**Lesson 42**

**Part 1**

**We connect the accelerometer LSM6DS0**

Today we will consider a sensor that combines two functional elements - an accelerometer and a gyroscope. This accelerometer is also an accelerometer made using MEMS technology - **LSM6DS0**. It is installed on the expansion board **X-NUCLEO-IKS01A1**, designed to work with the debug card Nucleo. We will connect this evaluation board to the Nucleo STM32F401RE board.

This accelerometer-gyroscope also can, along with the I2C interface, connect using the SPI interface. But we will use the connection specifically for I2C, since it is this connection that takes place in the evaluation board X-NUCLEO-IKS01A.

Also, we will use this sensor as an accelerometer in this lesson. As a gyroscope, we will connect it in later studies.

Accelerometer in this sensor has the following technical characteristics:

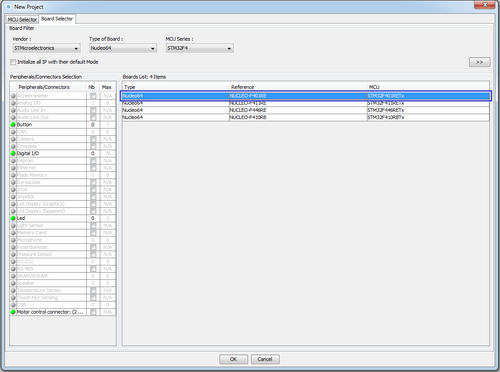
        The reading range is ± 2g / ± 4g / ± 8g / ± 16g;

The sensitivity is 0.061 - 0.73 mg / digit;

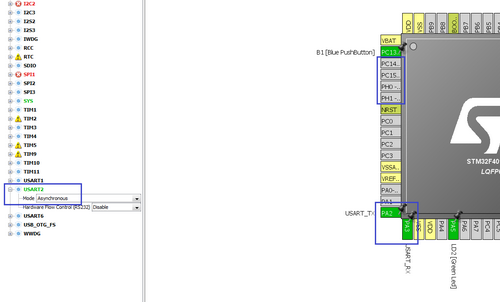
Deviation from zero ± 90 mg.

With some other indicators, registers, values ​​and other details of the accelerometer, we will get acquainted in the course of its programming.

Create a project for the Cube MX. We will not choose a controller, but a debug board (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2016/11/image00_1110.png)

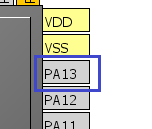
Disable the port legs PH0, PH1, PC14, PC15. USART 2 Turn on the "Asynchronous" mode (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2016/11/image02_1132.png)

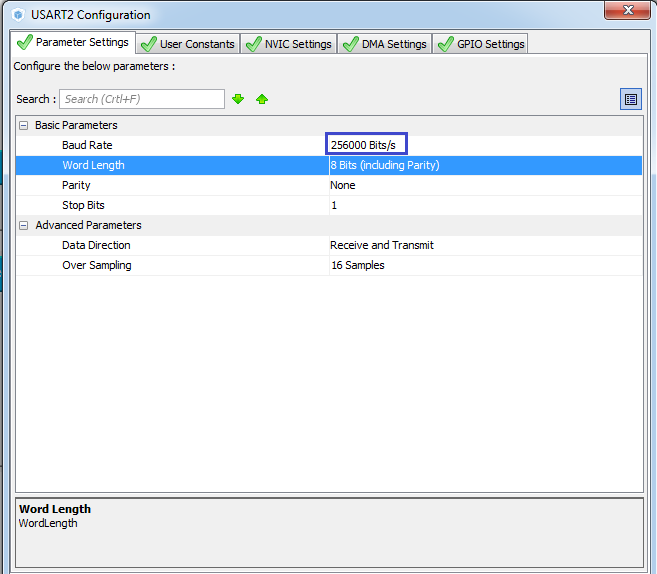
Turn on I2C1, only the tabs SCL and SDA will be redefined to the feet of the PB8 and PB9 ports. This is done by pressing the left mouse button on the leg with the Ctrl key pressed and then transferring it to the possible foot (click on the image to enlarge the image)

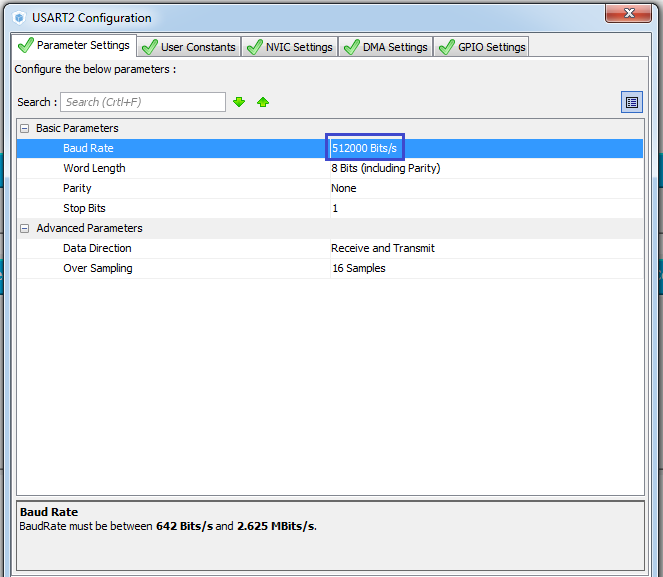
[](http://narodstream.ru/wp-content/uploads/2016/11/image01_1063.png)

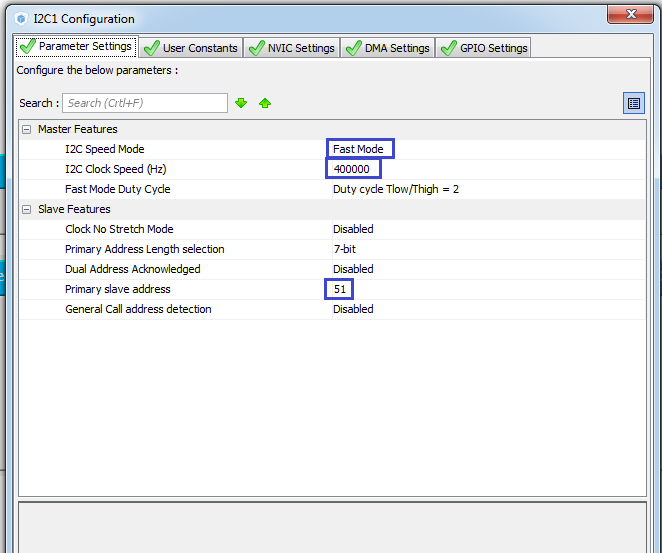
PA13 will also turn off

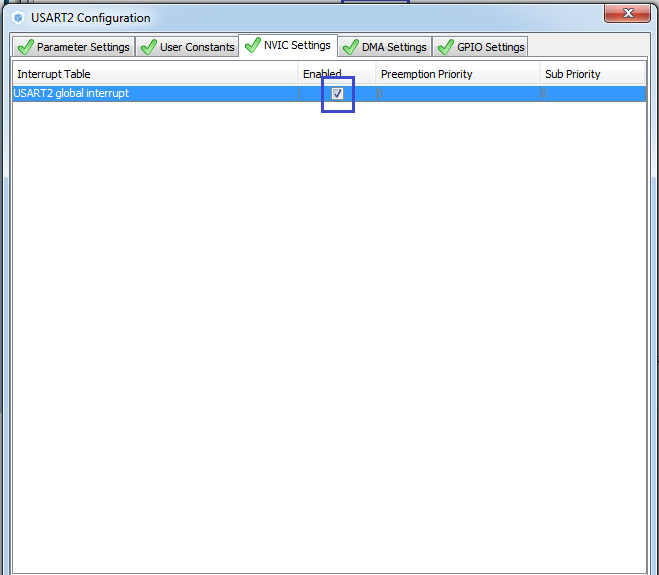


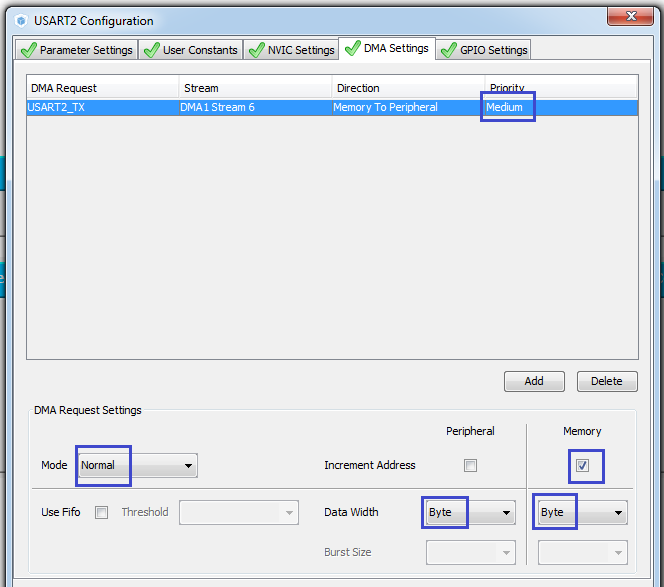
In Clock Configuration, we do not touch anything. In the Configuration I2C, configure at 400 kHz, and USART at 256,000 bps and turn on USART interrupts and DMA.



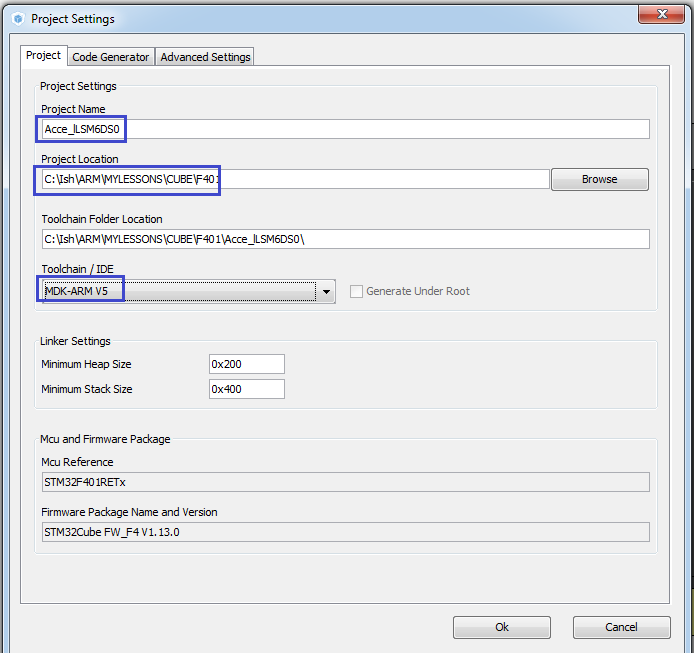








Configure Project Settings by specifying the name Acce\_lLSM6DS0, the programming environment, and the path to the project. The way for everyone can be different.



Generate the project, open it. Let's set up the programmer for auto-cutting. We will compile the project.

To further develop the project, we can take advantage of the experience with previous sensors. We will take the files of the sensor functions libraries with you from the 39th lesson, when we connected a sensor installed on the STM32F303 Discovery board, since we also used I2C and USART buses there.

In the Inc project folder, we copy the main.h file. Also in the corresponding folders Inc and Src we copy the files lsm303dlh.h lsm303dlh.c, respectively renaming them according to the name of our sensor in lsm6ds0.h and lsm6ds0.c.

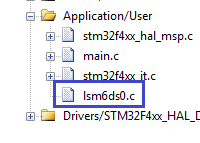
We connect the main.h file in main.c

/ \* USER CODE BEGIN Includes \* /

**#include "main.h"**

/ \* USER CODE END Includes \* /

We also connect the file lsm6ds0.c



The main.h file after some fixes will become this:

#ifndef MAIN\_H\_

#define MAIN\_H\_

#include "stm32f **4**xx\_hal.h"

#include " **lsm6ds0**.h"

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

Also, we'll correct the header file lsm6ds0.h and remove from it any extra macros

#ifndef **LIS3DSH\_H\_**

#define **LIS3DSH\_H\_**

#include "stm32f **4**xx\_hal.h"

#include <string.h>

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

#define ABS (x) (x <0)? (-x): x

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

#define LD2\_Pin GPIO\_PIN\_5

#define LD2\_GPIO\_Port GPIOA

#define LD2\_ON HAL\_GPIO\_WritePin (GPIOA, GPIO\_PIN\_5, GPIO\_PIN\_SET) // GREEN

#define LD2\_OFF HAL\_GPIO\_WritePin (GPIOA, GPIO\_PIN\_5, GPIO\_PIN\_RESET)

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

void Accel\_Ini (void);

void Accel\_ReadAcc (void);

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

#endif / \* **LIS3DSH\_H\_** \* /

Also, we will correct and remove unnecessary in the file lsm6ds0.c

#include " **lsm6ds0**.h"

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

extern I2C\_HandleTypeDef hi2c1;

extern UART\_HandleTypeDef huart2;

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

char str1 [30] = {0};

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

void Error (void)

{

        LD2\_OFF;

}

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

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) Error ();

        return value;

}

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

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);

        if (status! = HAL\_OK) Error ();

}

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

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

{

        return I2Cx\_ReadData (DeviceAddr, RegisterAddr);

}

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

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

{

        I2Cx\_WriteData (DeviceAddr, RegisterAddr, Value);

}

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

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

{

        uint8\_t buffer [6];

        uint8\_t i = 0;

        for (i = 0; i <3; i ++)

        {

        }

}

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

uint8\_t Accel\_ReadID (void)

{

        uint8\_t ctrl = 0x00;

        return ctrl;

}

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

void Accel\_ReadAcc (void)

{

        int16\_t buffer [3] = {0};

        int16\_t xval, yval, zval;

        Accel\_GetXYZ (buffer);

        xval = buffer [0];

        yval = buffer [1];

        zval = buffer [2];

// sprintf (str1, "X:% 06d Y:% 06d Z:% 06d \ r \ n", xval, yval, zval);

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

        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) (zval >> 8);

        buf2 [7] = (uint8\_t) zval;

        buf2 [8] = 0x13;

        HAL\_UART\_Transmit (& huart2, buf2,9.0 × 1000);

        if (xval> 1500)

        {

        }

        HAL\_Delay (20);

}

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

void AccInit (uint16\_t InitStruct)

{

}

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

void Accel\_Ini (void)

{

        uint16\_t ctrl = 0x0000;

        HAL\_Delay (1000);

        AccInit (ctrl);

}

In the main function, add the following code. We will light the LED in advance, since it is the only one on the board and we will call the initialization function

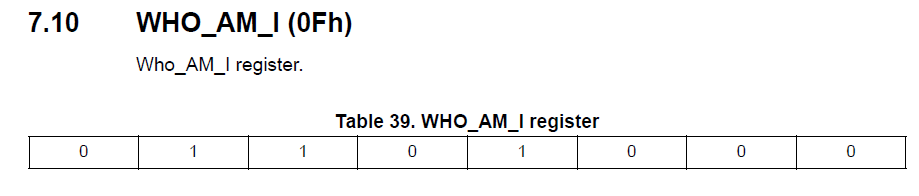
  / \* USER CODE BEGIN 2 \* /

**HAL\_GPIO\_WritePin (LD2\_GPIO\_Port, LD2\_Pin, GPIO\_PIN\_SET);**

**Accel\_Ini ();**

  / \* USER CODE END 2 \* /

Let's start, as usual with an attempt to read the identifier of the sensor. Let's turn to the technical documentation. We first find the address needed to access the device by I2C. If the SDO leg is connected to ground, then 1 bit is 0, if it is powered, then 1. 0th bit will be 0, because we address first in the recording mode, if the playback mode is required, the HAL library itself will take care of installing it in 1. It remains only to find out where the SDO foot is connected. I could not find the scheme. There are 2 options left, or try different addresses, or take it from the example. I chose 2nd path as more simple. Therefore, our address will be the last one from table 15 of the datasheet - 11010110 (D6h). Also we need to know from which register to take the identifier and what it should be to our sensor to apply the comparison operation in the condition.



So add a line to the Accel\_ReadID function

uint8\_t Accel\_ReadID (void)

{

        uint8\_t ctrl = 0x00;

**ctrl = Accel\_IO\_Read (0xD6,0x0F);**

        return ctrl;

}

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

Also insert the lines in the sensor initialization function

        HAL\_Delay (1000);

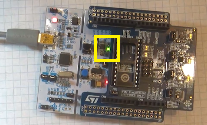
**if (Accel\_ReadID () == 0x68) LD2\_ON;**

**else Error ();**

        AccInit (ctrl);

We will collect the code and we will sew the controller and check that the sensor is the same, comparing it with the reference identifier and including the only green LED. But since the light-emitting diode was already lit at the start-up stage of the program, it is important for us that it does not go out.

If the LED at us remained to burn, means at us it's OK and it is possible to continue to continue the code of initialization of an accelerometer



Add a variable to the function AccInit

void AccInit (uint16\_t InitStruct)

{

**uint8\_t value = 0;**

Let's add some macros to the header file lsm6ds0.h, copying them from a previously prepared file

#define LD2\_OFF HAL\_GPIO\_WritePin (GPIOA, GPIO\_PIN\_5, GPIO\_PIN\_RESET)

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

**#define LSM6DS0\_ACC\_GYRO\_BDU\_DISABLE 0x00**

**#define LSM6DS0\_ACC\_GYRO\_BDU\_ENABLE 0x40**

**#define LSM6DS0\_ACC\_GYRO\_BDU\_MASK 0x40**

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

**#define LSM6DS0\_ACC\_GYRO\_CTRL\_REG5\_XL 0X1F**

**#define LSM6DS0\_ACC\_GYRO\_CTRL\_REG6\_XL 0X20**

**#define LSM6DS0\_ACC\_GYRO\_CTRL\_REG8 0X22**

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

**#define LSM6DS0\_ACC\_GYRO\_ODR\_XL\_POWER\_DOWN 0x00**

**#define LSM6DS0\_ACC\_GYRO\_ODR\_XL\_10Hz 0x20**

**#define LSM6DS0\_ACC\_GYRO\_ODR\_XL\_50Hz 0x40**

**#define LSM6DS0\_ACC\_GYRO\_ODR\_XL\_119Hz 0x60**

**#define LSM6DS0\_ACC\_GYRO\_ODR\_XL\_238Hz 0x80**

**#define LSM6DS0\_ACC\_GYRO\_ODR\_XL\_476Hz 0xA0**

**#define LSM6DS0\_ACC\_GYRO\_ODR\_XL\_952Hz 0xC0**

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

**#define LSM6DS0\_ACC\_GYRO\_ODR\_XL\_MASK 0xE0**

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

**#define LSM6DS0\_ACC\_GYRO\_FS\_XL\_2g 0x00**

**#define LSM6DS0\_ACC\_GYRO\_FS\_XL\_16g 0x08**

**#define LSM6DS0\_ACC\_GYRO\_FS\_XL\_4g 0x10**

**#define LSM6DS0\_ACC\_GYRO\_FS\_XL\_8g 0x18**

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

**#define LSM6DS0\_ACC\_GYRO\_FS\_XL\_MASK 0x18**

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

**#define LSM6DS0\_ACC\_GYRO\_XEN\_XL\_ENABLE 0x08**

**#define LSM6DS0\_ACC\_GYRO\_YEN\_XL\_ENABLE 0x10**

**#define LSM6DS0\_ACC\_GYRO\_ZEN\_XL\_ENABLE 0x20**

**#define LSM6DS0\_ACC\_GYRO\_XEN\_XL\_MASK 0x08**

**#define LSM6DS0\_ACC\_GYRO\_YEN\_XL\_MASK 0x10**

**#define LSM6DS0\_ACC\_GYRO\_ZEN\_XL\_MASK 0x20**

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

**#define LSM6DS0\_ACC\_GYRO\_OUT\_X\_L\_XL 0X28**

**#define LSM6DS0\_ACC\_GYRO\_OUT\_X\_H\_XL 0X29**

**#define LSM6DS0\_ACC\_GYRO\_OUT\_Y\_L\_XL 0X2A**

**#define LSM6DS0\_ACC\_GYRO\_OUT\_Y\_H\_XL 0X2B**

**#define LSM6DS0\_ACC\_GYRO\_OUT\_Z\_L\_XL 0X2C**

**#define LSM6DS0\_ACC\_GYRO\_OUT\_Z\_H\_XL 0X2D**

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

void Accel\_Ini (void);

What these settings mean and registers, we will disassemble as we write the functions.

In the [**next part of the**](http://narodstream.ru/stm-urok-42-podklyuchaem-akselerometr-lsm6ds0-chast-2/) lesson, we will continue to write the sensor initialization function, then write the data reading function, and, moreover, in this part we finish the work with this accelerometer.   
We will see the readings both in plain text and visually using the NS Port Monitor program.

**Lesson 42**

**Part 2**

# ****We connect the accelerometer LSM6DS0****

In the [**previous part of**](http://narodstream.ru/stm-urok-42-podklyuchaem-akselerometr-lsm6ds0-chast-1/) our lesson we got acquainted with the documentation for the sensor, created a project for it, added some macros and set up library files, also wrote the function of reading the accelerometer identifier and started writing the sensor initialization function.

Continue to write the code in the function AccInit

uint8\_t value = 0;

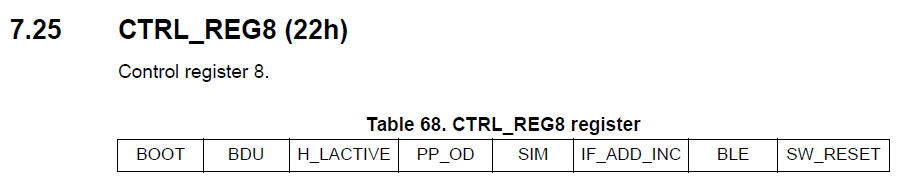
**// set the BDU bit**

**value = Accel\_IO\_Read (0xD6, LSM6DS0\_ACC\_GYRO\_CTRL\_REG8);**

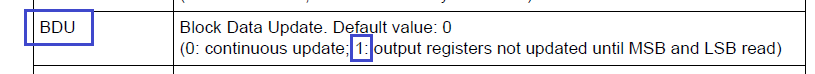
**value & = ~ LSM6DS0\_ACC\_GYRO\_BDU\_MASK;**

**value | = LSM6DS0\_ACC\_GYRO\_BDU\_ENABLE;**

**Accel\_IO\_Write (0xD6, LSM6DS0\_ACC\_GYRO\_CTRL\_REG8, value); // while we turn off the sensor (ODR\_XL = 000)**



In this register, we turn on bit 6 - BDU, which is responsible for protecting the high-order data byte from the change in case if the younger



Next, write the following code

Accel\_IO\_Write (0xD6, LSM6DS0\_ACC\_GYRO\_CTRL\_REG8, value);

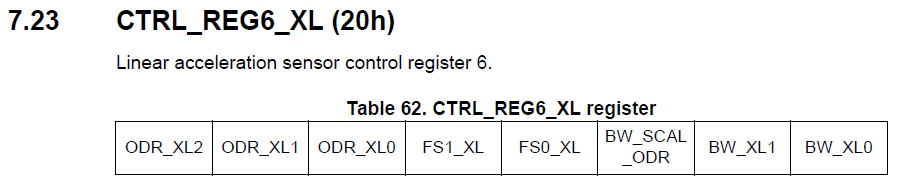
**// while we turn off the sensor (ODR\_XL = 000)**

**value = Accel\_IO\_Read (0xD6, LSM6DS0\_ACC\_GYRO\_CTRL\_REG6\_XL);**

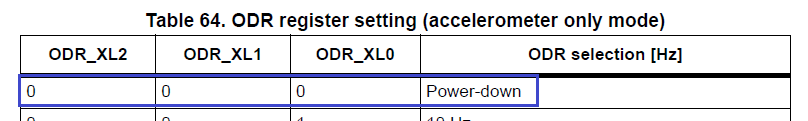
**value & = ~ LSM6DS0\_ACC\_GYRO\_ODR\_XL\_MASK;**

**value | = LSM6DS0\_ACC\_GYRO\_ODR\_XL\_POWER\_DOWN;**

**Accel\_IO\_Write (0xD6, LSM6DS0\_ACC\_GYRO\_CTRL\_REG6\_XL, value);**



In this register, we first turn on the low power mode.



We continue to work with control bits further

Accel\_IO\_Write (0xD6, LSM6DS0\_ACC\_GYRO\_CTRL\_REG6\_XL, value);

**// Full scale selection 2G**

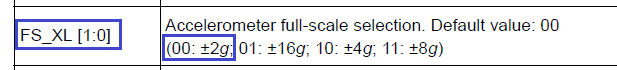
**value = Accel\_IO\_Read (0xD6, LSM6DS0\_ACC\_GYRO\_CTRL\_REG6\_XL);**

**value & = ~ LSM6DS0\_ACC\_GYRO\_FS\_XL\_MASK;**

**value | = LSM6DS0\_ACC\_GYRO\_FS\_XL\_2g;**

**Accel\_IO\_Write (0xD6, LSM6DS0\_ACC\_GYRO\_CTRL\_REG6\_XL, value);**

We work with the same register 6, only we adjust the bits that are responsible for the maximum acceleration. We expose, as usual 2g. For this we do not need to include 3 and 4 bits of the register.



Continue on

Accel\_IO\_Write (0xD6, LSM6DS0\_ACC\_GYRO\_CTRL\_REG6\_XL, value);

**// Turn on the axes**

**value = Accel\_IO\_Read (0xD6, LSM6DS0\_ACC\_GYRO\_CTRL\_REG5\_XL);**

**value & = ~ (LSM6DS0\_ACC\_GYRO\_XEN\_XL\_MASK |**

**LSM6DS0\_ACC\_GYRO\_YEN\_XL\_MASK |**

**LSM6DS0\_ACC\_GYRO\_ZEN\_XL\_MASK);**

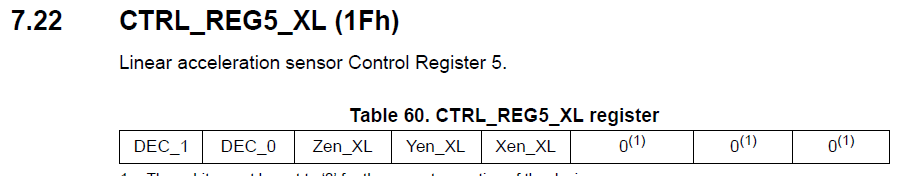
**value | = (LSM6DS0\_ACC\_GYRO\_XEN\_XL\_ENABLE |**

**LSM6DS0\_ACC\_GYRO\_YEN\_XL\_ENABLE |**

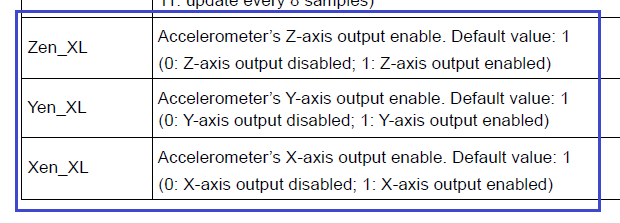
**LSM6DS0\_ACC\_GYRO\_ZEN\_XL\_ENABLE);**

**Accel\_IO\_Write (0xD6, LSM6DS0\_ACC\_GYRO\_CTRL\_REG5\_XL, value);**

We are now working with register 5



Here we will include the use of all three axes of the accelerometer



Continue to write the initialization function

Accel\_IO\_Write (0xD6, LSM6DS0\_ACC\_GYRO\_CTRL\_REG5\_XL, value);

**// Enable Data Rate 119 Hz**

**value = Accel\_IO\_Read (0xD6, LSM6DS0\_ACC\_GYRO\_CTRL\_REG6\_XL);**

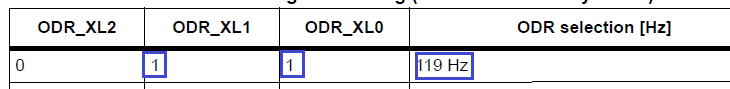
**value & = ~ LSM6DS0\_ACC\_GYRO\_ODR\_XL\_MASK;**

**value | = LSM6DS0\_ACC\_GYRO\_ODR\_XL\_119Hz;**

**Accel\_IO\_Write (0xD6, LSM6DS0\_ACC\_GYRO\_CTRL\_REG6\_XL, value);**

}

Here we go back to register 6 and turn on the speed (measurement frequency) of 119 Hz



The initialization function is complete. Now, in the main function, before exiting the AccInit function, we extinguish the LED, and after the call, we'll light it to make sure that we have nothing nowhere.

        else Error ();

**LD2\_OFF;**

        AccInit (ctrl);

**LD2\_ON;**

We will collect the code, we will sew the MK and see the result.

Now we will write the code into the data reading function from the Accel\_GetXYZ sensor

        uint8\_t i = 0;

**buffer [0] = Accel\_IO\_Read (0xD6, LSM6DS0\_ACC\_GYRO\_OUT\_X\_L\_XL);**

**buffer [1] = Accel\_IO\_Read (0xD6, LSM6DS0\_ACC\_GYRO\_OUT\_X\_H\_XL);**

**buffer [2] = Accel\_IO\_Read (0xD6, LSM6DS0\_ACC\_GYRO\_OUT\_Y\_L\_XL);**

**buffer [3] = Accel\_IO\_Read (0xD6, LSM6DS0\_ACC\_GYRO\_OUT\_Y\_H\_XL);**

**buffer [4] = Accel\_IO\_Read (0xD6, LSM6DS0\_ACC\_GYRO\_OUT\_Z\_L\_XL);**

**buffer [5] = Accel\_IO\_Read (0xD6, LSM6DS0\_ACC\_GYRO\_OUT\_Z\_H\_XL);**

Here we read data from the registers 0X28 - 0X2D, which are responsible for storing the read accelerations from the axes, respectively, of the lower and upper bytes.

Then in a cycle we will enter the bytes already by two-byte values ​​in the appropriate buffer

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

{

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

}

Let's move on to the Accel\_ReadAcc function, which we then call in an infinite loop in main (). Let us comment for the time being the lines associated with the USART and light the LED in case of exceeding the value along the X axis

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

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

// 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) (zval >> 8);

// buf2 [7] = (uint8\_t) zval;

// buf2 [8] = 0x13;

// HAL\_UART\_Transmit (& huart2, buf2,9,0 × 1000);

        if (xval> 1500)

        {

**LD2\_ON;**

        }

**else**

**{**

**LD2\_OFF;**

**}**

        HAL\_Delay (20);

Call this function in an infinite loop

  / \* USER CODE BEGIN 3 \* /

**Accel\_ReadAcc ();**

  }

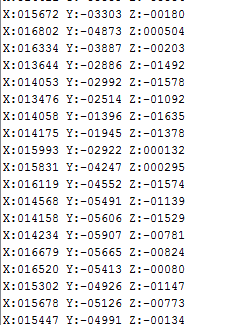
We will collect the code, we will tell the controller and check the operation of the program. With a certain deviation of the board, the LED should light up and go out.

If everything works, then we will try to read the exact data through the terminal program, uncomment the code in the Accel\_ReadAcc function, and also correct it a bit, since we have DMA connected.

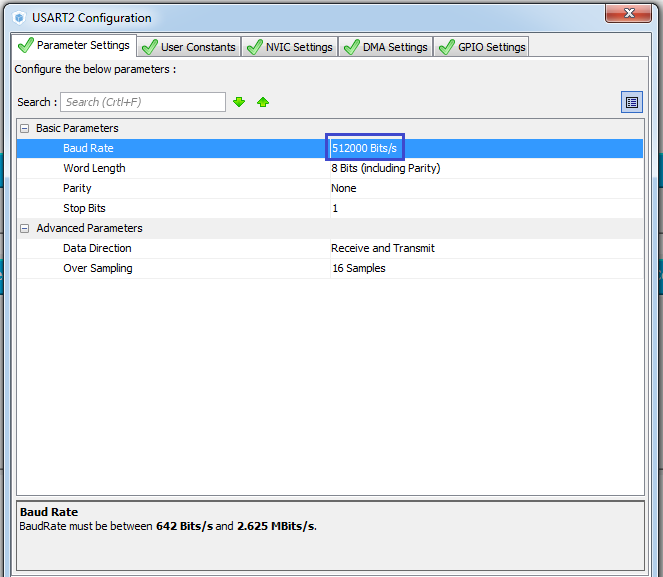
sprintf (str1, "X:% 06d Y:% 06d Z:% 06drn", xval, yval, zval);

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

Let's collect the code, we'll sew the controller, start the port monitor from the arduino-nightly program and see the result



We go into the Cube MX and in the USART settings instead of 256000 we will enable 512000.



We regenerate the project, collect it and make some more changes to the code of the Accel\_ReadAcc function, again commenting out the output code to the terminal. Uncomment and make changes to the output code in the visualization program. The program will use NS Port Monitor, so the code will be compiled in accordance with the protocol requirements of this program (first bytes 0x11 and 0x55).

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

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

**buf2 [0] = 0x11;**

**buf2 [1] = 0x55;**

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

**buf2 [3] = (uint8\_t) xval;**

**buf2 [4] = (uint8\_t) (yval >> 8);**

**buf2 [5] = (uint8\_t) yval;**

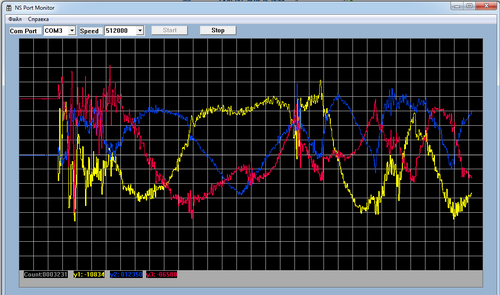
**buf2 [6] = (uint8\_t) (zval >> 8);**

**buf2 [7] = (uint8\_t) zval;**

**HAL\_UART\_Transmit\_DMA (& huart2, buf2.8);**

        if (xval> 1500)

We will collect the project, we will sew the controller, run the program and check the result in practice (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2016/11/image12_1103.png)