**Lesson 55**

**Part 1**

**Humidity sensor HTS221**

Today we will get acquainted with a sensor that measures the relative humidity.

Relative humidity is the ratio of the partial pressure of water vapor in the gas (primarily in air) to the equilibrium pressure of saturated vapor at a given temperature. Therefore, accordingly, in this sensor there is also a temperature sensor.

This sensor is installed on the expansion board **X-NUCLEO-IKS01A1**, designed to work with the Nucleo debug card. We will connect this evaluation board to the Nucleo STM32F401RE board. This sensor is also implemented using MEMS technology.

        This humidity sensor can also be connected to the I2C interface using the SPI interface. But we will use the connection on the usual I2C.

The sensor has the following specifications:

        Range of indication of relative humidity 0 - 100%;

        Range of temperature readings -40 - +120 ° С

        Resolution - 16 bits;

The pressure sensitivity is 0.004% rH / LSB or 256 LSB /% rH.

The temperature sensitivity is 0.016 ° C / LSB or 64 LSB / ° C.

The accuracy of the pressure readings is ± 3.5% rH at 20 - 80% rH and ± 5% rH at 0-100% rH.

The accuracy of the temperature readings is ± 0.5 ° C at 15 - 40 ° C and ± 1 ° C at 0 - 60 ° C

The root-mean-square value of the noise level (RMS noise) is 0.03 by pressure and 0.07 by temperature;

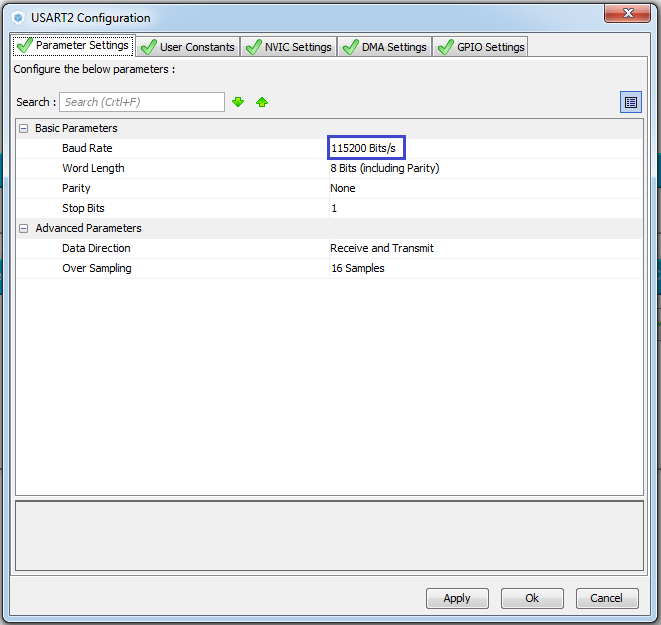
The measurement frequency is 1 - 12.5 Hz.

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

We will create the project from the project of the last lesson, in which we worked with a pressure sensor installed on the same expansion board - from the project Press\_ LPS25HB, only we call this project now, respectively, Humidity\_HTS221.

The files lps25hb.c and lps25hb.h are respectively renamed hts221.c and hts221.h.

Run the Cube MX project. Make sure that the USART speed is set to 115200 bps



We do not touch anything else. Generate the project, open it. Set up the programmer for auto-cutting and turn off the optimization. Add the file hts221.c. We will compile the project.

Due to the renaming of the files, we will have errors. To eliminate these errors, we will fix the connection of the header files to main.c and hts221.c

#include "stm32f4xx\_hal.h"

**#include "hts221.h"**

**#include "hts221.h"**

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

In the timer interrupt handler, we comment out the call to the data reading function and send it to USART.

In the endless cycle, he has already commented on us

**// Press\_Read ();**

        HAL\_UART\_Receive\_IT (& huart2, (uint8\_t \*) str, 8);

In the file hts221.h, delete all the code and copy it there to save valuable time in advance the prepared code with all the macros and variables from the file macro.txt

**#ifndef HTS221\_H\_**

**#define HTS221\_H\_**

**#include "stm32f4xx\_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)**

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

**#define HTS221\_ADDRESS 0xBE**

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

**#define HTS221\_WHO\_AM\_I\_REG 0x0F**

**#define HTS221\_CTRL\_REG1 0x20**

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

**#define HTS221\_WHO\_AM\_I\_VAL 0xBC**

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

**#define HTS221\_PD\_ACTIVE\_MODE 0x80**

**#define HTS221\_PD\_POWERDOWN\_MODE 0x00**

**#define HTS221\_PD\_MASK 0x80**

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

**#define HTS221\_ODR\_ONE\_SHOT 0x00**

**#define HTS221\_ODR\_1HZ 0x01**

**#define HTS221\_ODR\_7HZ 0x02**

**#define HTS221\_ODR\_12\_5HZ 0x03**

**#define HTS221\_ODR\_MASK 0x03**

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

**#define LPS25HB\_DIFF\_EN\_ENABLE 0x08**

**#define LPS25HB\_DIFF\_EN\_DISABLE 0x00**

**#define LPS25HB\_DIFF\_EN\_MASK 0x08**

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

**#define HTS221\_BDU\_DISABLE 0x00**

**#define HTS221\_BDU\_ENABLE 0x04**

**#define HTS221\_BDU\_MASK 0x04**

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

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

**#define HTS221\_HR\_OUT\_L\_REG 0x28**

**#define HTS221\_HR\_OUT\_H\_REG 0x29**

**#define HTS221\_H0\_RH\_X2 0x30**

**#define HTS221\_H1\_RH\_X2 0x31**

**#define HTS221\_T0\_DEGC\_X8 0x32**

**#define HTS221\_T1\_DEGC\_X8 0x33**

**#define HTS221\_T0\_T1\_DEGC\_H2 0x35**

**#define HTS221\_H0\_T0\_OUT\_L 0x36**

**#define HTS221\_H0\_T0\_OUT\_H 0x37**

**#define HTS221\_H1\_T0\_OUT\_L 0x3A**

**#define HTS221\_H1\_T0\_OUT\_H 0x3B**

**#define HTS221\_T0\_OUT\_L 0x3C**

**#define HTS221\_T0\_OUT\_H 0x3D**

**#define HTS221\_T1\_OUT\_L 0x3E**

**#define HTS221\_T1\_OUT\_H 0x3F**

**#define HTS221\_TEMP\_OUT\_L\_REG 0x2A**

**#define HTS221\_TEMP\_OUT\_H\_REG 0x2B**

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

**void Humidity\_Ini (void);**

**void Humidity\_Read (void);**

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

**#endif / \* HTS221\_H\_ \* /**

The function Press\_Ini is renamed to Humidity\_Ini in main.c and hts221.c files

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

void **Humidity**\_Ini (void)

{

**Humidity**\_Ini ();

  / \* USER CODE END 2 \* /

Also correct the function Press\_Read in Humidity\_Read () in main.c and hts221.c files, in main.c it should be commented out

void **Humidity**\_Read (void)

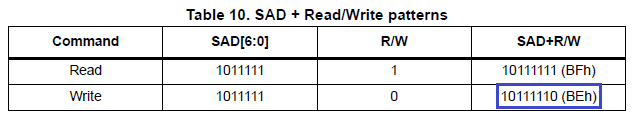
{

/ \* USER CODE BEGIN 3 \* /

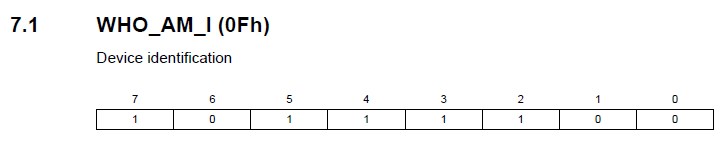
  // **Humidity**\_Read ();

The next task for us is to write the sensor initialization code. And, as always, according to the established tradition, since we have the connection on the I2C bus, we start it, of course, from the chip ID reading.

Based on the connection of the chip, the address we select is **0xBE**



The register for reading the identifier is used by **WHO\_AM\_I (0Fh)**



In this table, we also see by translating the binary code into hexadecimal, that the identifier should be exactly **0xBC**.

The function Press\_ReadID is renamed in Humidity\_ReadID both in the implementation and in the call

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

uint8\_t **Humidity**\_ReadID (void)

{

        HAL\_Delay (1000);

        if ( **Humidity**\_ReadID () == LPS25HB\_WHO\_AM\_I) LD2\_ON;

Similarly, we will do the same with the functions Press\_IO\_Read and Press\_IO\_Write

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

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

{

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

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

{

Also we will correct the code in the function of reading the identifier

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

uint8\_t **Press**\_ReadID (void)

{

        uint8\_t ctrl = 0x00;

        ctrl = **Humidity**\_IO\_Read ( **HTS221\_ADDRESS, HTS221\_WHO\_AM\_I\_REG**);

        return ctrl;

}

Let's fix the code in the main initialization function

        HAL\_Delay (1000);

        if ( **Humidity**\_ReadID () == **HTS221\_WHO\_AM\_I\_VAL**) LD2\_ON;

In functions that we do not use yet, we will comment out all the code, wrapping it here in such tags / \* \* /.

Also comment in the initialization is this

        else Error ();

**// LD2\_OFF;**

**// PressInit (ctrl);**

**// LD2\_ON;**

We compile the code, we will write the controller and check the result of our work. The green LED should light up.

If everything is fine, then we continue initialization.

Rename the function PressInit in HumidityInit.

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

void **Humidity**Init (uint16\_t InitStruct)

{

Also, we will correct and uncomment the call of this function in the main sensor initialization function

        LD2\_OFF;

**Humidity**Init (ctrl);

        LD2\_ON;

We continue to write the initialization, gradually uncommenting and correcting the code in the function

        uint8\_t value = 0;

// while we turn off the sensor (PD = 0)

        value = **Humidity**\_IO\_Read ( **HTS221\_ADDRESS, HTS221\_CTRL\_REG1**);

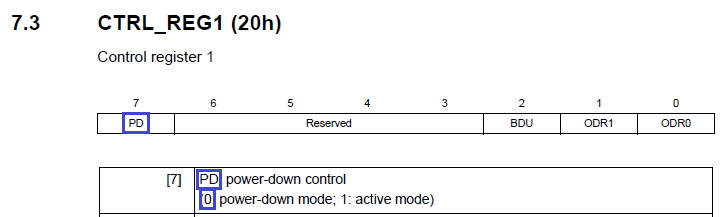
        value & = ~ **HTS221\_PD\_MASK**;

        value | = **HTS221\_PD\_POWERDOWN\_MODE**;

        Humidity\_IO\_Write ( **HTS221\_ADDRESS, HTS221\_CTRL\_REG1**, value);

        / \*

Here we use the **CTRL\_REG1**register **(20h)** and reset the PD bit there to stop the sensor



Write the code further

// Enable BDU

        value = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_CTRL\_REG1);

        value & = ~ **HTS221\_BDU\_MASK**;

        value | = **HTS221\_BDU\_ENABLE**;

        Humidity\_IO\_Write (HTS221\_ADDRESS, HTS221\_CTRL\_REG1, value);

We have the same register **CTRL\_REG1 (20h).**We now set the bit in it, which is responsible for enabling BDU (Block data update). With this regime, we are already familiar with you

image03

Go ahead

// Enable Data Rate 12.5 Hz

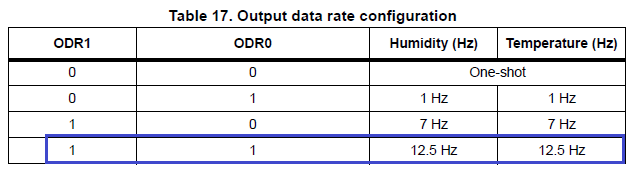
        value = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_CTRL\_REG1);

        value & = ~ **HTS221\_ODR\_MASK**;

        value | = **HTS221\_ODR\_12\_5HZ**;

        Humidity\_IO\_Write (HTS221\_ADDRESS, HTS221\_CTRL\_REG1, value);

Here we use the same register **CTRL\_REG1 (20h)** and set the bits responsible for the reading frequency, which we set the maximum



And the last:

// Now turn on the sensor (PD = 1)

        value = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_CTRL\_REG1);

        value & = ~ **HTS221\_PD\_MASK**;

        value | = **HTS221\_PD\_ACTIVE\_MODE**;

        Humidity\_IO\_Write (HTS221\_ADDRESS, HTS221\_CTRL\_REG1, value);

Here, we transfer the sensor to the active mode by bit 1 of the register, that is, turn it on

image05

This completes the initialization.

We will collect the code, we will sew the controller and check if the LED is off.

If everything is fine, then we'll start writing the code for reading the readings from the sensor

First, we write the function of reading the temperature readings

The function Press\_Get\_Temp is renamed to Humidity\_Get\_Temp, also we add some variables and move the comment below

void **Humidity**\_Get\_Temp (float \* pData)

{

        uint8\_t btnstat;

**int16\_t T0\_degC, T1\_degC;**

**int16\_t T0\_out, T1\_out, T\_out, T0\_degC\_x8\_u16, T1\_degC\_x8\_u16;**

**uint8\_t**buffer **[4], tmp;**

        / \*

Also we will correct the name of the function and in its call in the function Humidity\_Read

**Humidity**\_Get\_Temp (& temper);

Although initialization was simpler compared to the previous sensor, the same can not be said for reading, there will be a heavier one, there are a lot of calibration cells. The format of the lesson does not include calibration, so we will use them only for reading.

We **count the**registers **T0\_degC**\_x8 and **T1\_degC\_x8**

        uint8\_t buffer [4], tmp;

**buffer [0] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_T0\_DEGC\_X8);**

**buffer [1] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_T1\_DEGC\_X8);**

These registers

image08

We make some calculations

        buffer [1] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_T1\_DEGC\_X8);

**tmp = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_T0\_T1\_DEGC\_H2);**

In the [**next part of**](http://narodstream.ru/stm-urok-55-datchik-vlazhnosti-hts221-chast-2/) this lesson we will test the operation of our code and sensor first in the terminal program in text form, and then in the visualization program.

**Lesson 55**

**Part 2**

# ****Humidity sensor HTS221****

In the [**last part of**](http://narodstream.ru/stm-urok-55-datchik-vlazhnosti-hts221-chast-1/) this lesson, we added all the macros that make the reading and writing easier, read the sensor ID, wrote the initialization completely and started writing the reading and processing of the measured values.

Continue to write our function of reading temperature

**tmp = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_T0\_T1\_DEGC\_H2);**

**T0\_degC\_x8\_u16 = (((uint16\_t) (tmp & 0x03)) << 8) | ((uint16\_t) buffer [0]);**

**T1\_degC\_x8\_u16 = (((uint16\_t) (tmp & 0x0C)) << 6) | ((uint16\_t) buffer [1]);**

**T0\_degC = T0\_degC\_x8\_u16 >> 3;**

**T1\_degC = T1\_degC\_x8\_u16 >> 3;**

How to make calculations is defined in the technical documentation

image07

We consider the following quantities

  T1\_degC = T1\_degC\_x8\_u16 >> 3;

**buffer [0] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_T0\_OUT\_L);**

**buffer [1] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_T0\_OUT\_H);**

**buffer [2] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_T1\_OUT\_L);**

**buffer [3] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_T1\_OUT\_H);**

**T0\_out = (((uint16\_t) buffer [1]) << 8) | (uint16\_t) buffer [0];**

**T1\_out = (((uint16\_t) buffer [3]) << 8) | (uint16\_t) buffer [2];**

There are already used other 4 registers

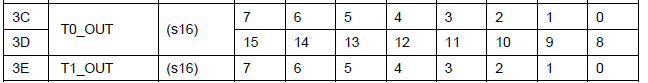


image09

Read the temperature readings of the sensor and put them in the corresponding variable

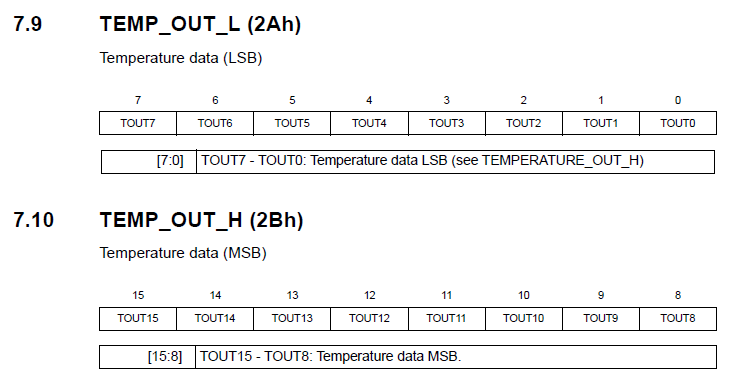
T1\_out = (((uint16\_t) buffer [3]) << 8) | (uint16\_t) buffer [2];

**buffer [0] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_TEMP\_OUT\_L\_REG);**

**buffer [1] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_TEMP\_OUT\_H\_REG);**

**T\_out = (((uint16\_t) buffer [1]) << 8) | (uint16\_t) buffer [0];**

Indications are stored in these registers



Move the beginning of the comment. In the uncommented area, remove the extra and replace **raw\_data** with **T\_out**

T\_out = (((uint16\_t) buffer [1]) << 8) | (uint16\_t) buffer [0];

        btnstat = HAL\_GPIO\_ReadPin (GPIOC, GPIO\_PIN\_13);

        // If the button is not pressed,

        // then call the moving average filter

        if (btnstat! = 0)         **T\_out** = MovingAverageTemp ( **T\_out**);

        / \*

Well, and remove all the comments from the function, since it's time to use the calculation from the technical documentation given above

        if (btnstat! = 0) T\_out = MovingAverageTemp (T\_out);

**\* pData = (float) (T\_out - T0\_out) \* (float) (T1\_degC - T0\_degC) /**

**(float) (T1\_out - T0\_out) + T0\_degC;**

}

In fact, there are ordinary numbers in the calibration cells, they just "can" remember in the cells (registers) after making changes and, consequently, the calibrated sensor is no longer dependent on the code.Previously, we calibrated other sensors in the code, and if we connected this sensor to another controller, then, writing the code for it, we would have to re-calibrate. And, if you look into the calculations given above, you can easily figure out what kind of data to make there to calibrate the readings. Because the error in the readings is not only in the shift of the readings, but also in the difference in error at different temperatures. And, using these cells, all these errors can be corrected. But we in this occupation will not do this, but let's hope, that the sensor manufacturer has already done this work in good faith. Also we will not do this because we do not have any reference temperature and humidity meters, comparing with the readings of which, we could calibrate the sensor.

This completes the function of reading the temperature reading.

Let's start to measure the relative humidity of the air. What it is, is given at the very beginning of the lesson.

We rename the function Press\_Get\_Press to Humidity\_Get\_Hum, also add some variables and move the comment below

void **Humidity**\_Get\_ **Hum**(float \* pData)

{

        uint8\_t btnstat;

        uint8\_t buffer [ **2**];

**int16\_t H0\_T0\_out, H1\_T0\_out, H\_T\_out;**

**int16\_t H0\_rh, H1\_rh;**

**float tmp\_f;**

        / \*

Also we will correct the name of the function and in its call in the function Humidity\_Read

**Humidity**\_Get\_ **Hum**(& press);

Let's look at the documentation (HTS221 Interpreting humidity and temperature readings), how the relative humidity of air is calculated taking into account the calibration registers

image12

We see that in this formula the temperature readings are also very actively used together with the calibration. But we already know from the definition of the relative humidity of air that this must be so.

We consider the following quantities and enter them into variables

float tmp\_f;

**buffer [0] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_H0\_RH\_X2);**

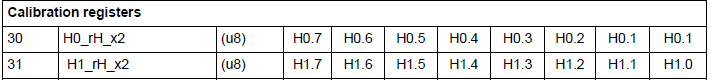
**buffer [1] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_H1\_RH\_X2);**

**H0\_rh = buffer [0] >> 1;**

**H1\_rh = buffer [1] >> 1;**

        / \*

Here we use the following 2 calibration registers



Consider the following 4 calibration registers and enter the data in the appropriate variables

**buffer [0] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_H0\_T0\_OUT\_L);**

**buffer [1] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_H0\_T0\_OUT\_H);**

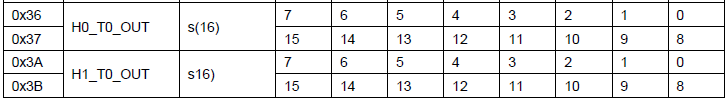
**H0\_T0\_out = (((uint16\_t) buffer [1]) << 8) | (uint16\_t) buffer [0];**

**buffer [0] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_H1\_T0\_OUT\_L);**

**buffer [1] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_H1\_T0\_OUT\_H);**

**H1\_T0\_out = (((uint16\_t) buffer [1]) << 8) | (uint16\_t) buffer [0];**

Let's see in the documentation what these registers are



From the data of the 3rd column we see that these values ​​are signed (s (16)) (s-signed, 16 - 16-bit)

We consider uncalibrated humidity readings

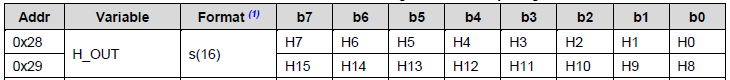
**buffer [0] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_HR\_OUT\_L\_REG);**

**buffer [1] = Humidity\_IO\_Read (HTS221\_ADDRESS, HTS221\_HR\_OUT\_H\_REG);**

**H\_T\_out = (((uint16\_t) buffer [1]) << 8) | (uint16\_t) buffer [0];**

        / \*

Everything is standard here. Two 8-bit registers are collected in one value



Here we also have a 16-bit signed value.

Move the comment and correct the code (removing unnecessary, since we had a 24-bit value in the previous lesson) to use or not to use (depending on the state of the button) the moving average, also change the variable **raw\_data** to **H\_T\_out**

H\_T\_out = (((uint16\_t) buffer [1]) << 8) | (uint16\_t) buffer [0];

        btnstat = HAL\_GPIO\_ReadPin (GPIOC, GPIO\_PIN\_13);

        // If the button is not pressed,

        // then call the moving average filter

        if (btnstat! = 0)         **H\_T\_out** = MovingAverage ( **H\_T\_out**);

        / \*

Now remove all comments and correct the calculation of the relative humidity of air and put it in a variable with a floating point

        if (btnstat! = 0) H\_T\_out = MovingAverage (H\_T\_out);

**tmp\_f = (float) (H\_T\_out - H0\_T0\_out) \* (float) (H1\_rh - H0\_rh) /**

**(float) (H1\_T0\_out - H0\_T0\_out) + H0\_rh;**

**\* pData = (tmp\_f> 100.0f)? 100.0f**

**: (tmp\_f <0.0f)? 0.0f**

**: tmp\_f;**

}

Now we will begin to make changes to the Humidity\_Read function. Let's correct the name of the variable for relative humidity, delete the calibration and transferring the values ​​to other units of measurement, uncomment the code to send symbol values ​​to USART, also correct it, and the code for sending floating point data is still commented out, and also uncomment the delay.

void Humidity\_Read (void)

{

        float temper, **hum**;

        Humidity\_Get\_Temp (& temper);

        Humidity\_Get\_Hum (& **hum**);

        sprintf (str1, "TEMP:% 06f; **HUMIDITY**:% 06f; rn", temper, **hum**);

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

// uint8\_t \* t = (uint8\_t \*) & temper;

// uint8\_t \* p = (uint8\_t \*) & press;

// buf2 [0] = 0x11;

// buf2 [1] = 0x55;

// buf2 [2] = (uint8\_t) (\* (uint32\_t \*) t);

// buf2 [3] = (uint8\_t) ((\* (uint32\_t \*) t) >> 8);

// buf2 [4] = (uint8\_t) ((\* (uint32\_t \*) t) >> 16);

// buf2 [5] = (uint8\_t) ((\* (uint32\_t \*) t) >> 24);

// buf2 [6] = (uint8\_t) (\* (uint32\_t \*) p);

// buf2 [7] = (uint8\_t) ((\* (uint32\_t \*) p) >> 8);

// buf2 [8] = (uint8\_t) ((\* (uint32\_t \*) p) >> 16);

// buf2 [9] = (uint8\_t) ((\* (uint32\_t \*) p) >> 24);

// HAL\_UART\_Transmit (& huart2, buf2,10.0 × 1000);

        HAL\_Delay (200);

}

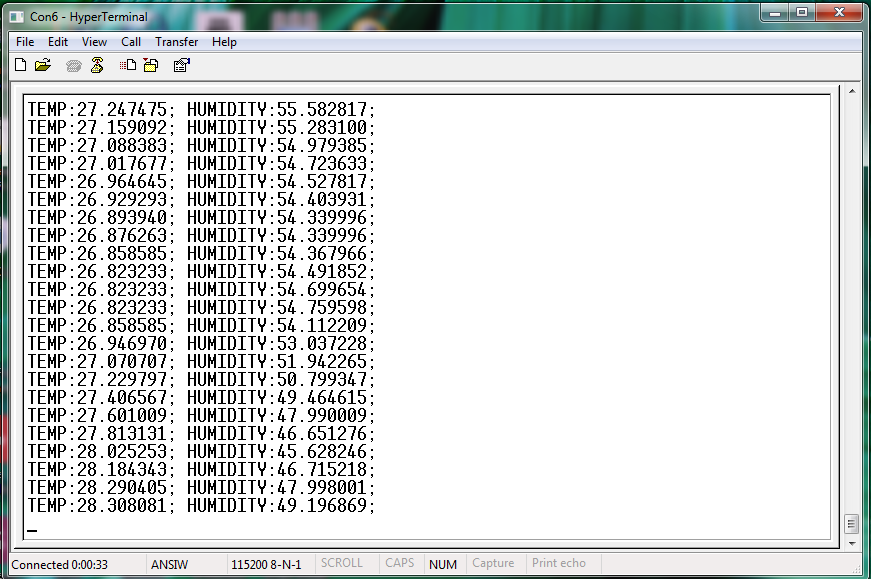
Uncomment the code in an infinite loop in **main ()**

  / \* USER CODE BEGIN 3 \* /

                Humidity\_Read ();

  }

We'll compile the code, we'll run the controller and see the result in the terminal program. In the process of viewing several times we will touch the sensor with your finger for a long time and see the changes in temperature and humidity



Now we will try to see the result without the button and with the button, but already in the visualization program.

Comment in the function of processing and sending data code to send in text form, uncomment the code for sending in the form of floating point values. How to send floating-point values, we studied in detail in the previous lesson. Also comment out the delay and correct some variables

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

        uint8\_t \* t = (uint8\_t \*) & temper;

        uint8\_t \* **h**= (uint8\_t \*) & **hum**;

        buf2 [0] = 0x11;

        buf2 [1] = 0x55;

        buf2 [2] = (uint8\_t) (\* (uint32\_t \*) t);

        buf2 [3] = (uint8\_t) ((\* (uint32\_t \*) t) >> 8);

        buf2 [4] = (uint8\_t) ((\* (uint32\_t \*) t) >> 16);

        buf2 [5] = (uint8\_t) ((\* (uint32\_t \*) t) >> 24);

        buf2 [6] = (uint8\_t) (\* (uint32\_t \*) **h**);

        buf2 [7] = (uint8\_t) ((\* (uint32\_t \*) **h**) >> 8);

        buf2 [8] = (uint8\_t) ((\* (uint32\_t \*) **h**) >> 16);

        buf2 [9] = (uint8\_t) ((\* (uint32\_t \*) **h**) >> 24);

        HAL\_UART\_Transmit (& huart2, buf2,10.0 × 1000);

// HAL\_Delay (200);

Also, in main.c, comment out the function call in an infinite loop and uncomment it in the interrupt-timer function

  / \* USER CODE BEGIN 3 \* /

        // Humidity\_Read ();

  }

  / \* USER CODE END 3 \* /

        if (huart2.RxXferCount == 0)

        {

                Humidity\_Read ();

                HAL\_UART\_Receive\_IT (& huart2, (uint8\_t \*) str, 8);

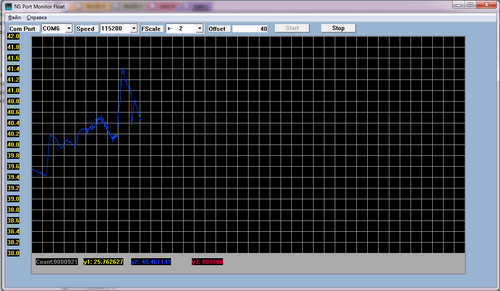
        }

We compile the code, let's go through the controller and see the results of the temperature and humidity readings with the button and without the button in the visualization program, also applying different values ​​of limits and shifting the readings

By measuring the temperature, we bring the finger of the hand to the sensor gradually and also gradually remove it, first without the button then with the button. A small difference, but still there.

Now the pressure. It's unchanged, since it's hard to affect the atmospheric pressure, unless you put the debug card into the car tire and pump it up (joke). Also we will try without the button and with the button, applying other values ​​of the divider and shift in the visualization program

[](http://narodstream.ru/wp-content/uploads/2016/11/image17_1156.png)

[](http://narodstream.ru/wp-content/uploads/2016/11/image18_1154.png)

We are seeing a difference with the use of the moving average filter and without it. Also, the last jump in moisture readings is caused by the fact that I partially immersed the motherboard with the sensor in a glass with a small amount of coffee (it's in the glass, not in the coffee itself, and then you'll think of something else!). It's just that a glass of unfinished coffee fell into his hand.