5IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0YGtwmBW0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mYg2n3En-uJ-54006slqIPRTGK0m0k0emN82u3Kam7P2_XHR4WC6CL8w0lawm7m2mQ83ABwthu1w0m2c0tylMRo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Em8Gzi0u1eGy00000003mFwWF-AE1phM1nk98?stat-id=4&test-tag=85761936351233&format-type=24&banner-test-tags=eyI2Mzg2Mzg5NjkyIjoiODU3NjE5MDY5OTkyOTYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

In the [**previous part of the**](http://narodstream.ru/stm-urok-113-nrf24l01-neskolko-peredatchikov-chast-1/) lesson we connected the symbolic LCD-display to the receiver board, set up the project for it, also corrected our libraries a little.

We also use only one transmitter, only for him we will start a new project, made from the project of [lesson 105](http://narodstream.ru/stm-urok-105-nrf24l01-peredayom-dannye-chast-1/)**NRF24\_TX**  and call it **NRF24\_TX\_PIPE0** .

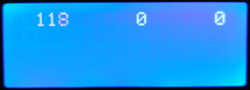
The receiver is disconnected from the PC, we supply it separately, we connect the transmitter circuit with the programmer to the PC and we will do our project.

Run it in the Cube MX, without touching anything, generate a project for Keil, open it in it, configure the programmer for auto- **reload, enable the** optimization level in **1** , connect the file **NRF24.c** and try to assemble the project.

If the project is going well, then go to the file **NRF24.c** and add there first our buffer

#define TX\_PLOAD\_WIDTH **7**

If we collect now and sew our project, then we will have everything correctly transferred



Only we wanted to send more information about the completely unreleased packages.

In principle, we already received such information, since we considered the **OBSERVE\_TX** register , in which there are both indicators, we simply did not send this information. Therefore, in this file we do not touch anything else and go to the file **main.c** and first in the **main ()** function we will slightly correct our local variables

/\* USER CODE BEGIN 1 \*/

**uint8\_t dt;**

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

/\* USER CODE END 1 \*/

Now we go to an endless cycle and start making corrections there as well.

First of all, we will remove all restrictions related to the lack of digits on the indicator, we now have no such problems, we have a display that is compatible

~~if(retr\_cnt\_full>999) retr\_cnt\_full=999;~~

~~if(i>=999) i=1;~~

We will send the number of completely unreleased packets, that is, those that were not transmitted to the receiver, even when using the maximum number of attempts. We'll put them in the buffer at 4

memcpy(buf1+2,(uint8\_t\*)&retr\_cnt\_full,2);

**memcpy(buf1+4,(uint8\_t\*)&cnt\_lost,2);**

Parameter data from the register with errors will be placed immediately in the desired variable

**retr\_cnt\_full** += dt & 0xF;

Also consider the data on non-transmitted packets

retr\_cnt\_full += dt & 0xF;

**cnt\_lost = dt>>4;**

But this line will not be necessary to us now either

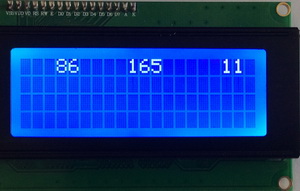
~~retr\_cnt\_full += retr\_cnt;~~

Packet errors are considered only up to 15, then they are not reset. It is not so easy to drop them, but we do not need them. If you get 15 errors, it will already be clear that we have something wrong, and so it will not work. Therefore, this is enough for us.

We also remove our previously commented code from the infinite loop, I think it will not be useful to us.

We will collect the code through the controller and make sure that our packets are also normally transmitted.

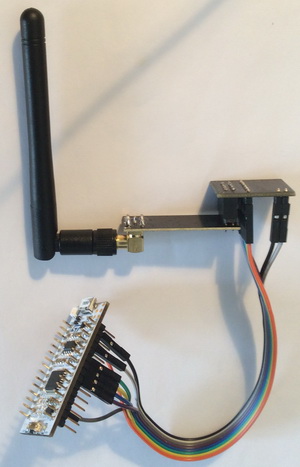
And to try that we have mistakes also are considered, then it is enough to disconnect the receiver from the power supply for a while (seconds to 10) and then turn it on again. We will see this picture



We have run 11 errors, as well as 165 transmission attempts, it all converges. 11 × 15 = 165. So everything works.

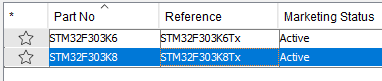
We can say that the preparatory measures to achieve the goal of the lesson are carried out. And we clearly remember the purpose of the lesson - to transfer data to the receiver from several transmitters - or rather from three.

Therefore, we now also power the transmitter separately from the PC and now connect the next transmitter. assembled on a convenient small motherboard **Nucleo F303K8** . This board we already used in lessons on modules HC-05, so there is no difficulty in connecting the module to it, I think it does not arise. Moreover, the programmer and the USART adapter we do not need to connect to this board, everything is already on board in it. We will connect the next NRF module to this board, we will get here such a pretty scheme



Which feet to connect, you can see in the project for the Cube MX, which we now create.

Run the Cube MX, select our controller



Turn on clocking

http://narodstream.ru/wp-content/uploads/2018/03/stm113img17.png

Turn on debugging

http://narodstream.ru/wp-content/uploads/2018/03/stm113img18.png

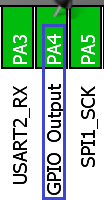
Enable **SPI**

http://narodstream.ru/wp-content/uploads/2018/03/stm113img19.png

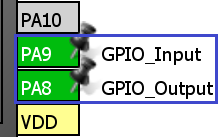
Will include **USART2** , it may come in handy

http://narodstream.ru/wp-content/uploads/2018/03/stm113img16.png

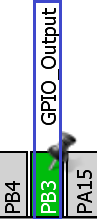
For the **CS** foot we turn on the foot **PA4**



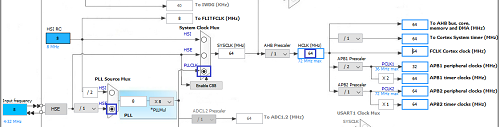
For the **CE** foot, **turn on the** foot **PA8** , and for the foot **IRQ** - the foot **PA9**



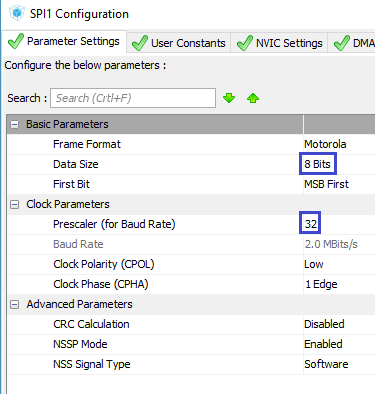
Also on the output we **turn** on the leg **PB3** , connected to the LED



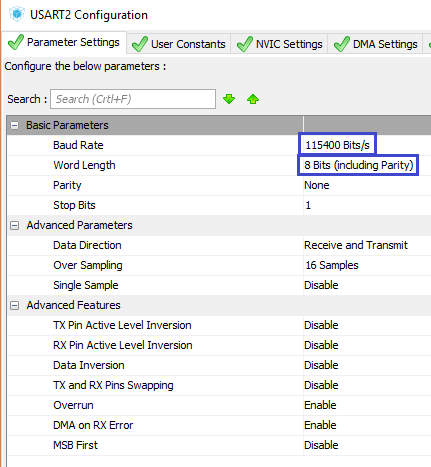
Now adjust our frequencies in the **Clock Configuration** tab (click on the picture to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2018/03/stm113img23.png)

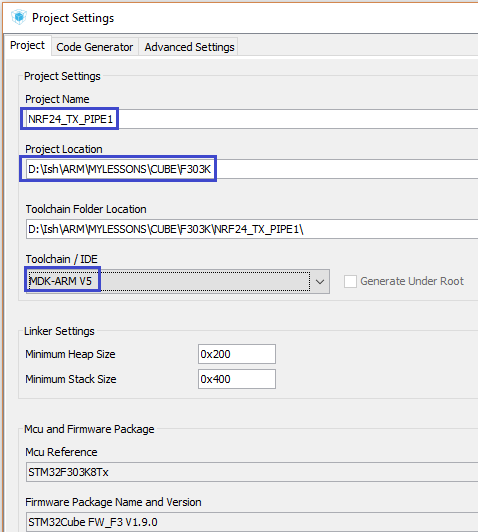
Now go to **Configuration** and configure the **SPI** bus



Set up **USART**



We go into the project settings, set the project name, select the programming environment and the path to the project



Save the settings, generate the project and open it in Keil.

Set up the programmer for auto-cutting, turn on the optimization level in **1** and try to collect our project.

If everything is going well, then from the project of our first transmitter we copy to our new project files for the modules **NRF24.c** and **NRF24.h** . The file **NRF24.c will** also be connected to the project tree.

Once again, we'll try to assemble the project, although he, of course, is unlikely to make any mistakes, but nevertheless, we will have the opportunity to navigate through the new files.

Let's **go** to file  **NRF24.h** and change the number of the connecting library

#include "stm32f**3**xx\_hal.h"

Also we will change some service legs

#define CE\_PIN GPIO\_PIN\_**8**

...

#define IRQ\_PIN GPIO\_PIN\_**9**

...

#define LED\_GPIO\_PORT GPIO**B**

#define LED\_PIN GPIO\_PIN\_**3**

And, since the LED in this board is connected standardly to the common wire, and not to the power supply, we will invert its state

#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 **move** on to file  **NRF24.c** and change something there too.

First, you can not allow two transmitters to have the same address, so change the address

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

Also one more desirable condition. When we try to make several attempts to send the same packet, we can also change the time between these attempts. If this time for two different transmitters is the same, then if there is a simultaneous transfer of packets from them to the receiver, then repeated attempts to transfer these packets will also coincide. Let it be a rare situation, but if it is possible to overcome it, then it is better to take advantage of this. Therefore, let's change the given time in the initialization function of the module

NRF24\_WriteReg(SETUP\_RETR, **0x6F**); // // **1750** us, 15 retrans

Go to **main.c** and transfer all the user code (or almost all) to it from the project file of the first transmitter of the same name

/\* USER CODE BEGIN Includes \*/

**#include "NRF24.h"**

**#include <string.h>**

/\* USER CODE END Includes \*/

...

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

**char str1[20]={0};**

**uint8\_t buf1[20]={0};**

/\* USER CODE END PV \*/

...

/\* USER CODE BEGIN 1 \*/

**uint8\_t dt;**

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

/\* USER CODE END 1 \*/

...

/\* USER CODE BEGIN 2 \*/

**NRF24\_ini();**

/\* USER CODE END 2 \*/

...

/\* USER CODE BEGIN 3 \*/

**HAL\_Delay(691);**

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

**memcpy(buf1+2,(uint8\_t\*)&retr\_cnt\_full,2);**

**memcpy(buf1+4,(uint8\_t\*)&cnt\_lost,2);**

**dt = NRF24L01\_Send(buf1);**

**retr\_cnt\_full += dt & 0xF;**

**cnt\_lost = dt>>4;**

**i++;**

}

/\* USER CODE END 3 \*/

I specifically highlighted the delay time with another color, as it will be different, so much more interesting. We from this module will transmit packets a little more often.

We will collect our code, we will impose the controller.

True to the sense of this until we do not wait, the receiver "does not know" about our transmitter, he does not even know his address.

Therefore, we power our second transmitter from a separate power supply, the receiver board is connected to the PC.

Let's **go** to file  **NRF24.c** and add the address of the second transmitter, slightly changing the name of the first address for beauty

uint8\_t TX\_ADDRESS**0**[TX\_ADR\_WIDTH] = {0xb3,0xb4,0x01};

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

In connection with changing the name of the address array, in the function **NRF24L01\_TX\_Mode we will** also make a correction

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

In the initialization function, we will also correct the name in some lines, and also add one for the second transmitter

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

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

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

For the first transmitter we will now have the **PIPE0** channel **set up** , and **PIPE1** will now be for the second one.

Only this is not all. We now need to include both channels in the corresponding register, so for it the string is also corrected

NRF24\_WriteReg(EN\_AA, **0x03**); // Enable Pipe0 **and Pipe1**

NRF24\_WriteReg(EN\_RXADDR, **0x03**); // Enable Pipe0 **and Pipe1**

Add one more line for setting the packet size for the first channel

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

**NRF24\_WriteReg(RX\_PW\_P0, TX\_PLOAD\_WIDTH); //Number of bytes in RX payload in data pipe 0**

NRF24\_WriteReg(RX\_PW\_P1, TX\_PLOAD\_WIDTH); //Number of bytes in RX payload in data pipe 1

If we collect the project and patch the controller, then we'll see the mapping of the packets from both transmitters, and it will look very chaotic, but interesting. The thing is. that we display everything in one line. For the beauty of the picture, we will display the contents of the packages on different lines. For each transmitter there will be a line, not for nothing that these lines have a whole 4 on the display. But before sending the packet data to a certain line, it is still necessary to determine which particular transmitter (address) this packet came from. All this we have in the status register. When we were just starting to learn how to receive data, we tracked the channel number through the terminal program.

To do this, we define the channel on which the packet came in **IRQ\_Callback** function  . Extract the channel number from the status register

if(status & 0x40)

{

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

Next, send the channel number to the buffer

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

**\*(RX\_BUF+7) = pipe;**

In this regard, slightly increase the buffer

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

You could just create a global variable, but I just decided to play around with the array, still we already have it.

Now, in the **NRF24L01\_Receive** function,  **we** fix the positioning

LCD\_SetPos(0, **\*(RX\_BUF+7)**);

We'll collect the code, we'll tell the controller, and after that all our packages are normally distributed on the display screen



In the [**next part of the**](http://narodstream.ru/stm-urok-113-nrf24l01-neskolko-peredatchikov-chast-3/) lesson we will configure the project for the third transmitter, in the receiver we will add the ability to receive packets from three sources and check the code in practice.

**STM Lesson 113. NRF24L01. Several transmitters. Part 3**

Posted on [March 30, 2018](http://narodstream.ru/stm-urok-113-nrf24l01-neskolko-peredatchikov-chast-3/)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-113-nrf24l01-neskolko-peredatchikov-chast-3/#respond)

[Datasheets of electronic components**Dateshots are** free of charge and without registration. Cases, marking, electrical parameters.To learn morerlocman.ru](https://an.yandex.ru/count/Oh8wes521sW50Du1CQ6-ari00000ECgs7402I09Wl0Xe172yqvon0O01sPlr2uW1ovh_-YkG0VodeFGdc07UvhsW9g01bggCz2Ue0TpclQ0ck07U_Dx-5i010jW1YCw01k01xjlP2UW15FW1hjNYlW680WEW0k03kGApDxWO0CzNMyW2hQZtp_sXa9S2y0AHcVcV2FW2We20W8204e03-8tKHuW3qwcLfmUO0w231QW34h031BW4_m7e1AuE-0IJOeW5asAG1Trie0Mv7QW5-XYm1VeOk0NI6y05Wz82o0NuLT05dGlW1Gdm1G6O1fxIjsgW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO002rmSG000Aa7-54aXmCRnKYm1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08y0602W712W0000000F0_s0e2u0g0YNhu2i3y5OWB1geB4FhEIGxuKG00Et-F9rjr1G302u2Z1SWBWDIJ0TaB-54aXmCRnKZe2vDYy0i6Y0okrUA-0UWC0fWDbOneyWq0-Wq0WWu0YGu00000003mFv0Em8Gzc0x6rypKbOdWkXEW3i24FR0E0Q4F00000000y3-e3_YZqRcjuCRYI000?stat-id=3&test-tag=79164866584577&format-type=24&banner-test-tags=eyI1MzU2MzkwOTEwIjoiNzkxNjQ4MzcyMzI2NDAifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Fundamentals of Programming LanguagesDownload the free book and become a real programmer. Get it now:To learn moresheremetev.info](https://an.yandex.ru/count/4cVquTHiaCO50De1CQE-ari00000ECgs7402I09Wl0Xe172WmvR10O01ZSt-9uW1jeRTa2cG0VYKhOyac07okhxU8w01lfFCZIIe0PB-hzuZk062f9dq5S010jW1bh2j4E01vfJJ3kW15lW1fkdUlW680WIW0jwugWAv0hCtk1W0prTRy0Asywq6-0A2W820W80IW0E9_8dz3OW3XiBkaWIO0_BJ1R031BW4_m7e1986-0JNf0681TUa0P05u9O2e0MqCQW5xocm1Uyfk0NlHi05y9a7o0MmwW7G1UiIu0K-y0K1c0Q0qApp3g06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00071n0000gGVuKTpYMHl5IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0Yld9K1W0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mYg2n2BWTui-54006eMfITRTGK0m0k0emN82u3Kam7P2_XHtE9P6yL8w0lNf07m2mQ83ARfthu1w0m2c0tylMRo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Em8Gzi0u1eGy00000003mFwWF-AFHefRhnk98?stat-id=4&test-tag=79164866584577&format-type=24&banner-test-tags=eyI0ODQ3OTM2ODI4IjoiNzkxNjQ4MzcyMzI2NDAifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

In the [**previous part of the**](http://narodstream.ru/stm-urok-113-nrf24l01-neskolko-peredatchikov-chast-2/) lesson we prepared the projects for two transmitters, also the receiver project was added the ability to receive packets from two transmitters and display them in different lines of the display and checked our code in practice.

Now let's deal with the third transmitter. The scheme for it will be the same as for the second one. We now connect the third transmitter to the PC, and connect the receiver board to a separate power supply.

The project for the third transmitter we will do from the project for the second one, and we will already call **NRF24\_TX\_PIPE2** .

Run this project in Cube MX, and, without touching it, generate the project and open it in Keil. Set up the programmer for auto-cutting, optimize it in **1** , connect the file  **NRF24.c** and try to assemble the project. We will not yet be stitching, since the same addresses will inevitably collide on the receiver.

Let's **go** to file  **NRF24.c** and change the address

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

Why do we change the first byte?

Yes, because when we studied the channels, we saw that since PIPE2, we can not completely specify addresses, and we can only specify the lower part of the address, which is located in the address buffer in the first (or, to be exact, on the zero) cell.

Also, we change the time between attempts to retransmit the packet in the initialization function

NRF24\_WriteReg(SETUP\_RETR, **0x8F**); // **2250** us, 15 retrans

In the **main.c** file in the infinite loop of **main (), we** change the delay time

/\* USER CODE BEGIN 3 \*/

  HAL\_Delay(**1358**);

We will collect the project and we will sew the controller.

Again, we will not see anything from this transmitter on the receiver's display, since we still did not tune the receiver to the address of the third transmitter.

Therefore, we connect the third transmitter board to the separate power supply, and connect the receiver board to the PC again.

Let's move to the receiver project in the **NRF24.h** file   and add macros for the third channel

#define RX\_ADDR\_P1 0x0B //'RX address pipe1' register address

**#define RX\_ADDR\_P2 0x0C //'RX address pipe2' register address**

#define TX\_ADDR 0x10 //'TX address' register address

#define RX\_PW\_P0 0x11 //'RX payload width, pipe0' register address

#define RX\_PW\_P1 0x12 //'RX payload width, pipe1' register address

**#define RX\_PW\_P2 0x13 //'RX payload width, pipe2' register address**

Then we'll go into the file  **NRF24.c** and add the third address, or rather its lower byte

#define TX\_PLOAD\_WIDTH 7

**#define TX\_ADDRESS2 0xb6**

We ask only the lower part of the address.

Let's pass now to the initialization function and first turn on the third channel

NRF24\_WriteReg(EN\_AA, **0x07**); // Enable Pipe0, Pipe1 **and Pipe2**

NRF24\_WriteReg(EN\_RXADDR, **0x07**); // Enable Pipe0, Pipe1 **and Pipe2**

Let's pass the address to the module (rather the lower part of the address, so we are already using the function not for the buffer, but for the ordinary register)

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

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

Also set the packet size for the third channel

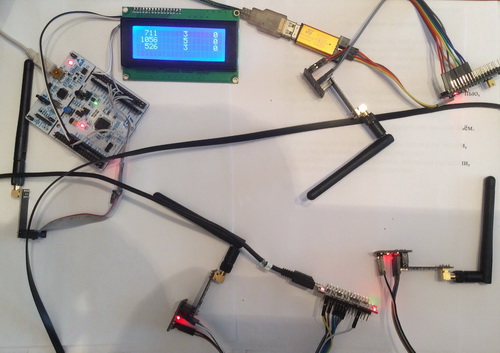
NRF24\_WriteReg(RX\_PW\_P1, TX\_PLOAD\_WIDTH); //Number of bytes in RX payload in data pipe 1

**NRF24\_WriteReg(RX\_PW\_P2, TX\_PLOAD\_WIDTH); //Number of bytes in RX payload in data pipe 2**

We will collect the project, we will sew the controller and see the following picture on our display



And that's how our whole scheme looks like (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2018/03/stm113img29.jpeg)

Now the goal of the session is achieved. We simultaneously receive from three transmitters the packets in one receiver, displaying their contents visually on the display, also learned how to work with the breakpoint of the module in hardware, which added to our project the professionalism

Thank you all for attention!