**Lesson 39**

**We connect the accelerometer LSM303DLHC**

**Part 1**

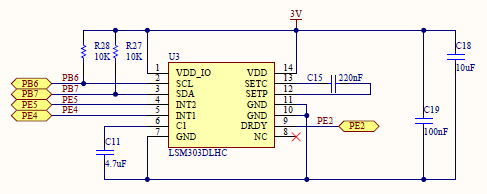
From today's lesson we start to study a difficult, but interesting topic - connecting sensors made using**MEMS** technology .

Since I had several **X-NUCLEO** extender boards , I decided to do a series of lessons on removing and processing the readings from the **MEMS** sensors installed on them. But since the sensors installed on these boards are not particularly simple, I decided to start studying with simpler MEMS sensors installed on other boards. I think that it will be easier for you to study this topic. As they say, from simple to complex.

First, we'll start the cycle of training with accelerometers, because they seemed to me easier, most likely because of their programming, I at least heard something and saw it.

We will begin our study with an accelerometer-magnetometer installed on the **STM32F3Discovery** board. This sensor is called **LSM303DLHC** . And we will study exactly one part of it - an accelerometer.

Let's see a diagram of our **F3** board . We will see there the connection of our accelerometer-magnetometer



The technical characteristics of this sensor are described in the datasheet on it, a link to which I will give at the end of the article. The main characteristics that we need are those that it is 16-bit, because there are also 8-bit accelerometers of the old type, but I think we will not touch them, because this is no longer true.

Also we need the following characteristics:

interface for taking readings - I2C;

The reading range is ± 2g / ± 4g / ± 8g / ± 16g;

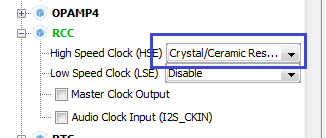
Sensitivity 1-12 mg / LSB

Deviation from zero ± 60 mg.

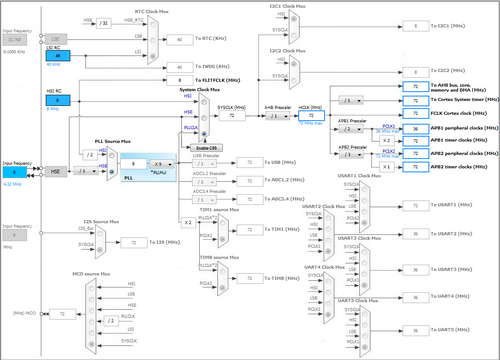
With the rest of the characteristics, subtleties, registers and other pitfalls of the accelerometer, we will get acquainted in the course of its programming.

Run the Cube MX. Create a new project. We choose the controller STM32F303VCTx.

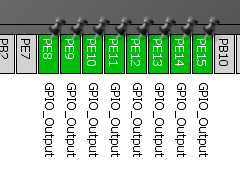
        We turn on the clock from the quartz resonator.



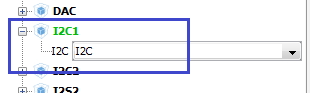
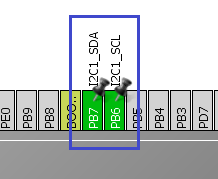
Set the Clock Configuration to maximum (click on the image to enlarge)

[](http://narodstream.ru/wp-content/uploads/2016/11/image07_1121.png)

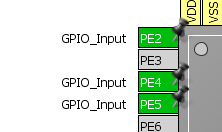
Include the tabs of the ports responsible for controlling the LEDs on the board, to the output



Turn on I2C1 redefine the legs on PB6 and PB7, because it is to these legs that we have an accelerometer connected.

Also, we'll plug in the remaining ports of the port, to which the remaining legs of the accelerometer are connected (the legs of the interrupts). Most likely we will not use them in the framework of our lesson, but we'll still include it for the order, so as not to confuse and not use them for something else.

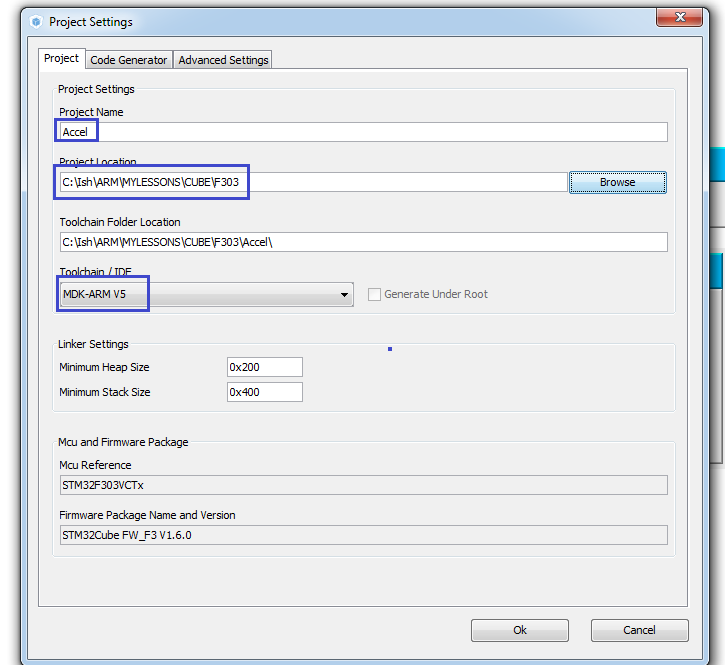


We also turn on USART2. On it we will transfer the taken readings from the sensor to the PC (click on the picture to enlarge).

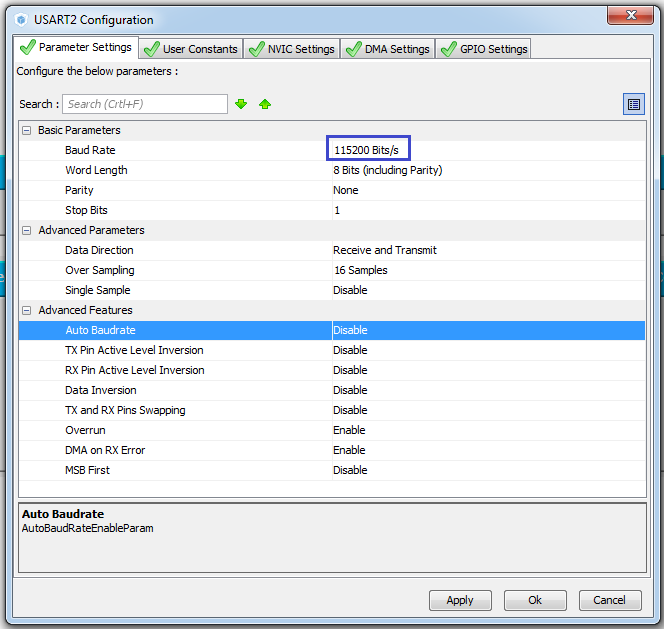
[](http://narodstream.ru/wp-content/uploads/2016/11/image10_0851.png)

Go to Configuration and make the following settings. The speed of USART will be 115200 bps. In principle, for the time being we will not touch anything else.

We will set up the project. For this, go to Project -> Settings. In the top line we will name the project Accel. In the next line, we select the place where our project will be located, also we choose the programming environment - MDK-ARM V5



        In Configuration in the USART2 settings, set the speed to 115200 bps.



Save the settings by clicking the "Ok" button.

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

Create a file main.h, including some macro substitutions, which give us the convenience in the future to work with the ports of the ports for LEDs:

#ifndef MAIN\_H\_

#define MAIN\_H\_

#include "stm32f3xx\_hal.h"

#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 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

#endif / \* MAIN\_H\_ \* /

Connect this file to main.c

/ \* USER CODE BEGIN Includes \* /

#include "main.h"

/ \* USER CODE END Includes \* /

Also create two files in Src and Inc with the respective names lsm303dlhc.c and lsm303dlhc.h by the name of our sensor.

We connect the file lsm303dlhc.h in the main.h file

#include "stm32f3xx\_hal.h"

**#include "lsm303dlhc.h"**

Also connect it to lsm303dlhc.c.

Contents of this file:

#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 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

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

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

To connect any external device, as we already know, initial initialization is required.

Therefore, create a function in the file lsm303dlhc.c

#include "lsm303dlhc.h"

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

**void Accel\_Ini (void)**

**{**

**}**

Let's create a prototype for it in the header file lsm303dlhc.h

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

**void Accel\_Ini (void);**

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

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

and call the main () function in the main function

  / \* USER CODE BEGIN 2 \* /

**Accel\_Ini ();**

  / \* USER CODE END 2 \* /

We also know that in the initialization an important role is played by reading the device identifier, not so much to find out whether we have attached the device, how many to make sure that we with this device successfully communicate on the bus and do not think, that some glitches we have because of an incorrect connection. Let's write one more function in lsm303dlhc.c

#include "lsm303dlhc.h"

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

**uint8\_t Accel\_ReadID (void)**

**{**

**uint8\_t ctrl = 0x00;**

**return ctrl;**

**}**

In the [**next part of**](http://narodstream.ru/stm-urok-39-podklyuchaem-akselerometr-lsm303dlhc-chast-2/) our lesson, we will continue to write the initialization of this sensor. Let's write convenient functions for reading and writing the accelerometer registers using the I2C bus, and also counting the identifier from the sensor.

In the [**previous part of**](http://narodstream.ru/stm-urok-39-podklyuchaem-akselerometr-lsm303dlhc-chast-1/) this lesson, we briefly studied the sensor documentation, created a project for it, added some macros, and started writing the sensor initialization function.

Now we need the following function

#include "lsm303dlhc.h"

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

**uint8\_t Accel\_IO\_Read (uint16\_t DeviceAddr, uint8\_t RegisterAddr)**

**{**

**}**

In bold type, I will allocate new fragments. And I place the old ones in the article in order to understand exactly where to write the code. Also, the functions we call from others, I try to write higher, otherwise I have to write a prototype. Add the declaration of the bus pointer I2C

#include "lsm303dlhc.h"

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

**extern I2C\_HandleTypeDef hi2c1;**

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

The following two functions are in the same file:

extern I2C\_HandleTypeDef hi2c1;

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

**static void Error (void)**

**{**

**LD3\_ON;**

**}**

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

**static uint8\_t I2Cx\_ReadData (uint16\_t Addr, uint8\_t Reg)**

**{**

**HAL\_StatusTypeDef status = HAL\_OK;**

**uint8\_t value = 0;**

**status = HAL\_I2C\_Mem\_Read (& hi2c1, Addr, Reg, I2C\_MEMADD\_SIZE\_8BIT, & value, 1, 0x10000);**

**if (status! = HAL\_OK)**

**{**

**/ \* Execute user timeout callback \* /**

**Error ();**

**}**

**return value;**

**}**

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

Now, filling in the functions higher in the file, we will move from top to bottom along the remaining functions. We add the function Accel\_IO\_Read:

uint8\_t Accel\_IO\_Read (uint16\_t DeviceAddr, uint8\_t RegisterAddr)

{

**return I2Cx\_ReadData (DeviceAddr, RegisterAddr);**

}

Now add the function Accel\_ReadID:

uint8\_t ctrl = 0x00;

**ctrl = Accel\_IO\_Read (0x32, 0x0F);**

return ctrl;

Well, let's add the function of reading the identifier of the sensor to the initialization function:

void Accel\_Ini (void)

{

  uint16\_t ctrl = 0x0000;

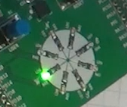
        HAL\_Delay (1000);

        if (Accel\_ReadID () == 0x33) LD6\_ON;

        else Error ();

}

The delay is added in view of the fact that the sensor needs some time to turn on. Those. If the identifier matches the identifier from the datasheet, then we light the green LED, and if not, we will call the error handler, which will light the red LED. We will sew the controller and check it



If everything is OK and the LED is green, continue on. If not, then I advise you to distort the board several times, go through the debugger, repeat the process. It happens.

We will introduce some macrosubstructions into the lsm303dlhc.h file for the registers and register values ​​of our sensor

#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\_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\_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 \* /**

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

void Accel\_Ini (void);

Let's continue our initialization. To do this, write a convenient function to send data to the I2C bus

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

**static void I2Cx\_WriteData (uint16\_t Addr, uint8\_t Reg, uint8\_t Value)**

**{**

**HAL\_StatusTypeDef status = HAL\_OK;**

**status = HAL\_I2C\_Mem\_Write (& hi2c1, Addr, (uint16\_t) Reg, I2C\_MEMADD\_SIZE\_8BIT, & Value, 1, 0x10000);**

**/ \* Check the communication status \* /**

**if (status! = HAL\_OK)**

**{**

**/ \* Execute user timeout callback \* /**

**Error ();**

**}**

**}**

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

Another convenient feature for I2C

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

**void Accel\_IO\_Write (uint16\_t DeviceAddr, uint8\_t RegisterAddr, uint8\_t Value)**

**{**

**I2Cx\_WriteData (DeviceAddr, RegisterAddr, Value);**

**}**

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

We will also write a function for initializing accelerometer settings:

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

void AccInit (uint16\_t InitStruct)

{

        uint8\_t ctrl = 0x00;

        ctrl = (uint8\_t) InitStruct;

        Accel\_IO\_Write (0x32, LSM303DLHC\_CTRL\_REG1\_A, ctrl);

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

        Accel\_IO\_Write (0x32, LSM303DLHC\_CTRL\_REG4\_A, ctrl);

}

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

# We connect the accelerometer LSM303DLHC

**Part 3**

In the [**previous part of**](http://narodstream.ru/stm-urok-39-podklyuchaem-akselerometr-lsm303dlhc-chast-2/) this lesson, we continued to write the initialization of this sensor. Wrote the convenient functions of reading and writing the accelerometer registers using the I2C bus, and also counting the identifier from the sensor.

Next, configure some registers in the basic initialization of the sensor:

        else Error ();

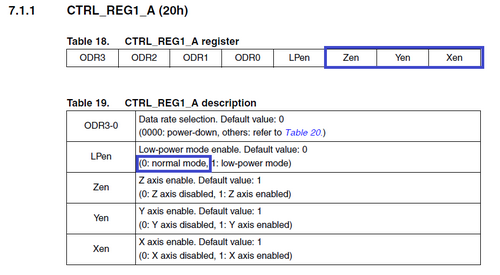
**ctrl | = (LSM303DLHC\_NORMAL\_MODE | LSM303DLHC\_ODR\_50\_HZ | LSM303DLHC\_AXES\_ENABLE);**

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

**AccInit (ctrl);**

}

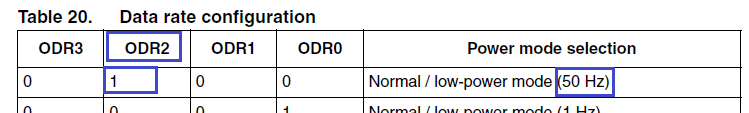
Now I'll try to explain what exactly we set up. The **LSM303DLHC\_CTRL\_REG1\_A** and**LSM303DLHC\_CTRL\_REG4\_A**registers are two setting registers with addresses **0x20** and **0x23**(see technical documentation for the sensor on pages 24-27) (click on the picture to enlarge).

[](http://narodstream.ru/wp-content/uploads/2016/11/image16_0877.png)

**LSM303DLHC\_NORMAL\_MODE**: value **0x00**. Those. We do not include the low-power bit of the sensor, we need a full-fledged mode.

**LSM303DLHC\_AXES\_ENABLE**: value **0x07**. Here we include all three low-order bits (Zen, Yen and Xen). That is, we will work with all three axes of coordinates.

**LSM303DLHC\_ODR\_50\_HZ**: value **0x40**. We include only the sixth bit of ODR2, that is, we set the speed to 50 Hz.



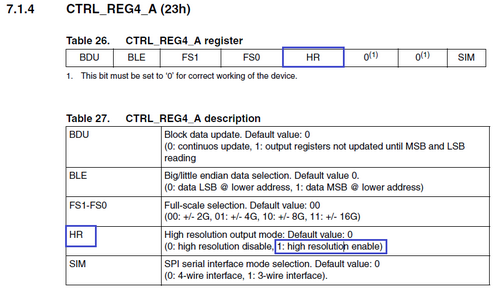
Now go to register **23h**.

In it we will include:

**LSM303DLHC\_BlockUpdate\_Continous**: value **0x00**. Thus, we tell the sensor that we will not include the most senior bit of **BDU in it**, which is responsible for protecting from writing one byte of the two while taking readings from the other. We do not need this, as this will somewhat complicate the recording of the readings in the register.

**LSM303DLHC\_BLE\_LSB**: value **0x00.** Here we tell the accelerometer that the **BLE**bit , which is responsible for filling the bytes of the sensor readings on the contrary (the older byte becomes the younger one), we also will not include.

**LSM303DLHC\_HR\_ENABLE**: value **0x08**. But the **HR**byte , which initializes the output mode in high resolution, we still include (click on the image to enlarge).

[](http://narodstream.ru/wp-content/uploads/2016/11/image15_0855.png)

This initialization of the sensor is not over yet. Let's write one more function for filtering control.

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

**void Accel\_AccFilterConfig (uint8\_t FilterStruct)**

**{**

**uint8\_t tmpreg;**

**/ \* Read CTRL\_REG2 register \* /**

**tmpreg = Accel\_IO\_Read (0x32, LSM303DLHC\_CTRL\_REG2\_A);**

**tmpreg & = 0x0C;**

**tmpreg | = FilterStruct;**

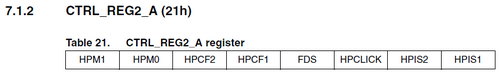
**/ \* Write value to ACC MEMS CTRL\_REG2 register \* /**

**Accel\_IO\_Write (0x32, LSM303DLHC\_CTRL\_REG2\_A, tmpreg);**

**}**

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

We call this function from the initialization function, having filled **beforehand**some parameters of the register **CTRL\_REG2\_A (21h)**(click on the picture to enlarge) .

[](http://narodstream.ru/wp-content/uploads/2016/11/image17_0845.png)

And at the end of the initialization, turn on the second green LED, which will signal us about the successful initialization of the sensor, because In the process of composing and debugging code, I encountered some problems and hangs.

        AccInit (ctrl);

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

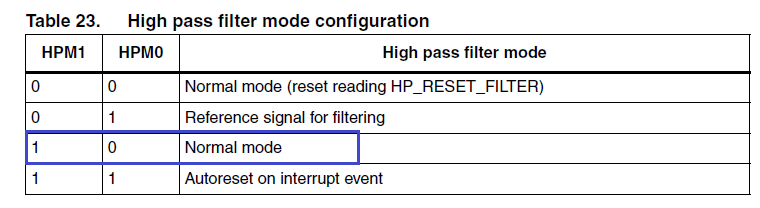
**LSM303DLHC\_HPF\_AOI1\_DISABLE |LSM303DLHC\_HPF\_AOI2\_DISABLE);**

**Accel\_AccFilterConfig (ctrl);**

**LD7\_ON;**

What are the bits of register 0x21 we have included?

**LSM303DLHC\_HPM\_NORMAL\_MODE**: value **0x80**. We have included the bit **HPM1**. This is one of the two bits responsible for configuring the high-pass filter mode. We will use normal mode without rebooting the filters.

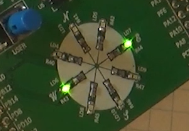


**LSM303DLHC\_HPFCF\_16**: value **0x10**. **Turn on the**bit **HPCF1**, which is one of two bits (HPCF2 and HPCF1), responsible for selecting the cutoff frequency of the high-pass filter. Unfortunately, the technical documentation does not have detailed information about which particular frequency value we are cutting, using this or that combination of data bits. Therefore, this setting was taken from the example, which lies in the folder with the programming environment.

**LSM303DLHC\_HPF\_AOI1\_DISABLE** and **LSM303DLHC\_HPF\_AOI2\_DISABLE**: both values ​​are **0x00**. These are the least significant bits of the register, which are responsible for the inclusion of interrupts on the same legs **INT1** and **INT2**. As part of our lesson, we will not use interrupts, so we do not include these bits.

With the initialization finished.

We will collect the code, we will wring the controller and check by the light of the LED that we have at least nothing left and we did not go into error



In the [**next part of**](http://narodstream.ru/stm-urok-39-podklyuchaem-akselerometr-lsm303dlhc-chast-4/) our lesson we will write a code that will read the sensor readings and will accordingly add it to the buffer for future use in the work.

# We connect the accelerometer LSM303DLHC

**Part 4**

In the [**previous part of**](http://narodstream.ru/stm-urok-39-podklyuchaem-akselerometr-lsm303dlhc-chast-3/) this lesson, we finally completed the initialization of the accelerometer.

Now, at last, the most interesting thing is taking the readings from the sensor. To do this, we first write the following function:

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

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

**{**

**int16\_t pnRawData [3];**

**uint8\_t ctrlx [2] = {0,0};**

**int8\_t buffer [6];**

**uint8\_t i = 0;**

**uint8\_t sensitivity = LSM303DLHC\_ACC\_SENSITIVITY\_2G;**

**/ \* Read the acceleration control register content \* /**

**ctrlx [0] = Accel\_IO\_Read (0x32, LSM303DLHC\_CTRL\_REG4\_A);**

**ctrlx [1] = Accel\_IO\_Read (0x32, LSM303DLHC\_CTRL\_REG5\_A);**

**/ \* Read output register X, Y & Z acceleration \* /**

**buffer [0] = Accel\_IO\_Read (0x32, LSM303DLHC\_OUT\_X\_L\_A);**

**buffer [1] = Accel\_IO\_Read (0x32, LSM303DLHC\_OUT\_X\_H\_A);**

**buffer [2] = Accel\_IO\_Read (0x32, LSM303DLHC\_OUT\_Y\_L\_A);**

**buffer [3] = Accel\_IO\_Read (0x32, LSM303DLHC\_OUT\_Y\_H\_A);**

**buffer [4] = Accel\_IO\_Read (0x32, LSM303DLHC\_OUT\_Z\_L\_A);**

**buffer [5] = Accel\_IO\_Read (0x32, LSM303DLHC\_OUT\_Z\_H\_A);**

**/ \* Check in the control register4 the data alignment \* /**

**if (! (ctrlx [0] & LSM303DLHC\_BLE\_MSB))**

**{**

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

**{**

**pnRawData [i] = ((int16\_t) ((uint16\_t) buffer [2 \* i + 1] << 8) + buffer [2 \* i]);**

**}**

**}**

**else / \* Big Endian Mode \* /**

**{**

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

**{**

**pnRawData [i] = ((int16\_t) ((uint16\_t) buffer [2 \* i] << 8) + buffer [2 \* i + 1]);**

**}**

**}**

**/ \* Normal mode \* /**

**/ \* Switch the sensitivity value set in the CRTL4 \* /**

**switch (ctrlx [0] & LSM303DLHC\_FULLSCALE\_16G)**

**{**

**case LSM303DLHC\_FULLSCALE\_2G:**

**sensitivity = LSM303DLHC\_ACC\_SENSITIVITY\_2G;**

**break;**

**case LSM303DLHC\_FULLSCALE\_4G:**

**sensitivity = LSM303DLHC\_ACC\_SENSITIVITY\_4G;**

**break;**

**case LSM303DLHC\_FULLSCALE\_8G:**

**sensitivity = LSM303DLHC\_ACC\_SENSITIVITY\_8G;**

**break;**

**case LSM303DLHC\_FULLSCALE\_16G:**

**sensitivity = LSM303DLHC\_ACC\_SENSITIVITY\_16G;**

**break;**

**}**

**/ \* Obtain the mg value for the three axis \* /**

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

**{**

**pData [i] = (pnRawData [i] \* sensitivity);**

**}**

**}**

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

Now let's try to deal with macros.

**LSM303DLHC\_ACC\_SENSITIVITY\_xG**is a series of macros for adjusting the sensitivity of the accelerometer. The figure before G is the multiplier, to which we multiply the value of G (9.8 m / s 2 ).

**LSM303DLHC\_CTRL\_REG4\_A**and **LSM303DLHC\_CTRL\_REG5\_A**are the 0x23 and 0x24 registers, of which we consider some of the sensor settings.

**LSM303DLHC\_OUT\_X\_L\_A, LSM303DLHC\_OUT\_X\_H\_A, LSM303DLHC\_OUT\_Y\_L\_A, LSM303DLHC\_OUT\_Y\_H\_A, LSM303DLHC\_OUT\_Z\_L\_A**and **LSM303DLHC\_OUT\_Z\_H\_A are the**registers in which the sensor readings are separately the lower and higher bytes of all three coordinate axes. We put them in certain buffer cells with **buffer**bytes .

Let's **check the**installation of the bit **LSM303DLHC\_BLE\_MSB**, which we mentioned above.Depending on its installation, we distribute the read bytes in the already 16-bit buffer.

At the end of the function, we change the two-byte values ​​of the read values ​​depending on the sensitivity of the sensor set and put them in the global buffer, which we will use later to monitor the readings.

Well, now start writing a custom function to use the readings and monitor them. Let's create several variables that will be useful in the future. We will also distribute the data from the buffer with respect to variables.

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

**void Accel\_ReadAcc (void)**

**{**

**int16\_t buffer [3] = {0};**

**int16\_t xval, yval, zval = 0x00;**

**uint16\_t tmp16 = 0;**

**Accel\_GetXYZ (buffer);**

**xval = buffer [0];**

**yval = buffer [1];**

**zval = buffer [2];**

**}**

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

We add a prototype of this function in lsm303dlhc.h.

**void Accel\_ReadAcc (void);**

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

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

Call this function in the main function of the main () program in an infinite loop

  / \* USER CODE BEGIN 3 \* /

**Accel\_ReadAcc ();**

  }

  / \* USER CODE END 3 \* /

In the [**next part of**](http://narodstream.ru/stm-urok-39-podklyuchaem-akselerometr-lsm303dlhc-chast-5/) our lesson we will finish the work with the accelerometer LSM303DLHC, finish all the code and see the work in practice.

# We connect the accelerometer LSM303DLHC

**Part 5**

In the [**previous part of**](http://narodstream.ru/stm-urok-39-podklyuchaem-akselerometr-lsm303dlhc-chast-4/) this lesson we wrote a code for reading the readings from the sensor and folding it into the buffer for further use in the work.

We continue to write the function Accel\_ReadAcc, which we started writing in the [**previous part of**](http://narodstream.ru/stm-urok-39-podklyuchaem-akselerometr-lsm303dlhc-chast-4/) our lesson . First, we will try to display the changes in the sensor reading by including certain LEDs on the board, since there are as many as eight of them, and therefore we should get an interesting picture.

zval = buffer [2];

**if (xval> 1500)**

**{**

**LD3\_ON;**

**if (yval> 1500)**

**{**

**LD3\_OFF;**

**LD4\_ON;**

**}**

**else if (yval <-1500)**

**{**

**LD3\_OFF;**

**LD5\_ON;**

**}**

**}**

**else if (xval <-1500)**

**{**

**LD10\_ON;**

**if (yval> 1500)**

**{**

**LD10\_OFF;**

**LD8\_ON;**

**}**

**else if (yval <-1500)**

**{**

**LD10\_OFF;**

**LD9\_ON;**

**}**

**}**

**else**

**{**

**if (yval> 1500)**

**{**

**LD6\_ON;**

**}**

**else if (yval <-1500)**

**{**

**LD7\_ON;**

**}**

**}**

**HAL\_Delay (10);**

**LD3\_OFF;**

**LD6\_OFF;**

**LD7\_OFF;**

**LD4\_OFF;**

**LD10\_OFF;**

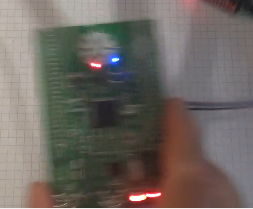
**LD8\_OFF;**

**LD9\_OFF;**

**LD5\_OFF;**

}

We'll compile the project, we'll clean up the controller and see the result. By deflecting the board in different directions, we see a certain luminous LED



But the LEDs are good. But why then did we include USART and plug in the adapter? We'll try to send the readings to the computer and see them there.

Let's add some global variables to the file lsm303dlhc.c.

extern I2C\_HandleTypeDef hi2c1;

**extern UART\_HandleTypeDef huart2;**

**uint8\_t buf2 [14] = {0};**

**char str1 [30] = {0};**

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

Add the following code to our function

zval = buffer [2];

**sprintf (str1, "X:% 06d Y:% 06d Z ::% 06drn", xval, yval, zval);**

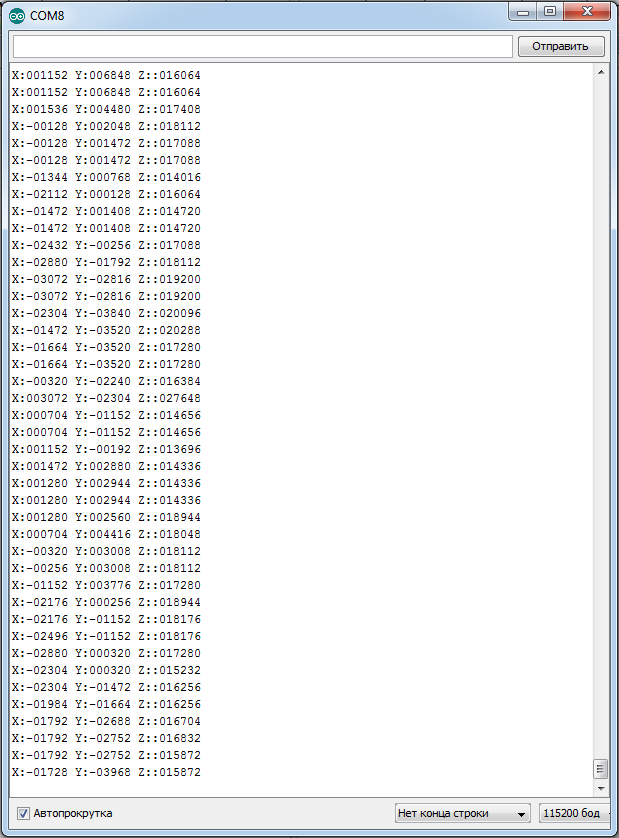
**HAL\_UART\_Transmit (& huart2, (uint8\_t \*) str1, strlen (str1), 0x1000);**

        if (xval> 1500)

We will collect the project, we will sew the microcontroller and try to follow the indications in some terminal program. To monitor the fastest reading, I use the Arduino port monitor, since it slows down a little.

Since we speed USART2 in the Cube MX set 115200 bps, then we will set the same speed in the monitor port.

The following picture should be obtained:



Now let's try to visualize our readings. For this I found a free program SFMonitor.

But before you configure and use it, we'll make some changes to the code. Remove the previous output code on the USART bus well, or comment:

~~sprintf (str1, "X:% 06d Y:% 06d Z ::% 06drn", xval, yval, zval);~~

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

And add a new code for transmission to the bus

  zval = buffer [2];

        buf2 [0] = 0x12;

        buf2 [1] = 0x10;

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

        buf2 [3] = (uint8\_t) xval;

        buf2 [4] = 0x10;

        buf2 [5] = 0x10;

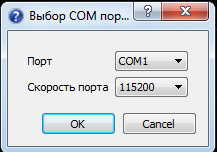
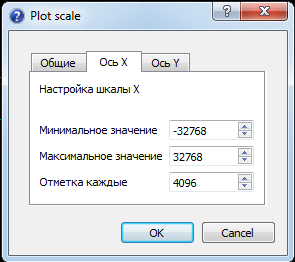
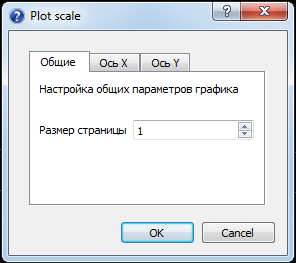
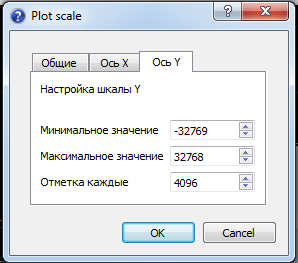
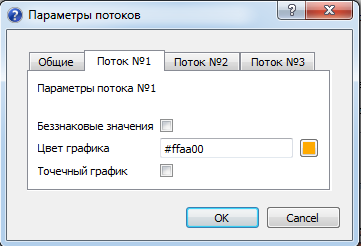
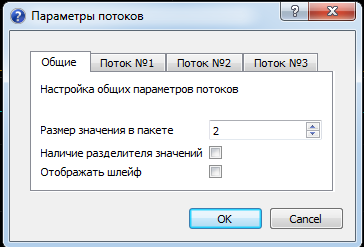
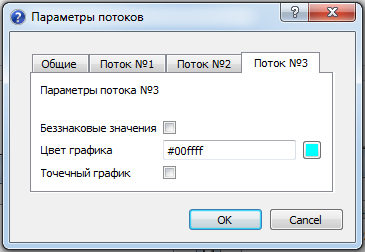
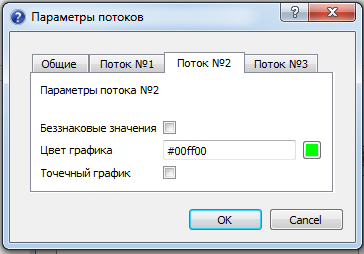
        buf2 [6] = (uint8\_t) (yval >> 8);

        buf2 [7] = (uint8\_t) yval;

        buf2 [8] = 0x13;

Bytes 0x12, 0x10 and 0x13 are inserted into the stream in accordance with the requirements of the program protocol. The explanation is on the developer's website [http://www.poprobot.ru/soft/sfmonitor](https://www.google.com/url?q=http://www.poprobot.ru/soft/sfmonitor&sa=D&ust=1479191517830000&usg=AFQjCNEoDAufjjbDDrAze-pSNMFZXAV4Pw)  .Unfortunately, I could not display all three axes in the program at the same time, apparently some data transmission requirements for it were not met, but the two were simultaneously successful.

We will collect the code, we will sew the controller. Run the SFMonitor program and configure it as follows:

        Let's start the monitor. Let's stir the board in different directions. It should look like this

