**STM Lesson 103. Modules NRF24L01. Part 1**

Posted on [December 28, 2017](http://narodstream.ru/stm-urok-103-moduli-nrf24l01-chast1/)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-103-moduli-nrf24l01-chast1/#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/12nR_l18Xk850Dq1CJIzari00000ECgs7402I09Wl0Xe172WmvR10O01sE3CgmU80QE8tP0fa06cakID99W1eiM-tYEW0SAapOqag07iWR3U8xW1tAUPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y083e0BUkAe2kGA6wuoM7K3BmV02nyIUnmpu0eA0W820i1E00-RYgP81Y0EIhhwt29W3ky06g0CIi0C4k0J_0UW4ZWJu1A6s0OW5eRO1a0MunWAW1TSsg0MhBh05gowu1VzIm0NmcGV81R3g0T05x1JW1GRm1G6O1eBGhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XHLA03nCH8i0U0W90Cq0S2u0U62lW70O080T08keg0WS0Gu0Yld9K1W0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mQg2n3jbU6x-54008w5pn5RTGK0m0k0emN82u3Kam7P2_XHLA03nCH8w0kXjW7m2mQ83BQLthu1w0m2c0sLZ6Zo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Em8Gzi0u1eGy00000003mFwWF-AEnZAo0iU98?stat-id=3&test-tag=85212180537345&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY3IjoiODUyMTIxNTExODU0MDgifQ%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/RFdIOZhL66K50D41CJMzari00000ECgs7402I09Wl0Xe172AffoG1e01xCtW6OW1rBMagJMG0VZAnCelc07gZ-3LBg01zih4oY-e0UYEuDKkk07st_Rr6S010jW1ieMB5U01qBto4UW1OFW1elhUlW680WIW0g2-sHQv0eRhZ9OTGCl1y0AjaDVJ2FW2We20W82m4u03cuYDb0-80_p-_FuFi0C4w0J80lW4kg82Y0MweWAG1PFS0g05aLUe1Un9i0NiIRW5mLZ01R3x3CW5yAK3q0MT8U05Ml050PW6Zhkm6Q06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00071n0000gGVuKGGL9iJ4IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0YGtwmBW0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mYg2n2C8rka-5400D8Ys15RTGK0m0k0emN82u3Kam7P2_XH11KcnCH8w0kweWBm2mQ83ABwthu1w0m2c0tylMRo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Em8Gzi0u1eGy00000003mFwWF-AE1sT24iU98?stat-id=4&test-tag=85212180537345&format-type=24&banner-test-tags=eyI2Mzg2Mzg5NjkyIjoiODUyMTIxNTExODU0MDgifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

We continue the topic of wireless data transmission.

We got acquainted and tried the work of the HC-05 modules working on Bluetooth technology, and now let's get acquainted, and then in the following classes and work with such wireless modules as **NRF24L01** from Nordic Semiconductor. These modules differ from devices that work with Bluetooth technology, as well as Wi-Fi, although many even believe that this is the same thing. Firstly, the protocol is different, the technologies are different, although these technologies work at practically the same frequency. The NRF modules operate at 2.4 MHz and are characterized by low power consumption, but at the same time a high transmission range. Modules with an antenna, with which we will subsequently work and write code for them, are capable of transmitting at distances up to 1 km, and in practice it was possible to transmit data up to 1.5 km. That's why we can not bypass this technology by any means.

These modules look like this way



These are modules with an antenna, but they are also inexpensive. There is still no antenna, they are even cheaper, but their range will be less. But energy consumption will also be much less. Also these modules are very demanding to power, or rather to the quality of it, although the specification has a voltage regulator built into the chip. Usually, inexpensive modules with a stabilizer are used at 3.3 volts and at the output still hang a capacitor of 10 μF. I did exactly the same thing, and the capacitor for one module was set at 50, and on the other 100 μf. It's just that I got such things on hand. It seems to work normally.

Let's get acquainted with the characteristics of these modules in more detail.

Transmission speed of the transmitter is 250 kbps, 1 Mbps and 2 Mbps.

The transmit power can also be adjusted. There are special bits in special registers for this.

The current consumption of the module in different conditions is different. It depends on many factors. Most of all, in what state the module is at the moment. In the transmission mode with a power of 0 db, the module consumes 11.3 mA. In the receive mode at the maximum speed of 2 Mbps - 13.5 mA. In power down mode - only 900 nA. In standby mode - 26 mkA.

The module feeds from 1.6 to 3.6 V. Only the power supply should be very stable and should not depend on anything and the best option is 3.3 V.

Also announced are certain channel technologies that provide guaranteed data delivery ( **ShockBurst ™**and **MultiCeiver ™** ). Perhaps this is also the reason for such a keen interest in these receivers.

Up to 126 possible radio channels with a band difference of 1 MHz.

Up to 6 channels of information exchange, with which the module can simultaneously operate in the receiver mode.

Information inputs are tolerant to 5 volts.

Compact housing 20-pin 4x4mm QFN.

In the **next part of the** lesson we will study some more features and features of the module, learn about practical circuits for connecting the receiver and transmitter, prepare the module's power, and create and configure a project to work with it in the programming environment.

**STM Lesson 103. Modules NRF24L01. Part 2**

Posted on [December 29, 2017](http://narodstream.ru/stm-urok-103-moduli-nrf24l01-chast2/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Published in [Programming STM32](http://narodstream.ru/rub_stm32/)- [No Comments ↓](http://narodstream.ru/stm-urok-103-moduli-nrf24l01-chast2/#respond)

[Programming CircleDownload the free book and become a real programmer. Get it now:To learn moresheremetev.info](https://an.yandex.ru/count/Opb4cuZR_Tq50Dy1CK-zari00000ECgs7402I09Wl0Xe172WmvR10O01rAB4cWQ80QE5tP0fa060kUAD99W1XhQ-tYEW0Q2Hp8qag07i-w_U8xW1tA6Pz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y083e0BUkAe2kG84XsTDoodZFl02t-YCx0Ju0eA0W820e1600xEqXviEY0Eaa-Zc2fW3q-O4g0CIi0C4k0J_0UW4WWNu1EJj0OW5vEq1a0MMaWAW1Qj7g0MwFB05kZou1Tf5m0NmcGV81R3g0T05cXlW1Lhm1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XHt8XpoiH8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2A-SbG602W712W0000000F0_s0e2u0g0YNhu2i3y5OWB1geB4DhWW6RuKG00mtzh4rjr1G302u2Z1SWBWDIJ0TaB-57SY7FAn4Ze2-Jj0V0B1eWCjfNUlW7e30AO3PMCQF8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_ueu6lgSwouaW0?stat-id=3&test-tag=75866331701249&format-type=24&banner-test-tags=eyI0ODQ2Mjc2MTYwIjoiNzU4NjYzMDIzNDkzMTIifQ%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/RTvSShcnhti50D41CL2zari00000ECgs7402I09Wl0Xe172AffoG1e01xCtW6OW1rBMagJMG0VZAnCelc07gZ-3LBg01zih4oY-e0UYEuDKkk07st_Rr6S010jW1ieMB5U01qBto4UW1OFW1elhUlW680WIW0g2-sHQv0WI7PqtBAUC-y0BnZCkD2VW2We20W82W4O03cuYDb0-80_p-_FuFi0C4w0Ik0lW4aPy2Y0MHdmAG1ShT0g05abQe1O19i0M0IRW5y5Z01R3x3CW5yAK3q0Ni8E05Ml050PW6Zhkm6Q06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W0001050000gGVuKOmAayh4IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0YGtwmBW0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mYg2n1afXRu-54003TWSnDRTGK0m0k0emN82u3Kam7P2_XHZ0gJoiH8w0kHdmBm2mQ83ABwthu1w0m2c0tylMRo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Em8Gzi0u1eGy00000003mFwWF-AFHoQ3Iik98?stat-id=4&test-tag=75866331701249&format-type=24&banner-test-tags=eyI2Mzg2Mzg5NjkyIjoiNzU4NjYzMDIzNDkzMTIifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

In the **previous part of the** lesson we got acquainted with some characteristics of the NRF24L01 module.

The process of data exchange of the module with the controller takes place via the SPI bus with the use of some more legs. There is a foot **CE** , which controls the activation of the module, as well as the inclusion of the module in a certain mode. There is also an **IRQ** foot that changes its level at a moment of certain events, for example, when the packet has completely left, or a noobot, when we received the packet and it is already in the FIFO buffer. Also, the level changes when the maximum repetition limit for sending a packet is reached.

Basically, these are the characteristics. And with all the features we will already get acquainted in the process of using.

For example, there is a feature that the speed is stated to be 2 Mbps, but in fact, with this speed, we certainly will not transmit the data stream, since data packets can not exceed 32 bytes, since such memory cells are contained in the FIFO buffer. And we can not accept the packet for the packet, we still have to release the buffer once, there are also special impulses of a certain duration for sending the data. There is also a timing for transferring the module from one mode to another - 135 μs. For example, to transfer from receiver mode to transmitter mode and vice versa. I also read that it is necessary to re-calibrate the module at least 4 milliseconds, which also takes about 200 μs, first I did not recalculate anything and the data was normally transmitted and received. The only thing, that sometimes when you turn on the modules, the transfer and reception did not start right away, and sometimes it was necessary to restart one of the modules for start-up, and sometimes to a cold one, that is, with a power outage. But nevertheless I managed to start the modules, I managed to transfer the data. But then, after examining the mass of source codes (you can not even imagine which one), I came to the conclusion that if we do not transmit packets more than once every 4 milliseconds, and there is a risk of losing communication, for example, due to battery shrinkage on one of the exchange devices, it is better after transferring the data to transfer the module to the receiver mode and already when we transfer the next packet, transfer it again to the transmitter mode, waiting for all timings. The technical documentation states that if the module is idle for more than 4 milliseconds, it will go into standby mode, so it will have to be output from there. I do not even know why this worked without it. Also I managed to track by certain bits in which of the 6 channels the packet was addressed. Otherwise, I would not have started a lesson in modulus.

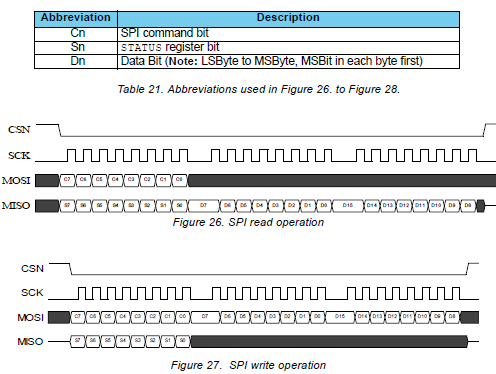
So we continue further.

Also, we can set the transmitter to such a mode that each packet the receiving module will confirm and send a checksum, the digit of which is also configured.

The technical documentation also details the process of data transmission itself at a low level. I think this is not very interesting for us, since we can not manage this process anyway, we can not change preambles and other subtleties. So let's at least see how the process of data exchange of the module with the microcontroller should occur.

In general, the module supports data exchange with the controller at speeds up to 10 Mbps.

This is the process of exchanging module data with the controller in the receive and transmit mode



As we see, both in the receive mode and in the SPI data transfer mode, the first command is transmitted, and then the data is transmitted or received. I just think that we are not completely crazy about the theory, let's still go down to earth a little, create and configure the project, at least while for the transmitter, connect the circuit, start writing code and then in the course of writing, when we have to directly apply the commands and registers, we will get acquainted with them as we go along. And writing will be more fun with feeding up the information and even if we are already studying everything, we will forget half of it by the time of writing and we will still have to jump over the text, and if we watch in the video lesson mode, then unwind. Somehow it is not good.

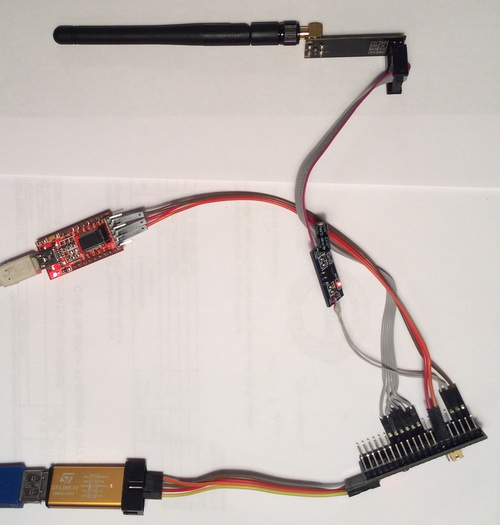
The main goal of this lesson is to study the commands and registers of the module we need to use to write the project code, and also write a project in which we initialize the module, learn how to write data to registers, and also read them from there. I think this will be enough for the first lesson. And in the next lessons we will already connect the two modules, we will test them for transmission and reception, and in the subsequent lessons we will try to transfer some useful data, for example the ambient temperature, and on the receiving device it is displayed on some display or on the LED indicator .

Therefore, we will think about the project.

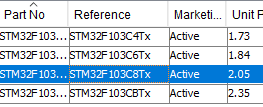
For the transmitter, today we will apply a small debug card based on the **STM32F103C8T6** controller , which has been faith and truth for a long time.

We connect the module to the board, without forgetting to connect a stabilizer with a capacitor to the power supply, and also connect a USB-TTL adapter through which we will monitor the state of some registers. The legs for the connection we will see in the Cube MX.

Our scheme is as follows



Well, let's start the Cube MX project generator, create a new project, select the controller



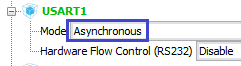
Let's add a resonator

http://narodstream.ru/wp-content/uploads/2017/12/stm103img05.png

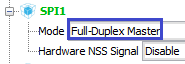
Turn on the debugger

http://narodstream.ru/wp-content/uploads/2017/12/stm103img06.png

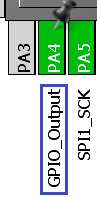
Turn on USART



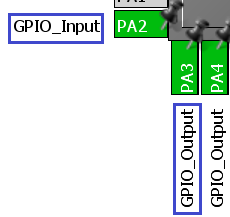
Enable SPI



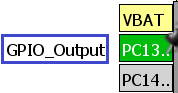
We plug foot PA4 to the output. This will be the foot for the CSN contact



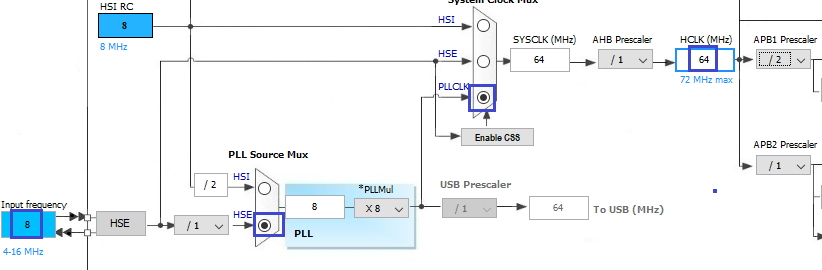
Also, we plug the PA3 leg to the output for the CE contact, and also the foot PA2 to the IRQ interrupt contact input



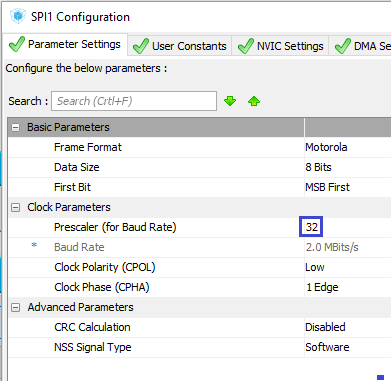
The PC13 pin is also used on the output, to control the LED



Go to Clock Configuration and configure the board at 64 MHz, I think, enough for the eyes (click on the image to enlarge the image)

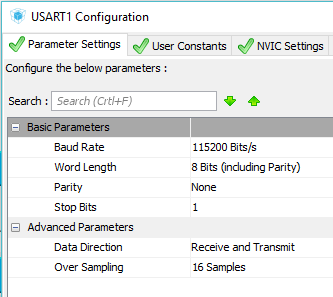
[](http://narodstream.ru/wp-content/uploads/2017/12/stm103img12.png)

Go to **Configuration** and configure the first case of SPI

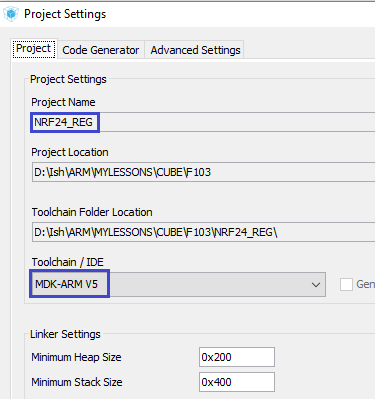


I think this speed is enough for us. But do not catch bugs with wires.

USART do not touch, just go in and make sure that everything is fine, otherwise everything happens



Fill in the project settings



Save the settings, generate the project, open it in Keil, configure the programmer on autoreset and configure the optimization level **1** .

We will collect the project and continue working with it in the  **next part of the** lesson, in which we begin to write the module initialization function, write the functions of reading and writing registers, and also get acquainted with some commands and registers.

**STM Lesson 103. Modules NRF24L01. Part 3**

Posted on [December 30, 2017](http://narodstream.ru/stm-urok-103-moduli-nrf24l01-chast3/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Published in [Programming STM32](http://narodstream.ru/rub_stm32/)- [No Comments ↓](http://narodstream.ru/stm-urok-103-moduli-nrf24l01-chast3/#respond)

[Programming for beginnersDownload the free book and become a real programmer. Get it now:To learn moresheremetev.info](https://an.yandex.ru/count/8FY8Bn0_jXS50Du1COEzari00000ECgs7402I09Wl0Xe172WmvR10O01vE3CgmU80VY6tP0fa06YakID99W1Zhs-tYEW0VwIp8qag06M_w_U8xW1XgMPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y083e0BUkAe2kGAvp-bFdfA87F02X_FiAVW2We20W82G4e03iftFemg80vc3bFiFc0FAZWIe0nAm0mIu1Fy1w0Ji1_W4pSW1Y0NDo06G1SNU0g05bpoe1O0pi0M0CxW5Zrd01V2P1yW5iEe1q0Nv5k051l050PW6WD2iymwW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000Aa7-56yCBJMn4Ym1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08hvoL0O0A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i6gWiGoSuFfFXH002YgNiMMtK50C0BWAC5o0k0r9C1sGluKRmmjDR4IEWBpSW1y0i6Y0osbTw-0UWC0fWDbOneyWq0-Wq0WWu0YGu00000003mFv0Em8Gzc0x6rypKbOdWkXEW3i24FR0E0Q4F00000000y3-e3_YZiRQkrhNYI000?stat-id=3&test-tag=79714622398465&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY1IjoiNzk3MTQ1OTMwNDY1MjgifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[How did I learn to draw at home?Participate in a series of webinars and get your favorite painting lessons immediately!To learn morenewdirections.ru](https://an.yandex.ru/count/PxkOp-x9xW850CW1COEzari00000ECgs7402I09Wl0Xe173mgPjnW07Cl-VExEgVyII80OF0sv4ra06ojhIyBvW1YgYrjowW0SwtjBmlg07qfhMtBhW1qE_sxXd00GBO0UQClmtW0UgOlWte0Go80WIW0fBxjnMv0hdFwK-UaeWSy0Auvgkx2VW2We20W82G4e03xQhkeGw80zpicVi2i0C4k0J_0UW4mGxu1BfYY0MwOf05lAi2e0N57QW5X1cm1OGPk0MHJC05y9a7o0MmwW7G1Q4Bu0LQy0K1c0Qyiyque0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XHnBxJriH8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2Cdzsmg02W712W0000000F0_s0e2u0g0YNhu2i3y5OWB2AeB4Cq1PltuKG006Os35bjr1G302u2Z1SWBWDIJ0TaB-574ljFMn4Ze2xfYy0i6w0m2c0tylMRo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0E-QGTi0u1eGy00000003mFwWF-AE1-ApQjU98?stat-id=4&test-tag=79714622398465&format-type=24&banner-test-tags=eyI2MzcxNTc3MjQxIjoiNzk3MTQ1OTMwNDY1MjgifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

In the **previous part of the** lesson we studied some more features and features of the module, got acquainted with practical circuits for connecting the receiver and transmitter, prepared the module's power, and also created and configured the project to work with it in the programming environment.

First, we'll add for the global array in the **main.c** file

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

**char str1[20]={0};**

**uint8\_t buf1[20]={0};**

/\* USER CODE END PV \*/

Then create two library files of the module **NRF24.h** and **NRF24.c** with the following content

**NRF24.h** :

**#ifndef NRF24\_H\_**

**#define NRF24\_H\_**

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

**#include "stm32f1xx\_hal.h"**

**#include <string.h>**

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

**#define CS\_GPIO\_PORT GPIOA**

**#define CS\_PIN GPIO\_PIN\_4**

**#define CS\_ON HAL\_GPIO\_WritePin(CS\_GPIO\_PORT, CS\_PIN, GPIO\_PIN\_RESET)**

**#define CS\_OFF HAL\_GPIO\_WritePin(CS\_GPIO\_PORT, CS\_PIN, GPIO\_PIN\_SET)**

**#define CE\_GPIO\_PORT GPIOA**

**#define CE\_PIN GPIO\_PIN\_3**

**#define CE\_RESET HAL\_GPIO\_WritePin(CE\_GPIO\_PORT, CE\_PIN, GPIO\_PIN\_RESET)**

**#define CE\_SET HAL\_GPIO\_WritePin(CE\_GPIO\_PORT, CE\_PIN, GPIO\_PIN\_SET)**

**#define IRQ\_GPIO\_PORT GPIOA**

**#define IRQ\_PIN GPIO\_PIN\_2**

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

**#define LED\_GPIO\_PORT GPIOC**

**#define LED\_PIN GPIO\_PIN\_13**

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

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

**#define LED\_TGL HAL\_GPIO\_TogglePin(LED\_GPIO\_PORT, LED\_PIN)**

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

**#define ACTIVATE 0x50 //**

**#define RD\_RX\_PLOAD 0x61 // Define RX payload register address**

**#define WR\_TX\_PLOAD 0xA0 // Define TX payload register address**

**#define FLUSH\_TX 0xE1**

**#define FLUSH\_RX 0xE2**

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

**#define CONFIG 0x00 //'Config' register address**

**#define EN\_AA 0x01 //'Enable Auto Acknowledgment' register address**

**#define EN\_RXADDR 0x02 //'Enabled RX addresses' register address**

**#define SETUP\_AW 0x03 //'Setup address width' register address**

**#define SETUP\_RETR 0x04 //'Setup Auto. Retrans' register address**

**#define RF\_CH 0x05 //'RF channel' register address**

**#define RF\_SETUP 0x06 //'RF setup' register address**

**#define STATUS 0x07 //'Status' register address**

**#define RX\_ADDR\_P0 0x0A //'RX address pipe0' register address**

**#define RX\_ADDR\_P1 0x0B //'RX address pipe1' 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 FIFO\_STATUS 0x17 //'FIFO Status Register' register address**

**#define DYNPD 0x1C**

**#define FEATURE 0x1D**

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

**#define PRIM\_RX 0x00 //RX/TX control (1: PRX, 0: PTX)**

**#define PWR\_UP 0x01 //1: POWER UP, 0:POWER DOWN**

**#define RX\_DR 0x40 //Data Ready RX FIFO interrupt**

**#define TX\_DS 0x20 //Data Sent TX FIFO interrupt**

**#define MAX\_RT 0x10 //Maximum number of TX retransmits interrupt**

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

**#define W\_REGISTER 0x20 //запись в регистр**

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

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

**#endif /\* NRF24\_H\_ \*/**

**NRF24.c:**

**#include "NRF24.h"**

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

**extern SPI\_HandleTypeDef hspi1;**

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

**\_\_STATIC\_INLINE void DelayMicro(\_\_IO uint32\_t micros)**

**{**

**micros \*= (SystemCoreClock / 1000000) / 9;**

**/\* Wait till done \*/**

**while (micros--) ;**

**}**

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

In order not to waste time, we immediately filled the necessary headers with the header file. As the registers are studied and used, we will use them.

In **main.c we** also connect our library, and at the same time, the library for working with strings

/\* USER CODE BEGIN Includes \*/

**#include "NRF24.h"**

**#include <string.h>**

/\* USER CODE END Includes \*/

Go to file  **NRF24.c** and add the module initialization function

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

**void NRF24\_ini(void)**

**{**

**}**

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

Let's create a prototype in the header file for this function and call it in **main ()**

/\* USER CODE BEGIN 2 \*/

**NRF24\_ini();**

/\* USER CODE END 2 \*/

Let's return to the file and start writing the body of the receiver initialization function

void NRF24\_ini(void)

{

**CE\_RESET;**

**DelayMicro(5000);**

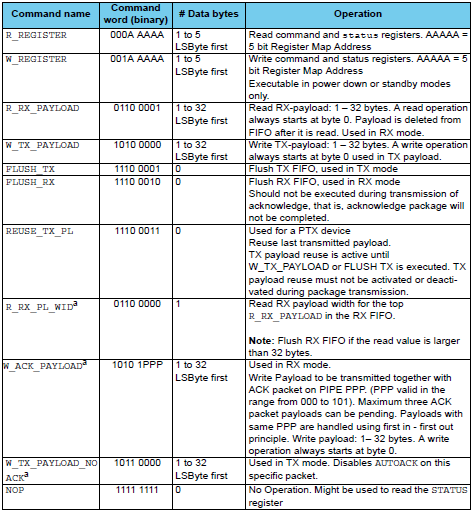
}

First we drop the CE contact and apply a small delay. In timing, it's a little confusing how delay should be, apparently depending on the type of transceiver (active or passive antenna). But the maximum is 4.5 milliseconds. Let's give a little more, since we have the delay function homemade and it gives the delay a little less (checked by a logical analyzer).

Further work on registers is already underway, so you will have to plunge into the theory a little.

As we have already seen from the exchange diagram, first, as a rule, there is a command. Let's get acquainted with some of them.

First let's see the entire list of commands



Now in order. All teams are eight-bit, we'll sort them out a little, and also with bits in them.

**R\_REGISTER** - the command to read the register. The command is recognized by the three most significant bits. If they are in zeros, then this is the given command. The remaining five bits contain the register address.

**W\_REGISTER** - command to write to the register. It is also recognized by the three most significant bits. The sixth must be in the unit, the rest - zeros. The register address is also contained in the lower five bits of the instruction.

**R\_RX\_PAYLOAD** - the command to read the FIFO buffer. After the MISO command, data from the young byte begins to be transmitted. After completing the reading, the FIFO buffer is released.

**W\_TX\_PAYLOAD** - the command to write data to the FIFO buffer. After the MOSI instruction, we transfer the data to the module and write it to the FIFO buffer. Load data should also be from the low-order byte.

**FLUSH\_TX** - the command to clear the FIFO buffer to be transmitted.

**FLUSH\_RX** - command to clear the FIFO buffer, intended for receiving.

**ACTIVATE - the**  command activates the commands  **R\_RX\_PL\_WID, W\_ACK\_PAYLOAD and W\_TX\_PAYLOAD\_NOACK** . In this data, from which the screenshot was made, it is not, there is another.

We do not need the rest of the team. If you need it, we'll get acquainted with them later.

Add the functions of writing and reading registers above the initialization function

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

**uint8\_t NRF24\_ReadReg(uint8\_t addr)**

**{**

**uint8\_t dt=0, cmd;**

**CS\_ON;**

**HAL\_SPI\_TransmitReceive(&hspi1,&addr,&dt,1,1000);**

**if (addr!=STATUS)//если адрес равен адрес регистра статус то и возварщаем его состояние**

**{**

**cmd=0xFF;**

**HAL\_SPI\_TransmitReceive(&hspi1,&cmd,&dt,1,1000);**

**}**

**CS\_OFF;**

**return dt;**

**}**

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

**void NRF24\_WriteReg(uint8\_t addr, uint8\_t dt)**

**{**

**addr |= W\_REGISTER;//включим бит записи в адрес**

**CS\_ON;**

**HAL\_SPI\_Transmit(&hspi1,&addr,1,1000);//отправим адрес в шину**

**HAL\_SPI\_Transmit(&hspi1,&dt,1,1000);//отправим данные в шину**

**CS\_OFF;**

**}**

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

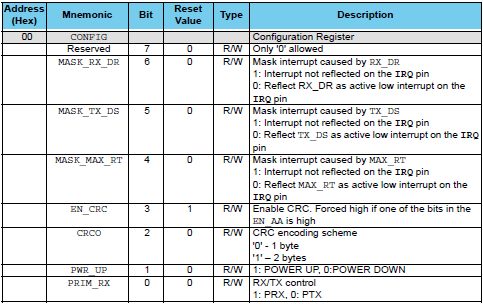
When we register, we first send the register address to the SPI bus. We do not use the command separately, since there are zeros, and since the register addresses do not exceed 0x1F, they will already be there.

If this is the STATUS register, then in this operation already the data will be in the returned value, and if not, then we make one more exchange on the SPI bus and only after that the data will return.

And if we write to the register, then we add the command to the register address, and then we send this address alternately along with the command, and then we send the data.

Now the registers themselves. I think they also should not study everything. And to study them better one by one as you use, as we did with MEMS sensors.

The first register is **CONFIG** - the configuration register



The name of the register speaks for itself. This register is intended for configuring the basic parameters of the transceiver.

7 bit is not used.

The following bits are interrupt mask bits. If the bit is at zero, then the interrupt, for which this bit is responsible, will not be used. There are three types of interruptions, characterizing certain events, in the event of an incursion, the IRQ leg will be attracted to the ground. Now, bit by bit - which bit for what interrupt answers

**MASK\_RX\_DR** is an interrupt that occurs when a packet is received in the receiver at the moment the packet appears in the receiving buffer.

**MASK\_TX\_DS** is an interrupt that occurs when the packet is successfully sent to the receiver in the transmitting receiver. If auto-confirmation is enabled, this interrupt occurs after receiving confirmation from the receiver.

**MASK\_MAX\_RT** is the interrupt occurring when the maximum number of repeated transmissions of the packet is exhausted by the transmitter. The maximum number is also set in a certain register.

**EN\_CRC** - enabling the use of a checksum. If at least one channel is auto-asserted, this bit will turn on itself.

**CRCO**  - the number of bytes of the checksum. 0 - 1 byte, 1 - 2 bytes.

**PWR\_UP** is the transmit control enable bit. 1 - on, 0 - off (or hibernate).

Continuing the initialization function, we will enter the bit values ​​in the configuration register

DelayMicro(5000);

**NRF24\_WriteReg(CONFIG, 0x0a); // Set PWR\_UP bit, enable CRC(1 byte) &Prim\_RX:0 (Transmitter)**

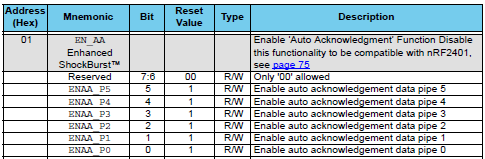
We included the module in the transmitter mode, the checksum (in the size of 1 byte).

We'll wait another 5 milliseconds for the transmitter to turn on. Here, just need a delay. Well, let it be before turning on and after. Not prevent.

NRF24\_WriteReg(CONFIG, 0x0a); // Set PWR\_UP bit, enable CRC(1 byte) &Prim\_RX:0 (Transmitter)

**DelayMicro(5000);//Дадим модулю включиться, по даташиту около 1,5 мсек, а лучше 5**

The next register is **EN\_AA** ( **Enhanced ShockBurst ™** ). This is a register that uses the specified technology and includes auto-confirmation for a certain exchange channel (not to be confused with frequency channels, which are many)

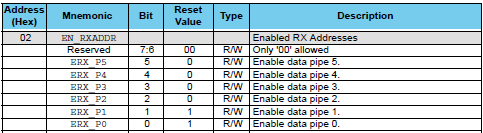


Each bit is responsible for enabling this mode of a certain channel, which can be up to six. We will use 1 channel (not zero). In later studies we will understand why not a zero

DelayMicro(5000);//Дадим модулю включиться, по даташиту около 1,5 мсек, а лучше 5

**NRF24\_WriteReg(EN\_AA, 0x02); // Enable Pipe1**

The next register is **EN\_RXADDR** , which just includes the use of channels. And these channels will differ at the addresses by which the receivers will understand that this packet is addressed to him

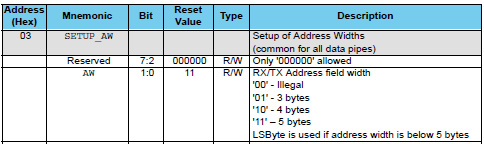


Enter the values ​​in this register, thereby showing the module that we will use exactly 1 exchange channel

NRF24\_WriteReg(EN\_AA, 0x02); // Enable Pipe1

**NRF24\_WriteReg(EN\_RXADDR, 0x02); // Enable Pipe1**

The next register is **SETUP\_AW** ( **Setup of Address Widths** ), which is common for all exchange channels and sets the receiver and transmitter addresses (can be said to be network addresses) in bytes, which can be adjusted from 3 to 5 bytes



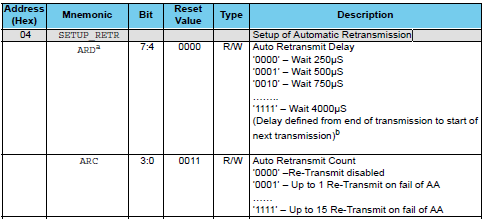
The older six bits carry no information, and the two younger ones set the size.

Set the address width to 3 bytes

NRF24\_WriteReg(EN\_RXADDR, 0x01); // Enable Pipe1

**NRF24\_WriteReg(SETUP\_AW, 0x01); // Setup address width=3 bytes**

The next register is **SETUP\_RETR** ( **Setup of Automatic Retransmission** ) - 'this is the register that sets the parameters for retransmissions of the packet upon their unsuccessful sending



The older tetrad contains information about the amount of delay between sending the same packet for failed delivery, and the younger one for the maximum number of such shipments.

Initialize this register in your code

NRF24\_WriteReg(SETUP\_AW, 0x01); // Setup address width=3 bytes

**NRF24\_WriteReg(SETUP\_RETR, 0x5F); // // 1500us, 15 retrans**

We will use a delay of 1500 microseconds and the maximum number of attempts is 15.

There is one more command - **ACTIVATE** , which I did not immediately find in the documentation, but it is used in the library for Arduino, and I decided to use it too. Superfluous will not. Anyway, we initialize the module only once when you turn it on or when you reboot. Then I found the command after all. It turns out that the documentation is different. This command activates the commands  **R\_RX\_PL\_WID, W\_ACK\_PAYLOAD and W\_TX\_PAYLOAD\_NOACK** . To use the command, add one more function above the initialization function

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

**void NRF24\_ToggleFeatures(void)**

**{**

**uint8\_t dt[1] = {ACTIVATE};**

**CS\_ON;**

**HAL\_SPI\_Transmit(&hspi1,dt,1,1000);**

**DelayMicro(1);**

**dt[0] = 0x73;**

**HAL\_SPI\_Transmit(&hspi1,dt,1,1000);**

**CS\_OFF;**

**}**

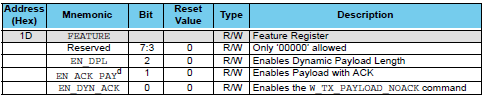
**//-----------------------------------------------**

Call this function in the receiver initialization function

NRF24\_WriteReg(SETUP\_RETR, 0x5F); // // 1500us, 15 retrans

**NRF24\_ToggleFeatures();**

Next, the register we will look at is the **FEATURE** register , which is used in the dynamic number of bytes in the packet and contains some settings for this mode



The older five bits are not used. The **EN\_DPL** bit enables the dynamic number of bytes in the packet, the **EN\_ACK\_PAY** bit enables the sending of packets with pending acknowledgment, and the **EN\_DYN\_ACK**bit without acknowledgment, in which case the transmitter sends a special command to the receiver to understand that it is not necessary to confirm the packet.

We will not use this mode, we will use the fixed-byte transfer mode, so we do not enable this mode

NRF24\_ToggleFeatures();

**NRF24\_WriteReg(FEATURE, 0);**

In the **next part of the** lesson, we will end the introduction to registers by modules, finish writing the initialization code and check the values ​​of the registers, counting them and displaying them in the terminal program.

**STM Lesson 103. Modules NRF24L01. Part 4**

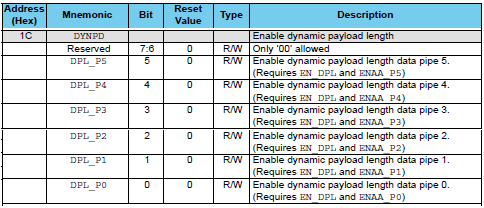
Posted on [December 30, 2017](http://narodstream.ru/stm-urok-103-moduli-nrf24l01-chast4/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Published in [Programming STM32](http://narodstream.ru/rub_stm32/)- [5 comments ↓](http://narodstream.ru/stm-urok-103-moduli-nrf24l01-chast4/#comments)

[How did I learn to draw at home?Participate in a series of webinars and get your favorite painting lessons immediately!To learn morenewdirections.ru](https://an.yandex.ru/count/Lavjgss4HrK50Cu1CP6zari00000ECgs7402I09Wl0Xe173mgPjnW07Cl-VExEgVyII80OF0sv4ra06ojhIyBvW1YgYrjowW0SwtjBmlg07qfhMtBhW1qE_sxXd00GBO0UQClmtW0UgOlWte0Go80WEW0fBxjnMv0khkaiG-n8Xmy0B-mAEC3FW2We20W82W4e03xQhkeGw80zpicVi2g0CIi0C4k0J_0UW4m0tu19d10OW5cS41a0MEkmAW1Vivg0MFCR05Zp6u1OvGm0NmcGV81R3g0T05ZHRW1Lhm1G6O1hoppJYW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000Aa7-54yX0dQn4Ym1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08oVtR2e0A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i6gWiGTgDBPlXH0000_b0NMtK50C0BWAC5o0k0r9C1sGluKJo42Th4IEWBcS41y0i6w0m2c0sLZ6Zo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0E-QGTi0u1eGy00000003mFwWF-AFHqv31jk98?stat-id=3&test-tag=80264378212353&format-type=24&banner-test-tags=eyI2MzcxNTc3MjQxIjoiODAyNjQzNDg4NjA0MTYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Programming for beginnersDownload the free book and become a real programmer. Get it now:To learn moresheremetev.info](https://an.yandex.ru/count/S9XCfwKdTCm50Dm1CPAzari00000ECgs7402I09Wl0Xe172WmvR10O01vE3CgmU80VY6tP0fa06YakID99W1Zhs-tYEW0VwIp8qag06M_w_U8xW1XgMPz1N00GBO0PQmhH3W0UQKqmxe0HRu0RQLthu1Y084e0BUkAe2kGBgxfB4FiI8SF02nu-8nWpu0eA0W820e1A00xATpwCAY0EPWvJx3vW3oeu4i0C4k0J_0UW4x0Zu1FY20OW5-881a0NotWAW1QGdg0Mb8R05fI6u1PfPm0NmcGV81R3g0T05_GxW1GRm1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XHSAHpsiH8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW2A-SbG602W712W0000000F0_s0e2u0g0YNhu2i3y5OWB2AeB48uw9F3uKG00aePh5rjr1G302u2Z1SWBWDIJ0TaB-55mf7FQn4Ze2_Y20V0B1eWCjfNUlW7e30AO3VozPl8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_ue_6WiiwsuaW0?stat-id=4&test-tag=80264378212353&format-type=24&banner-test-tags=eyI0ODQ2MjkwMDY1IjoiODAyNjQzNDg4NjA0MTYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

In the **previous part of the** lesson we started writing the module initialization function, wrote the functions of reading and writing registers, and also got acquainted with some commands and registers.

The next register is **DYNPD** , which controls the use of the dynamic byte in the packet mode for each exchange channel

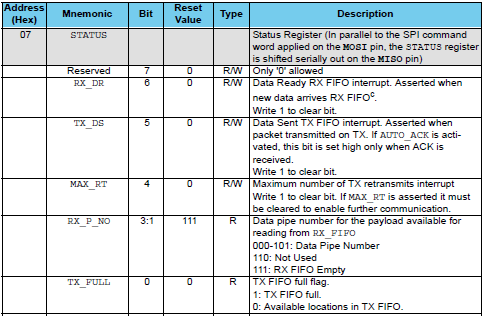


Since we do not use this mode, we will not include any bits either

NRF24\_WriteReg(FEATURE, 0);

**NRF24\_WriteReg(DYNPD, 0);**

The next interesting register is **STATUS** , which is usually used to find out a particular state, but sometimes it will also need to be used for writing



The oldest bit is not used. Then there are three bits, the state of which depends on the interrupt that occurred. The type of interrupts fully corresponds to the types of interrupts considered by us above and located in the configuration register. When a low level on the IRQ foot appears, we examine these bits, and in order to reset the interrupt flag, we must put a one in the corresponding bit. That is, the process of entering the bit manually is important, although it seems to be there already.

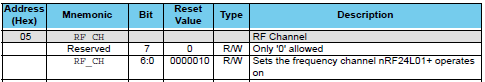
The next three bits carry the information about the channel number of the data exchange, to which the information entered is located in RX\_FIFO. These are very necessary bits, that's why we will learn from which transmitter the packet came to us, which we are currently reading from the buffer. The next bit  **TX\_FULL is**  set if the FIFO buffer is full for transmission.

During initialization, you must reset the flags of all interrupts

NRF24\_WriteReg(DYNPD, 0);

**NRF24\_WriteReg(STATUS, 0x70); //Reset flags for IRQ**

The next register is **RF\_CH** , which sets the frequency of the transmitter operation. We transmit data on one of the frequency channels (not to be confused with data exchange channels), which we set here



We fill the 6: 0 bits with a value that carries information in itself, how many MHz should be added to the frequency of 2400 to get the desired channel. In total there can be 126 channels - from 2 to 127. For example, if we enter the number 5 in these bits, we get a frequency of 2405 megahertz.

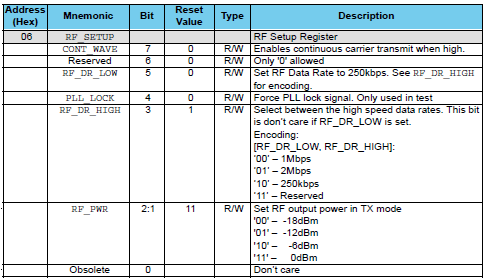
Turn on the desired frequency channel

NRF24\_WriteReg(STATUS, 0x70 ); //Reset flags for IRQ

**NRF24\_WriteReg(RF\_CH, 76); // частота 2476 MHz**

We must include the same channel on all receivers and transmitters, otherwise they will not be able to communicate with each other at different frequencies.

The next register is **RF\_SETUP** , which is used to adjust the speed and transmit power



The most senior bit - **CONT\_WAVE** includes a constant carrier transfer, can be useful for tuning and reception-transmission tests. We do not use this bit.

The next bit is skipped, not used at all.

The bits  **RF\_DR\_LOW** and **RF\_DR\_HIGH** are used to adjust the transmission rate. The speed of 250 kbps is supported only by modules with a " **+** " sign .

The **PLL\_LOCK** bit includes a transmit-receive lock, is also used for testing.

The **RF\_PWR** bits **are** responsible for setting the transmit power. Can be useful in use, for example, in a quat, if there is a possibility of interference to neighbors.

Let us configure this register in the module initialization function

NRF24\_WriteReg(RF\_CH, 76); // частота 2476 MHz

**NRF24\_WriteReg(RF\_SETUP, 0x06); //TX\_PWR:0dBm, Datarate:1Mbps**

Now let's learn how to write and read the buffer. To do this, add two functions above the initialization function

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

**void NRF24\_Read\_Buf(uint8\_t addr,uint8\_t \*pBuf,uint8\_t bytes)**

**{**

**CS\_ON;**

**HAL\_SPI\_Transmit(&hspi1,&addr,1,1000);//отправим адрес в шину**

**HAL\_SPI\_Receive(&hspi1,pBuf,bytes,1000);//отправим данные в буфер**

**CS\_OFF;**

**}**

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

**void NRF24\_Write\_Buf(uint8\_t addr,uint8\_t \*pBuf,uint8\_t bytes)**

**{**

**addr |= W\_REGISTER;//включим бит записи в адрес**

**CS\_ON;**

**HAL\_SPI\_Transmit(&hspi1,&addr,1,1000);//отправим адрес в шину**

**DelayMicro(1);**

**HAL\_SPI\_Transmit(&hspi1,pBuf,bytes,1000);//отправим данные в буфер**

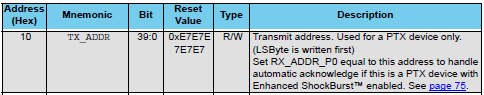
**CS\_OFF;**

**}**

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

The writing and reading of the buffer occurs in a manner similar to writing and reading registers with the only difference being that we transmit or receive not one but several bytes after the transfer of the address.

The next register is **TX\_ADDR** , which carries information about the address of the transmitter



The address in this register is entered as follows, we transfer to the SPI bus first the register address along with the write command, and then the address bytes in turn, than the buffer write function itself does. Add a global array with an address and some de-nails at the beginning of the current file

extern SPI\_HandleTypeDef hspi1;

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

**#define TX\_ADR\_WIDTH 3**

**#define TX\_PLOAD\_WIDTH 2**

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

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

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

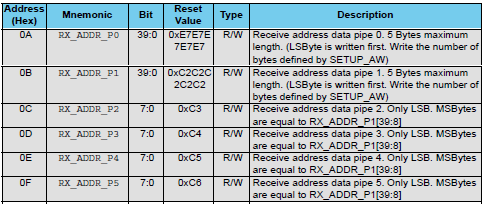
We defrained the width of the address, as well as the size of the packet (until there are 2 bytes), an array with an address was also added and the buffer for the packets was initialized.

Now put this address in the memory of the receiver in the function of its initialization, using the above-mentioned register

NRF24\_WriteReg(RF\_SETUP, 0x06); //TX\_PWR:0dBm, Datarate:1Mbps

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

The following registers are **RX\_ADDR\_P0: RX\_ADDR\_P5** . These are the address registers of the information channels for receiving devices



Here we enter the various addresses of the transmitters in order to distinguish which channel the data came from. The addresses must be unique, and for the Pipe2-Pipe5 channels, we register only the low-order byte of the register, and the older ones are taken from the pipe1 pipe address.

We'll add the address to pipe1 in case we need to use the module as a receiver. We will enter the same address as the address register of the transmitter

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

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

The following registers are **RX\_PW\_P0: RX\_PW\_P5** . In these registers, the number of bytes in the packet for each information exchange channel is recorded. These registers take up a lot of space in the screenshot, so I will not post it, there's nothing special about it. In each of these registers only 6 low-order bits, 2 senior ones are not used. We will use it until the packet size is 2 bytes

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

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

We also add the function of clearing the FIFO buffers of reception and transmission above the initialization function

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

**void NRF24\_FlushRX(void)**

**{**

**uint8\_t dt[1] = {FLUSH\_RX};**

**CS\_ON;**

**HAL\_SPI\_Transmit(&hspi1,dt,1,1000);**

**DelayMicro(1);**

**CS\_OFF;**

**}**

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

**void NRF24\_FlushTX(void)**

**{**

**uint8\_t dt[1] = {FLUSH\_TX};**

**CS\_ON;**

**HAL\_SPI\_Transmit(&hspi1,dt,1,1000);**

**DelayMicro(1);**

**CS\_OFF;**

**}**

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

I think that everything is simple in these functions, we just pass the corresponding commands to the module, and then wait a little for the cleaning process to finish.

We will first initialize each module as a receiver and only then, when we need to send the packet, we will turn on the transmission mode. In most sources, which I looked at, it's done just like that.

Therefore, we add a function to enable the receive mode over the initialization function

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

**void NRF24L01\_RX\_Mode(void)**

**{**

**uint8\_t regval=0x00;**

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

**//разбудим модуль и переведём его в режим приёмника, включив биты PWR\_UP и PRIM\_RX**

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

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

**CE\_SET;**

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

**// Flush buffers**

**NRF24\_FlushRX();**

**NRF24\_FlushTX();**

**}**

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

In this function, we read the configuration register and initialize the bits in it to turn on the receiver and to turn on the receive mode, then register them in the configuration register, then raise the CE foot, thereby enabling the reception mode already physically, wait at least 130 microseconds, Documentation and clean buffers.

Call this function in the initialization function of the module and turn off the LED on the board, which will indicate that the initialization was completed, that is, nothing hanging anywhere

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

**//пока уходим в режим приёмника**

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

**LED\_OFF;**

}

On the function of reading the register and reading the buffer, we will create prototypes in the header file **NRF24.h**

void NRF24\_ini(void);

**uint8\_t NRF24\_ReadReg(uint8\_t addr);**

**void NRF24\_Read\_Buf(uint8\_t addr,uint8\_t \*pBuf,uint8\_t bytes);**

Then go to the **main.c** file in the **main ()** function and add the local variable

/\* USER CODE BEGIN 1 \*/

**uint8\_t dt\_reg=0;**

/\* USER CODE END 1 \*/

In an infinite loop, we read a few registers and an address from the buffer, thereby verifying that the exchange with the module on the SPI bus is normal for us

/\* USER CODE BEGIN 3 \*/

**HAL\_Delay(1000);**

**dt\_reg = NRF24\_ReadReg(CONFIG);**

**sprintf(str1,"CONFIG: 0x%02Xrn",dt\_reg);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**dt\_reg = NRF24\_ReadReg(EN\_AA);**

**sprintf(str1,"EN\_AA: 0x%02Xrn",dt\_reg);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**dt\_reg = NRF24\_ReadReg(EN\_RXADDR);**

**sprintf(str1,"EN\_RXADDR: 0x%02Xrn",dt\_reg);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**dt\_reg = NRF24\_ReadReg(STATUS);**

**sprintf(str1,"STATUS: 0x%02Xrn",dt\_reg);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**dt\_reg = NRF24\_ReadReg(RF\_SETUP);**

**sprintf(str1,"RF\_SETUP: 0x%02Xrn",dt\_reg);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**NRF24\_Read\_Buf(TX\_ADDR,buf1,3);**

**sprintf(str1,"TX\_ADDR: 0x%02X, 0x%02X, 0x%02Xrn",buf1[0],buf1[1],buf1[2]);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

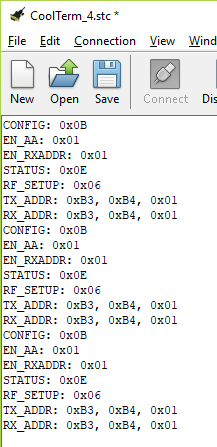
**NRF24\_Read\_Buf(RX\_ADDR\_P1,buf1,3);**

**sprintf(str1,"RX\_ADDR: 0x%02X, 0x%02X, 0x%02Xrn",buf1[0],buf1[1],buf1[2]);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

}

We will collect the code, we will sew the controller and see the result in the terminal program



As you can see, everything is working for us.

Thank you all for your attention!

Wait for the next lessons on this module, in which we will already try to exchange data over a radio channel between two such receivers.