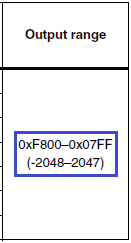
**Lesson 49. HAL. Magnetometer LSM303DLHC**

**Part 1**

Today we will continue the topic of sensors using MEMS technology and return to the sensor, which we have already partially considered 10 lessons back - **LSM303DLHC**.

I repeat that this sensor is installed on the STM32F3Discovery board. And we will study today its second part - the magnetometer. A magnetometer is a sensor that is able to react and measure a value such as magnetic induction, which is measured in Gauss (GHS system). Also in the official SI table, the magnetic induction is measured in Tesla. 1 Gauss is equal to 100 μT or 1 Tesla is equal to 10,000 Gauss.

The technical characteristics of this are described in the datasheet on it, a reference to which I will repeat at the end of the article. The main characteristics that we will need are those that it is also 16-bit, although not all bits are used for the readings-only 12 bits and its measurements it interprets in registers in the range from -2048 to +2047

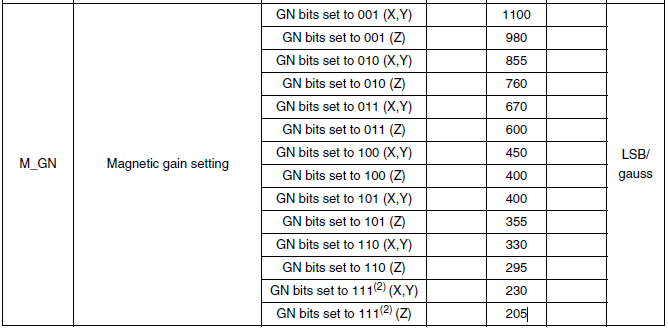


Also we need the following characteristics:

interface for taking readings - I2C;

The reading range is ± 1.3 / ± 1.9 / ± 2.5 / ± 4.0 / ± 4.7 / ± 5.6 / ± 8.1 gauss;

The discreteness of 1 Gauss or the number of gradations in 1 Gauss depends on the set range of readings, as well as on the axes, and is in the range from 205 to 1100

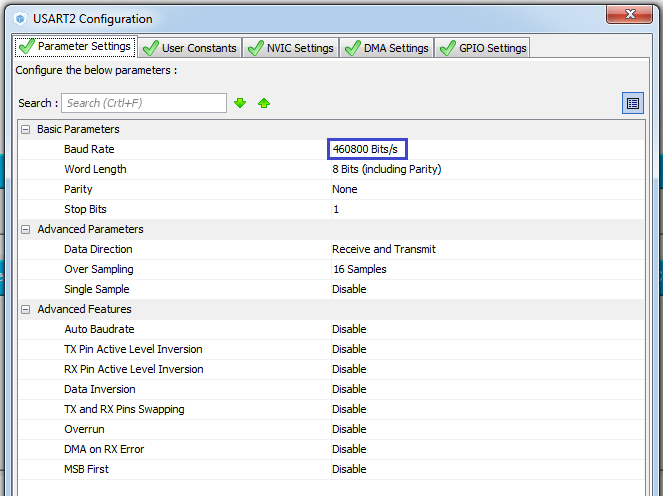


With other characteristics, subtleties, magnetometer, we will get acquainted in the course of its programming.

The project we will create from the previous project for this sensors - from the Accel project and call it Mag

Run the project in the Cube MX.

Let's change the following - add the speed of the USART2 bus



Generate the code, open the project, configure the programmer to auto-cut, as usual. We connect the file lsm303dlhc.c. We will compile the project.

In the file main.c in an infinite loop, while commenting out the function call

// Accel\_ReadAcc ();

In the file lsm303dlhc.h, delete everything completely and insert the ready prepared code with new macros for the magnetometer and with the corrected prototypes of functions

#ifndef LSM303DLHC\_H\_

#define LSM303DLHC\_H\_

#include "stm32f3xx\_hal.h"

#include <string.h>

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

#define ABS (x) (x <0)? (-x): x

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

#define LD\_PORT GPIOE

#define LD3 GPIO\_PIN\_9 // RED1

#define LD4 GPIO\_PIN\_8 // BLUE1

#define LD5 GPIO\_PIN\_10 // ORANGE1

#define LD6 GPIO\_PIN\_15 // GREEN1

#define LD7 GPIO\_PIN\_11 // GREEN2

#define LD8 GPIO\_PIN\_14 // ORANGE2

#define LD9 GPIO\_PIN\_12 // BLUE2

#define LD10 GPIO\_PIN\_13 // RED2

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

#define ACC\_I2C\_ADDRESS 0x32

#define MAG\_I2C\_ADDRESS 0x3C

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

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

#define LSM303DLHC\_CTRL\_REG1\_A 0x20 / \* Control register 1 acceleration \* /

#define LSM303DLHC\_CTRL\_REG2\_A 0x21 / \* Control register 2 acceleration \* /

#define LSM303DLHC\_CTRL\_REG3\_A 0x22 / \* Control register 3 acceleration \* /

#define LSM303DLHC\_CTRL\_REG4\_A 0x23 / \* Control register 4 acceleration \* /

#define LSM303DLHC\_CTRL\_REG5\_A 0x24 / \* Control register 5 acceleration \* /

#define LSM303DLHC\_CRA\_REG\_M 0x00 / \* Control register A magnetic field \* /

#define LSM303DLHC\_CRB\_REG\_M 0x01 / \* Control register B magnetic field \* /

#define LSM303DLHC\_MR\_REG\_M 0x02 / \* Control register MR magnetic field \* /

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

#define LD3\_ON HAL\_GPIO\_WritePin (LD\_PORT, LD3, GPIO\_PIN\_SET) // RED1

#define LD4\_ON HAL\_GPIO\_WritePin (LD\_PORT, LD4, GPIO\_PIN\_SET) // BLUE1

#define LD5\_ON HAL\_GPIO\_WritePin (LD\_PORT, LD5, GPIO\_PIN\_SET) // ORANGE1

#define LD6\_ON HAL\_GPIO\_WritePin (LD\_PORT, LD6, GPIO\_PIN\_SET) // GREEN1

#define LD7\_ON HAL\_GPIO\_WritePin (LD\_PORT, LD7, GPIO\_PIN\_SET) // GREEN2

#define LD8\_ON HAL\_GPIO\_WritePin (LD\_PORT, LD8, GPIO\_PIN\_SET) // ORANGE2

#define LD9\_ON HAL\_GPIO\_WritePin (LD\_PORT, LD9, GPIO\_PIN\_SET) // BLUE2

#define LD10\_ON HAL\_GPIO\_WritePin (LD\_PORT, LD10, GPIO\_PIN\_SET) // RED2

#define LD3\_OFF HAL\_GPIO\_WritePin (LD\_PORT, LD3, GPIO\_PIN\_RESET) // RED1

#define LD4\_OFF HAL\_GPIO\_WritePin (LD\_PORT, LD4, GPIO\_PIN\_RESET) // BLUE1

#define LD5\_OFF HAL\_GPIO\_WritePin (LD\_PORT, LD5, GPIO\_PIN\_RESET) // ORANGE1

#define LD6\_OFF HAL\_GPIO\_WritePin (LD\_PORT, LD6, GPIO\_PIN\_RESET) // GREEN1

#define LD7\_OFF HAL\_GPIO\_WritePin (LD\_PORT, LD7, GPIO\_PIN\_RESET) // GREEN2

#define LD8\_OFF HAL\_GPIO\_WritePin (LD\_PORT, LD8, GPIO\_PIN\_RESET) // ORANGE2

#define LD9\_OFF HAL\_GPIO\_WritePin (LD\_PORT, LD9, GPIO\_PIN\_RESET) // BLUE2

#define LD10\_OFF HAL\_GPIO\_WritePin (LD\_PORT, LD10, GPIO\_PIN\_RESET) // RED2

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

#define LSM303DLHC\_NORMAL\_MODE ((uint8\_t) 0x00)

#define LSM303DLHC\_LOWPOWER\_MODE ((uint8\_t) 0x08)

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

#define LSM303DLHC\_ODR\_1\_HZ ((uint8\_t) 0x10) / \*! <Output Data Rate = 1 Hz \* /

#define LSM303DLHC\_ODR\_10\_HZ ((uint8\_t) 0x20) / \*! <Output Data Rate = 10 Hz \* /

#define LSM303DLHC\_ODR\_25\_HZ ((uint8\_t) 0x30) / \*! <Output Data Rate = 25 Hz \* /

#define LSM303DLHC\_ODR\_50\_HZ ((uint8\_t) 0x40) / \*! <Output Data Rate = 50 Hz \* /

#define LSM303DLHC\_ODR\_100\_HZ ((uint8\_t) 0x50) / \*! <Output Data Rate = 100 Hz \* /

#define LSM303DLHC\_ODR\_200\_HZ ((uint8\_t) 0x60) / \*! <Output Data Rate = 200 Hz \* /

#define LSM303DLHC\_ODR\_400\_HZ ((uint8\_t) 0x70) / \*! <Output Data Rate = 400 Hz \* /

#define LSM303DLHC\_ODR\_1620\_HZ\_LP ((uint8\_t) 0x80) / \*! <Output Data Rate = 1620 Hz only in Low Power Mode \* /

#define LSM303DLHC\_ODR\_1344\_HZ ((uint8\_t) 0x90) / \*! <Output Data Rate = 1344 Hz in Normal mode and 5376 Hz \* /

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

#define LSM303DLHC\_X\_ENABLE ((uint8\_t) 0x01)

#define LSM303DLHC\_Y\_ENABLE ((uint8\_t) 0x02)

#define LSM303DLHC\_Z\_ENABLE ((uint8\_t) 0x04)

#define LSM303DLHC\_AXES\_ENABLE ((uint8\_t) 0x07)

#define LSM303DLHC\_AXES\_DISABLE ((uint8\_t) 0x00)

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

#define LSM303DLHC\_HR\_ENABLE ((uint8\_t) 0x08)

#define LSM303DLHC\_HR\_DISABLE ((uint8\_t) 0x00)

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

#define LSM303DLHC\_FULLSCALE\_2G ((uint8\_t) 0x00) / \*! <± 2 g \* /

#define LSM303DLHC\_FULLSCALE\_4G ((uint8\_t) 0x10) / \*! <± 4 g \* /

#define LSM303DLHC\_FULLSCALE\_8G ((uint8\_t) 0x20) / \*! <± 8 g \* /

#define LSM303DLHC\_FULLSCALE\_16G ((uint8\_t) 0x30) / \*! <± 16 g \* /

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

#define LSM303DLHC\_BlockUpdate\_Continous ((uint8\_t) 0x00) / \*! <Continuos Update \* /

#define LSM303DLHC\_BlockUpdate\_Single ((uint8\_t) 0x80) / \*! <Single Update: output registers not updated until MSB and LSB reading \* /

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

#define LSM303DLHC\_BLE\_LSB ((uint8\_t) 0x00) / \*! <Little Endian: data LSB @ lower address \* /

#define LSM303DLHC\_BLE\_MSB ((uint8\_t) 0x40) / \*! <Big Endian: data MSB @ lower address \* /

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

#define LSM303DLHC\_HPM\_NORMAL\_MODE\_RES ((uint8\_t) 0x00)

#define LSM303DLHC\_HPM\_REF\_SIGNAL ((uint8\_t) 0x40)

#define LSM303DLHC\_HPM\_NORMAL\_MODE ((uint8\_t) 0x80)

#define LSM303DLHC\_HPM\_AUTORESET\_INT ((uint8\_t) 0xC0)

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

#define LSM303DLHC\_HPFCF\_8 ((uint8\_t) 0x00)

#define LSM303DLHC\_HPFCF\_16 ((uint8\_t) 0x10)

#define LSM303DLHC\_HPFCF\_32 ((uint8\_t) 0x20)

#define LSM303DLHC\_HPFCF\_64 ((uint8\_t) 0x30)

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

#define LSM303DLHC\_HPF\_AOI1\_DISABLE ((uint8\_t) 0x00)

#define LSM303DLHC\_HPF\_AOI1\_ENABLE ((uint8\_t) 0x01)

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

#define LSM303DLHC\_HPF\_AOI2\_DISABLE ((uint8\_t) 0x00)

#define LSM303DLHC\_HPF\_AOI2\_ENABLE ((uint8\_t) 0x02)

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

#define LSM303DLHC\_ACC\_SENSITIVITY\_2G ((uint8\_t) 1) / \*! <accelerometer sensitivity with 2 g full scale [mg / LSB] \* /

#define LSM303DLHC\_ACC\_SENSITIVITY\_4G ((uint8\_t) 2) / \*! <accelerometer sensitivity with 4 g full scale [mg / LSB] \* /

#define LSM303DLHC\_ACC\_SENSITIVITY\_8G ((uint8\_t) 4) / \*! <accelerometer sensitivity with 8 g full scale [mg / LSB] \* /

#define LSM303DLHC\_ACC\_SENSITIVITY\_16G ((uint8\_t) 12) / \*! <accelerometer sensitivity with 12 g full scale [mg / LSB] \* /

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

#define LSM303DLHC\_OUT\_X\_L\_A 0x28 / \* Output Register X acceleration \* /

#define LSM303DLHC\_OUT\_X\_H\_A 0x29 / \* Output Register X acceleration \* /

#define LSM303DLHC\_OUT\_Y\_L\_A 0x2A / \* Output Register Y acceleration \* /

#define LSM303DLHC\_OUT\_Y\_H\_A 0x2B / \* Output Register Y acceleration \* /

#define LSM303DLHC\_OUT\_Z\_L\_A 0x2C / \* Output Register Z acceleration \* /

#define LSM303DLHC\_OUT\_Z\_H\_A 0x2D / \* Output Register Z acceleration \* /

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

#define LSM303DLHC\_OUT\_X\_H\_M 0x03 / \* Output Register X magnetic field \* /

#define LSM303DLHC\_OUT\_X\_L\_M 0x04 / \* Output Register X magnetic field \* /

#define LSM303DLHC\_OUT\_Z\_H\_M 0x05 / \* Output Register Z magnetic field \* /

#define LSM303DLHC\_OUT\_Z\_L\_M 0x06 / \* Output Register Z magnetic field \* /

#define LSM303DLHC\_OUT\_Y\_H\_M 0x07 / \* Output Register Y magnetic field \* /

#define LSM303DLHC\_OUT\_Y\_L\_M 0x08 / \* Output Register Y magnetic field \* /

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

#define LSM303DLHC\_TEMPSENSOR\_ENABLE ((uint8\_t) 0x80) / \*! <Temp sensor Enable \* /

#define LSM303DLHC\_TEMPSENSOR\_DISABLE ((uint8\_t) 0x00) / \*! <Temp sensor Disable \* /

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

#define LSM303DLHC\_ODR\_0\_75\_HZ ((uint8\_t) 0x00) / \*! <Output Data Rate = 0.75 Hz \* /

#define LSM303DLHC\_ODR\_1\_5\_HZ ((uint8\_t) 0x04) / \*! <Output Data Rate = 1.5 Hz \* /

#define LSM303DLHC\_ODR\_3\_0\_HZ ((uint8\_t) 0x08) / \*! <Output Data Rate = 3 Hz \* /

#define LSM303DLHC\_ODR\_7\_5\_HZ ((uint8\_t) 0x0C) / \*! <Output Data Rate = 7.5 Hz \* /

#define LSM303DLHC\_ODR\_15\_HZ ((uint8\_t) 0x10) / \*! <Output Data Rate = 15 Hz \* /

#define LSM303DLHC\_ODR\_30\_HZ ((uint8\_t) 0x14) / \*! <Output Data Rate = 30 Hz \* /

#define LSM303DLHC\_ODR\_75\_HZ ((uint8\_t) 0x18) / \*! <Output Data Rate = 75 Hz \* /

#define LSM303DLHC\_ODR\_220\_HZ ((uint8\_t) 0x1C) / \*! <Output Data Rate = 220 Hz \* /

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

#define LSM303DLHC\_FS\_1\_3\_GA ((uint8\_t) 0x20) / \*! <Full scale = ± 1.3 Gauss \* /

#define LSM303DLHC\_FS\_1\_9\_GA ((uint8\_t) 0x40) / \*! <Full scale = ± 1.9 Gauss \* /

#define LSM303DLHC\_FS\_2\_5\_GA ((uint8\_t) 0x60) / \*! <Full scale = ± 2.5 Gauss \* /

#define LSM303DLHC\_FS\_4\_0\_GA ((uint8\_t) 0x80) / \*! <Full scale = ± 4.0 Gauss \* /

#define LSM303DLHC\_FS\_4\_7\_GA ((uint8\_t) 0xA0) / \*! <Full scale = ± 4.7 Gauss \* /

#define LSM303DLHC\_FS\_5\_6\_GA ((uint8\_t) 0xC0) / \*! <Full scale = ± 5.6 Gauss \* /

#define LSM303DLHC\_FS\_8\_1\_GA ((uint8\_t) 0xE0) / \*! <Full scale = ± 8.1 Gauss \* /

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

#define LSM303DLHC\_CONTINUOS\_CONVERSION ((uint8\_t) 0x00) / \*! <Continuous-Conversion Mode \* /

#define LSM303DLHC\_SINGLE\_CONVERSION ((uint8\_t) 0x01) / \*! <Single-Conversion Mode \* /

#define LSM303DLHC\_SLEEP ((uint8\_t) 0x02) / \*! <Sleep Mode \* /

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

void AccelMag\_Ini (void);

void AccelMag\_Read (void);

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

#endif / \* LSM303DLHC\_H\_ \* /

We correct the name of the initialization function of the sensor to a more universal one also in the files main.c and lsm303dlhc.c

  / \* USER CODE BEGIN 2 \* /

        Accel **Mag**\_Ini ();

  / \* USER CODE END 2 \* /

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

void Accel **Mag**\_Ini (void)

{

Comment in the implementation code of this function of the line associated with the initialization of the accelerometer, in addition to reading the identifier, because this option is universal for the sensor

        else Error ();

// ctrl | = (LSM303DLHC\_NORMAL\_MODE | LSM303DLHC\_ODR\_50\_HZ | LSM303DLHC\_AXES\_ENABLE);

// ctrl | = ((LSM303DLHC\_BlockUpdate\_Continous | LSM303DLHC\_BLE\_LSB | LSM303DLHC\_HR\_ENABLE) << 8);

// AccInit (ctrl);

// ctrl = (uint8\_t) (LSM303DLHC\_HPM\_NORMAL\_MODE | LSM303DLHC\_HPFCF\_16 | \

// LSM303DLHC\_HPF\_AOI1\_DISABLE | LSM303DLHC\_HPF\_AOI2\_DISABLE);

// Accel\_AccFilter (ctrl);

        LD7\_ON;

Add the function MagInit, copying completely and fixing a similar function for the accelerometer (AccInit). The incoming argument will already be 32-bit, because We will use 3 different registers and 16-bit will not be enough for us

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

void **Mag**Init (uint **32**\_t InitStruct)

{

        uint8\_t ctrl = 0x00;

        ctrl = (uint8\_t) InitStruct;

        Accel\_IO\_Write ( **MAG**\_I2C\_ADDRESS, LSM303DLHC\_CRA\_REG\_M, ctrl);

        ctrl = (uint8\_t) (InitStruct >> 8);

        Accel\_IO\_Write ( **MAG**\_I2C\_ADDRESS, LSM303DLHC\_CRB\_REG\_M, ctrl);

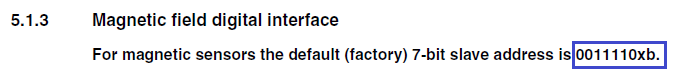
**ctrl = (uint8\_t) (InitStruct >> 16);**

**Accel\_IO\_Write (MAG\_I2C\_ADDRESS, LSM303DLHC\_MR\_REG\_M, ctrl);**

}

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

We learn about the registers and settings of their bits a little later, when we will enter the values ​​in them before calling this function. If you notice, then the address of I2C for communication with the magnetometer we will already be different



We add the code for setting the initialization registers to the general initialization function, for this we first add one more variable for the possibility of transferring the 32-bit value of the settings

        uint16\_t ctrl = 0x0000;

**uint32\_t ctrl32 = 0x00000000;**

-

// Accel\_AccFilter (ctrl);

**ctrl32 | = (LSM303DLHC\_TEMPSENSOR\_DISABLE | LSM303DLHC\_ODR\_220\_HZ);**

**ctrl32 | = LSM303DLHC\_FS\_4\_0\_GA << 8;**

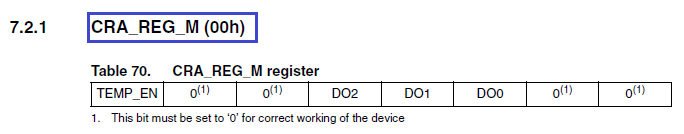
**ctrl32 | = LSM303DLHC\_CONTINUOS\_CONVERSION << 16;**

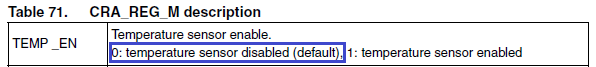
**MagInit (ctrl32);**

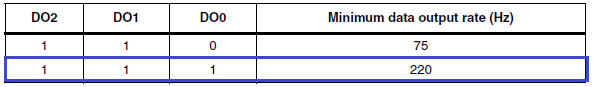
        LD7\_ON;

And now we'll get to know briefly what is going on with the settings of control register bits.

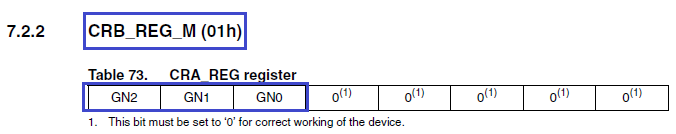
In the register **CRA\_REG\_M** ( **00h**), we disconnect the temperature sensor and adjust the frequency of data reading from the axes of the magnetometer

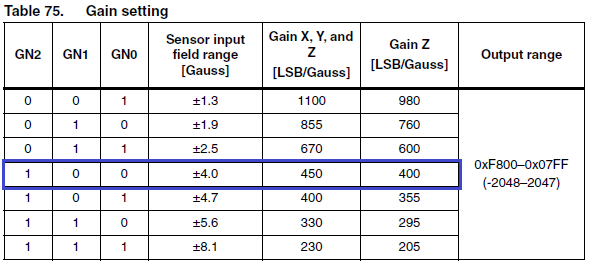




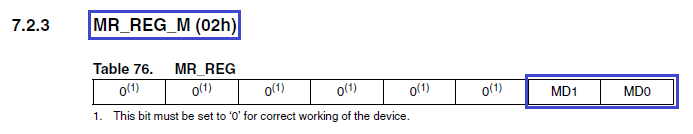


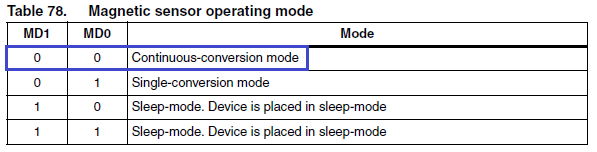
In the register **CRB\_REG\_M** ( **01h**), adjust the measurement limit of the magnetometer





In the **MR\_REG\_M** ( **02h**) register, **enable the**continuous conversion mode





In the [**next part of**](http://narodstream.ru/stm-urok-49-hal-magnitometr-lsm303dlhc-chast-2/) our lesson, we'll write a sensor data reading function and try to see the readings on the PC first in the terminal program, and then in the two visualization programs.

**Lesson 49**

**Part 2**

**HAL. Magnetometer LSM303DLHC**

In the [last part of the](http://narodstream.ru/stm-urok-49-hal-magnitometr-lsm303dlhc-chast-1/) lesson we briefly studied the documentation for the magnetometer, created a project for it, added some macros and wrote the sensor initialization function.

Add the function of reading data from the axes of the magnetometer, copying and correcting the same for the axes of the accelerometer

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

void **Mag**\_GetXYZ (int16\_t \* pData)

{

        uint8\_t buffer [6];

        uint8\_t i = 0;

        buffer [0] = Accel\_IO\_Read ( **MAG\_I2C\_ADDRESS, LSM303DLHC\_OUT\_X\_H\_M**);

        buffer [1] = Accel\_IO\_Read ( **MAG\_I2C\_ADDRESS, LSM303DLHC\_OUT\_X\_L\_M**);

        buffer [2] = Accel\_IO\_Read ( **MAG\_I2C\_ADDRESS, LSM303DLHC\_OUT\_Y\_H\_M**);

        buffer [3] = Accel\_IO\_Read ( **MAG\_I2C\_ADDRESS, LSM303DLHC\_OUT\_Y\_L\_M**);

        buffer [4] = Accel\_IO\_Read ( **MAG\_I2C\_ADDRESS, LSM303DLHC\_OUT\_Z\_H\_M**);

        buffer [5] = Accel\_IO\_Read ( **MAG\_I2C\_ADDRESS, LSM303DLHC\_OUT\_Z\_L\_M**);

                for (i = 0; i <3; i ++)

                {

**if (pData [i]! = - 4096) pData**[i] = ((int16\_t) ((uint16\_t) buffer [ **2 \* i**] << 8) + buffer [ **2 \* i + 1**]);

                }

}

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

Accel\_ReadAcc function will change the name to Accel **Mag**\_Read, also changing it and uncommenting it in the main.c file in an infinite loop

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

void Accel **Mag**\_Read (void)

{

  / \* USER CODE BEGIN 3 \* /

                Accel **Mag**\_Read ();

  }

Now let's make some changes to this function. Instead of different variables for the axes, we will now use an array. A little later we will find out what it is for

void AccelMag\_Read (void)

{

        int16\_t buffer [3] = {0};

**static int16\_t val [3], tmp16;**

**Mag**\_GetXYZ (buffer);

Write this function on

        Mag\_GetXYZ (buffer);

        // remove the overflow from the axes

**tmp16 = buffer [0]; if (tmp16! = - 4096) val [0] = tmp16;**

**tmp16 = buffer [1]; if (tmp16! = - 4096) val [1] = tmp16;**

**tmp16 = buffer [2]; if (tmp16! = - 4096) val [2] = tmp16;**

From the comment here in principle everything is clear.

Next, a little bit of code

        tmp16 = buffer [2]; if (tmp16! = - 4096) val [2] = tmp16;

        sprintf (str1, "X:% 06d Y:% 06d Z:% 06drn", val **[0]**, val **[1]**, val **[2]**);

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

// buf2 [0] = 0x11;

// buf2 [1] = 0x55;

// buf2 [2] = (uint8\_t) (val **[0]**>> 8);

// buf2 [3] = (uint8\_t) val **[0]**;

// buf2 [4] = (uint8\_t) (val **[1]**>> 8);

// buf2 [5] = (uint8\_t) val **[1]**;

// buf2 [6] = (uint8\_t) (val **[2]**>> 8);

// buf2 [7] = (uint8\_t) val **[2]**;

// HAL\_UART\_Transmit (& huart2, buf2,8,0 × 1000);

We fix the code further, changing also the LEDs in places in connection with the definite position of the axes of the magnetometer

// HAL\_UART\_Transmit (& huart2, buf2,8,0 × 1000);

        if (val [0]> 60)

        {

                LD10\_ON;

                if (val [1]> 60)

                {

                        LD10\_OFF;

                        LD9\_ON;

                }

                if (val [1] <- 60)

                {

                        LD10\_OFF;

                        LD8\_ON;

                }

        }

        else if (val [0] <- 60)

        {

                LD3\_ON;

                if (val [1]> 60)

                {

                        LD3\_OFF;

                        LD5\_ON;

                }

                if (val [1] <- 60)

                {

                        LD3\_OFF;

                        LD4\_ON;

                }

        }

        else

        {

                if (val [1]> 60)

                {

                        LD7\_ON;

                }

                if (val [1] <- 60)

                {

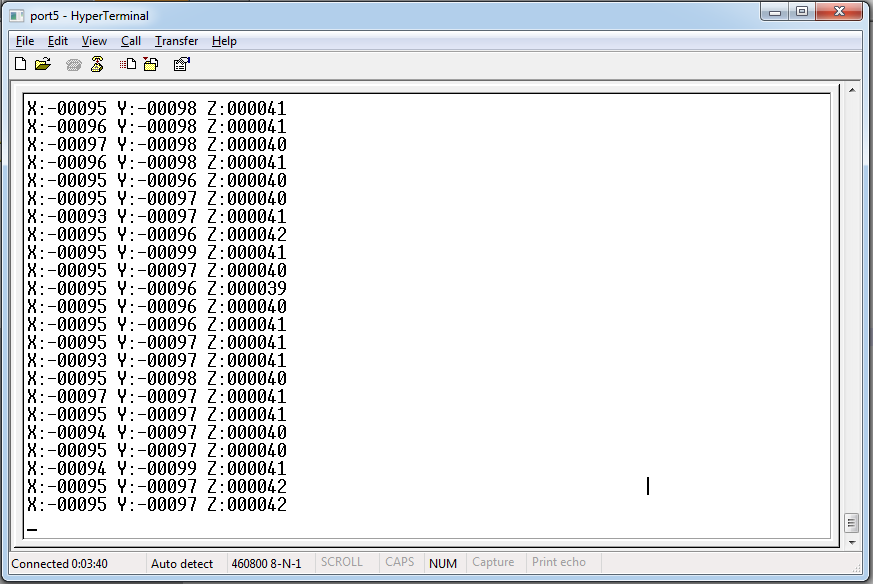
                        LD6\_ON;

                }

        }

        HAL\_Delay ( **10**);

We will collect the code, we will sew the controller and we will look in the terminal indications of axes



We'll turn the board around all three and check the readings. Based on this reconciliation, we will update the zeros in the code. That is, we compare the maximum positive reading for each axis with the maximum negative reading along the same axis, add them and divide the result in half. It is best to orient each axis vertically in different directions. In general, if the testimony will differ only in sign, then there is no need to correct it, and if there is a difference in modulus, then we correct it by the sign in the direction of the smaller one.

For example, the X reading is an average of -165 and 165, which means that we do not need an adjustment for this axis.

Do the same for the other three axes. I did it like this

        tmp16 = buffer [0]; if (tmp16! = - 4096) val [0] = tmp16;

        tmp16 = buffer [1]; if (tmp16! = - 4096) val [1] = tmp16 **+40**;

        tmp16 = buffer [2]; if (tmp16! = - 4096) val [2] = tmp16 **-50**;

In general, if we implement a compass, then in a good way we must connect an accelerometer with a gyroscope and in the values ​​along the axes we must also take into account the value of roll and pitch in addition to the magnetic induction. But since we use only a magnetometer, we will have to do only with the indication of magnetic induction.

Now we will uncomment the code to visualize the data, and for the text output, on the contrary, we comment

        tmp16 = buffer [2]; if (tmp16! = - 4096) val [2] = tmp16-50;

// sprintf (str1, "X:% 06d Y:% 06d Z:% 06drn", val [0], val [1], val [2]);

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

        buf2 [0] = 0x11;

        buf2 [1] = 0x55;

        buf2 [2] = (uint8\_t) (val [0] >> 8);

        buf2 [3] = (uint8\_t) val [0];

        buf2 [4] = (uint8\_t) (val [1] >> 8);

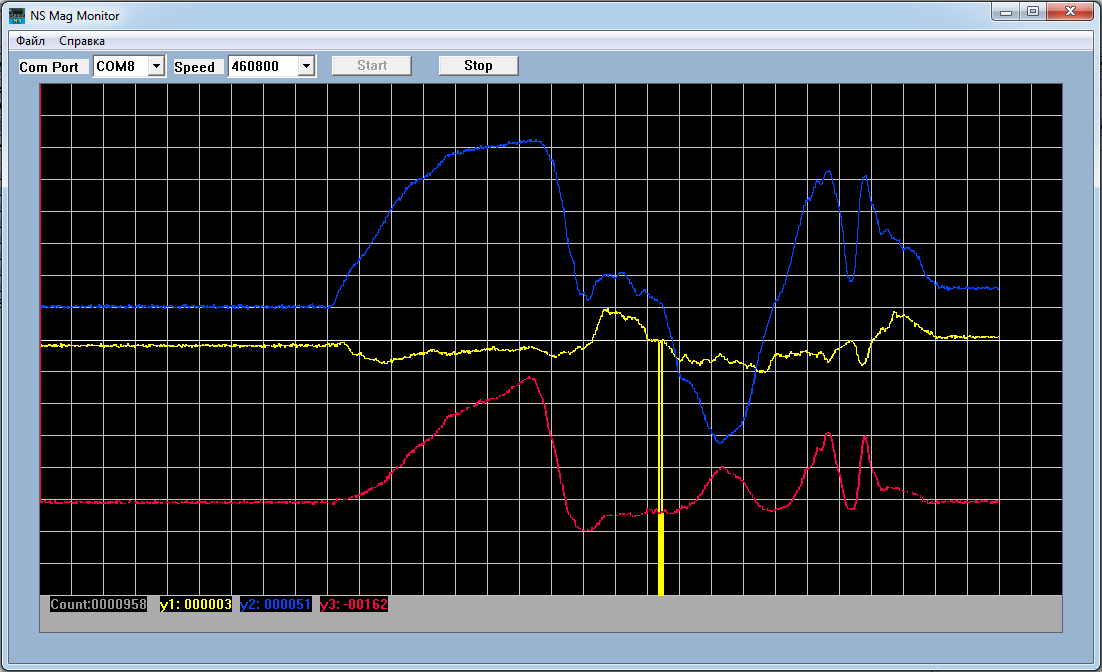
        buf2 [5] = (uint8\_t) val [1];

        buf2 [6] = (uint8\_t) (val [2] >> 8);

        buf2 [7] = (uint8\_t) val [2];

        HAL\_UART\_Transmit (& huart2, buf2.8.0 × 1000);

I slightly corrected my data visualization programs under the magnetometer proper. Let's start the first one. Let's twist a little fee



From this picture, we see that our magnetometer works fine, only the only one, there is some noise. Let's write some low-pass filter. The simplest filter, but at the same time it seemed to me quite working, was the moving average. I have researched some of the best practices on this topic, and yet decided to compose my own implementation. In general, the moving average is based on the fact that we collect a certain number of readings in succession of each measured value, calculate their average arithmetic and use it already at the output. Then, after the next measured value, we take the same number of last readings and also calculate their arithmetic mean. But in fact, this operation is slightly different. We first type the buffer from a certain number of consecutive readings, we find the arithmetic mean, but also remember their sum, not divided by the number of readings. Then we add to this amount the next measured reading and subtract the oldest reading, and then divide by the number of readings. And so on. That is, for large numbers of values, we do not have to add them each time, which brings optimization to the code and saves CPU time.

I implemented this algorithm as follows. I typed the buffer, found the average arithmetic. Then I take the buffer part with the number of elements smaller by one than the total number of elements of the buffer and shift them as a normal memory section. Thus, one element is released, into which we will write a new value.This trick works only one way, the other does not work. In the other, you must first copy to another buffer then copy it to the correct part of it. This does not suit us, since the operation time is doubled, and we therefore choose the first option.

Here, actually, is the code of the moving average calculation function. First, add global variables

char str1 [30] = {0};

**// buffers for the moving average**

**volatile int16\_t xbuf\_avg [8] = {0}, ybuf\_avg [8] = {0}, zbuf\_avg [8] = {0};**

**// the counter of buffers of moving average**

**volatile int8\_t avg\_cnt;**

**// sum for the arithmetic mean**

**volatile int64\_t tmp64 [3];**

Now the function itself

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

**void MovingAverage (int16\_t \* dt)**

**{**

**if (avg\_cnt <8)**

**{**

**xbuf\_avg [avg\_cnt] = dt [0];**

**ybuf\_avg [avg\_cnt] = dt [1];**

**zbuf\_avg [avg\_cnt] = dt [2];**

**if (avg\_cnt == 7)**

**{**

**tmp64 [0] = xbuf\_avg [7] + xbuf\_avg [6] + xbuf\_avg [5] + xbuf\_avg [4] + xbuf\_avg [3] + xbuf\_avg [2] + xbuf\_avg [1] + xbuf\_avg [0];**

**tmp64 [1] = ybuf\_avg [7] + ybuf\_avg [6] + ybuf\_avg [5] + ybuf\_avg [4] + ybuf\_avg [3] + ybuf\_avg [2] + ybuf\_avg [1] + ybuf\_avg [0];**

**tmp64 [2] = zbuf\_avg [7] + zbuf\_avg [6] + zbuf\_avg [5] + zbuf\_avg [4] + zbuf\_avg [3] + zbuf\_avg [2] + zbuf\_avg [1] + zbuf\_avg [0];**

**dt [0] = tmp64 [0] / 8;**

**dt [1] = tmp64 [1] / 8;**

**dt [2] = tmp64 [2] / 8;**

**}**

**avg\_cnt ++;**

**}**

**else**

**{**

**// subtract from the total sums the last elements**

**tmp64 [0] - = xbuf\_avg [0];**

**tmp64 [1] - = ybuf\_avg [0];**

**tmp64 [2] - = zbuf\_avg [0];**

**// shift the buffers to 1 element**

**memcpy ((void \*) xbuf\_avg, (void \*) (xbuf\_avg + 1), sizeof (int16\_t) \* 7);**

**memcpy ((void \*) ybuf\_avg, (void \*) (ybuf\_avg + 1), sizeof (int16\_t) \* 7);**

**memcpy ((void \*) zbuf\_avg, (void \*) (zbuf\_avg + 1), sizeof (int16\_t) \* 7);**

**// replace the 7 elements with new ones**

**xbuf\_avg [7] = dt [0];**

**ybuf\_avg [7] = dt [1];**

**zbuf\_avg [7] = dt [2];**

**// add new items**

**tmp64 [0] + = dt [0];**

**tmp64 [1] + = dt [1];**

**tmp64 [2] + = dt [2];**

**// update the mean values**

**dt [0] = tmp64 [0] / 8;**

**dt [1] = tmp64 [1] / 8;**

**dt [2] = tmp64 [2] / 8;**

**}**

**}**

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

In the initialization function AccelMag\_Ini we add the initialization of the counter

        uint32\_t ctrl32 = 0x00000000;

**// initialize the moving average buffer**

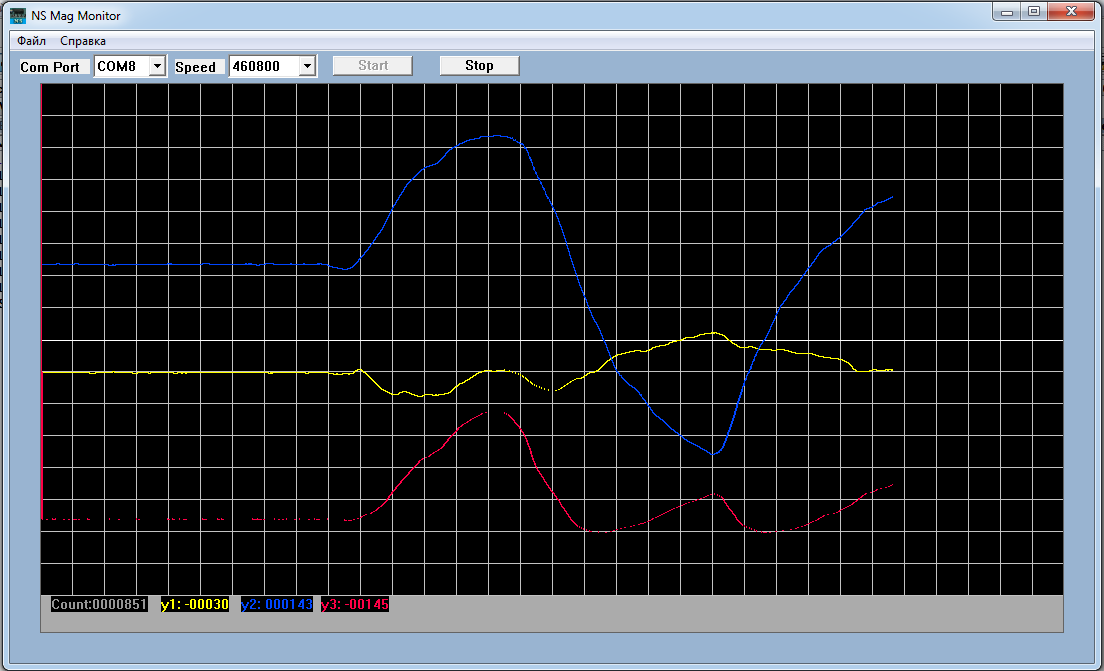
**avg\_cnt = 0; // fill counter**

In function AccelMag\_Read we call our filter function

        tmp16 = buffer [2]; if (tmp16! = - 4096) val [2] = tmp16-50;

**MovingAverage (val);**

We'll compile the code, we'll make the controller and see what we get



I think there are changes.

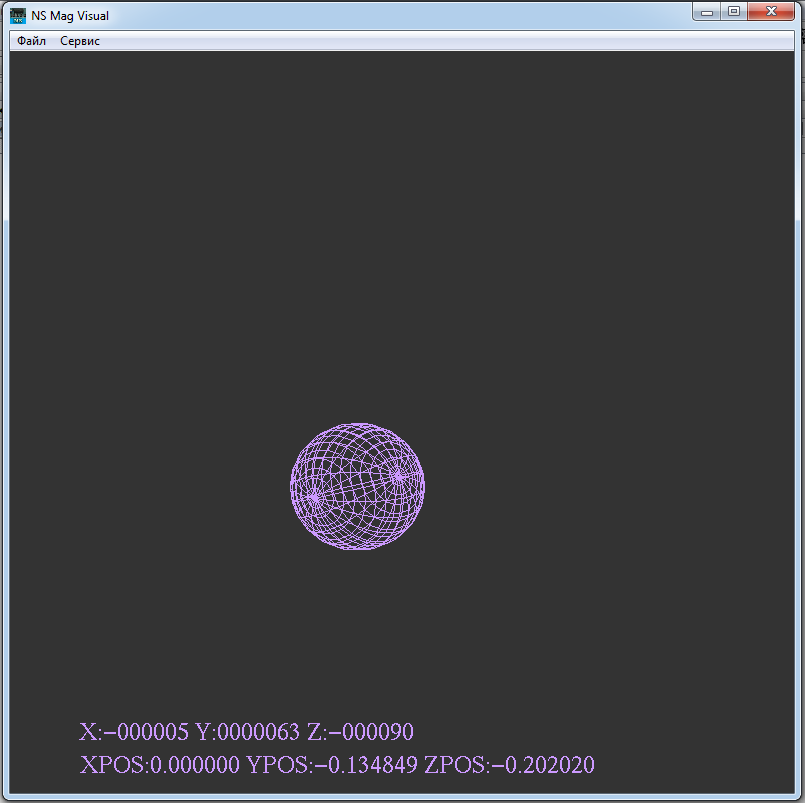
Now remove all the adjustments

        tmp16 = buffer [0]; if (tmp16! = - 4096) val [0] = tmp16;

        tmp16 = buffer [1]; if (tmp16! = - 4096) val [1] = tmp16;

        tmp16 = buffer [2]; if (tmp16! = - 4096) val [2] = tmp16;

and run another visualization program



But the ball at us quickly was rolled, as constantly operate magnetic fields. Correct now the indications on the axes in the code, so that they are generally zero

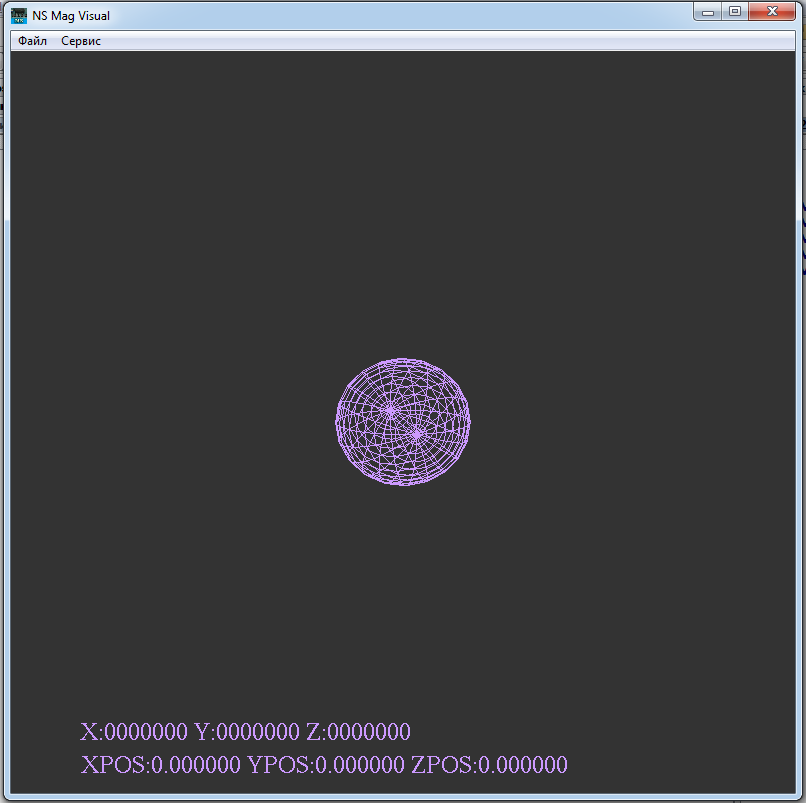
// remove the overflow from the axes

        tmp16 = buffer [0]; if (tmp16! = - 4096) val [0] = tmp16 **+6**;

        tmp16 = buffer [1]; if (tmp16! = - 4096) val [1] = tmp16 **-64**;

        tmp16 = buffer [2]; if (tmp16! = - 4096) val [2] = tmp16 **+90**;

Run the visualization program again. The ball will now roll away when we put up the magnet, and also rotate it



Well, that's all. I think we got to know each other and learned how to work with this magnetometer