**HAL. Bus I2C. We continue to work with DS3231**

In the [**last lesson,**](http://narodstream.ru/stm-urok-8-hal-i2c-podklyuchaem-chasy-realnogo-vremeni-ds3231/) we got to know the I2C bus, as well as the DS3231 real-time clock chip, created and configured the project in **Cube MX** and **Keil** .

In this lesson, we will continue to work with the same project and will be directly dealing with the code.

In the file main.h we will also add a variable for the data

#include "RTC.h"

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

**uint8\_t aTxBuffer [8];**

In it, we will take the values ​​of all 7 registers of the chip at once and, having filled in all the cells in it, we will send it there also when we set the time in the clock.

Ecternal it in i2c.c, also write there a variable for the string and write there two functions for working with the bus.

#include "i2c.h"

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

**extern uint8\_t aTxBuffer [8];**

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

**char str [100];**

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

**void I2C\_WriteBuffer (I2C\_HandleTypeDef hi, uint8\_t DEV\_ADDR, uint8\_t sizebuf)**

**{**

**while (HAL\_I2C\_Master\_Transmit (& hi, (uint16\_t) DEV\_ADDR, (uint8\_t \*) & aTxBuffer, (uint16\_t) sizebuf, (uint32\_t) 1000)! = HAL\_OK)**

**{**

**if (HAL\_I2C\_GetError (& hi)! = HAL\_I2C\_ERROR\_AF)**

**{**

**sprintf (str, "Buffer error");**

**LCD\_SetPos (8, 0);**

**LCD\_String (str);**

**}**

**}**

**}**

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

**void I2C\_ReadBuffer (I2C\_HandleTypeDef hi, uint8\_t DEV\_ADDR, uint8\_t sizebuf)**

**{**

**while (HAL\_I2C\_Master\_Receive (& hi, (uint16\_t) DEV\_ADDR, (uint8\_t \*) & aTxBuffer, (uint16\_t) sizebuf, (uint32\_t) 1000)! = HAL\_OK)**

**{**

**if (HAL\_I2C\_GetError (& hi)! = HAL\_I2C\_ERROR\_AF)**

**{**

**sprintf (str, "Buffer error");**

**LCD\_SetPos (8, 0);**

**LCD\_String (str);**

**}**

**}**

**}**

The first function will be for data transfer to the bus, and the second for reception.

Input parameters in the first function:

**I2C\_HandleTypeDef hi** is the **I2C** bus identifier **.**

**DEV\_ADDR** is the address of the device

**sizebuf** - the number of bytes that we will transmit.

Then there is a loop from which we will exit when the data is transferred and we get the status HAL\_OK.

The status will be returned to us by the standard data transfer function in I2C from the HAL\_I2C\_Master\_Transmit HAL library. In this function, we pass almost the same parameters, some of them are only explicitly converted to a different type, and we also pass the timeout in milliseconds. This does not mean that the function will be executed that long. If successful, we will return from the function instantly. But if something goes wrong, we'll wait that long, and then we'll still leave, though with an error.

Then we check the transmission for an error, and in case it happens, we will display the error text on the display.

The second function has absolutely the same input parameters, only there we will already read the bytes and send them to the same buffer.

The body of the function is also similar to the body of the previous function, only the library function has already been applied there HAL\_I2C\_Master\_Receive.

Let's create for these two functions the prototypes in the file **i2c.h**

#include "lcd.h"

**void I2C\_WriteBuffer (I2C\_HandleTypeDef hi, uint8\_t DEV\_ADDR, uint8\_t sizebuf);   
void I2C\_ReadBuffer (I2C\_HandleTypeDef hi, uint8\_t DEV\_ADDR, uint8\_t sizebuf);**

We remove all the code on the display except the initialization from the main function.

We begin to work with the bus in the main function main ()

First, delete the output code for the test lines on the display screen. Leave only this

  / \* USER CODE BEGIN 2 \* /   
**LCD\_ini ();   
 LCD\_Clear ();**  
  / \* USER CODE END 2 \* /

Next we will start working with functions.

We write code in an infinite loop

  while (1)

  {

**LCD\_SetPos (0,1);**

**aTxBuffer [0] = 0;**

**I2C\_WriteBuffer (hi2c1, (uint16\_t) 0xD0,1);**

**while (HAL\_I2C\_GetState (& hi2c1)! = HAL\_I2C\_STATE\_READY)**

**{**

**}**

Here we first set the line position in the display, initialize the buffer, or rather not all, but only its very first cell.

Next we will give the address of the device and the address of the first register of the chip.

Write further in an infinite loop

                while (HAL\_I2C\_GetState (& hi2c1)! = HAL\_I2C\_STATE\_READY)

                {

                }

**I2C\_ReadBuffer (hi2c1, (uint16\_t) 0xD0.7);**

And then we already count all 7 registers in the buffer from the RTC

Since we took the data in binary-coded decimal code, we can not show them normally without conversion. Therefore, we write two conversion functions in the file RTC.c

#include "RTC.h"

**uint8\_t RTC\_ConvertFromDec (uint8\_t c)**

**{**

**uint8\_t ch = ((c >> 4) \* 10 + (0x0F & c));**

**return ch;**

**}**

**uint8\_t RTC\_ConvertFromBinDec (uint8\_t c)**

**{**

**uint8\_t ch = ((c / 10) << 4) | (c% 10);**

**return ch;**

**}**

Also we will assign prototypes to them in the same header.

#ifndef RTC\_H\_

#define RTC\_H\_

#include "stm32f4xx\_hal.h"

**uint8\_t RTC\_ConvertFromDec (uint8\_t c); // translate binary-decimal to decimal**

**uint8\_t RTC\_ConvertFromBinDec (uint8\_t c); // convert the decimal number to binary-decimal**

#endif / \* RTC\_H\_ \* /

Adding Variables

  / \* USER CODE BEGIN 1 \* /

**uint32\_t i = 0;**

**uint8\_t sec = 0, min = 0, hour = 0, day = 0, date = 0, month = 0, year = 0;**

  / \* USER CODE END 1 \* /

Further routine.

                I2C\_ReadBuffer (hi2c1, (uint16\_t) 0xD0.7);

**date = aTxBuffer [4];**

**date = RTC\_ConvertFromDec (date); // Convert to decimal format**

**LCD\_SendChar ((char) ((date / 10)% 10) + 0x30);**

**LCD\_SendChar ((char) (date% 10) + 0x30);**

**LCD\_SendChar (':');**

**month = aTxBuffer [5];**

**month = RTC\_ConvertFromDec (month); // Convert to decimal format**

**LCD\_SendChar ((char) ((month / 10)% 10) + 0x30);**

**LCD\_SendChar ((char) (month% 10) + 0x30);**

**LCD\_SendChar (':');**

**year = aTxBuffer [6];**

**year = RTC\_ConvertFromDec (year); // Convert to decimal format**

**LCD\_SendChar ((char) ((year / 10)% 10) + 0x30);**

**LCD\_SendChar ((char) (year% 10) + 0x30);**

**LCD\_SendChar (':');**

**day = aTxBuffer [3];**

**day = RTC\_ConvertFromDec (day); // Convert to decimal format**

**LCD\_SendChar ((char) (day% 10) + 0x30);**

**LCD\_SendChar (':');**

**hour = aTxBuffer [2];**

**hour = RTC\_ConvertFromDec (hour); // Convert to decimal format**

**LCD\_SendChar ((char) ((hour / 10)% 10) + 0x30);**

**LCD\_SendChar ((char) (hour% 10) + 0x30);**

**LCD\_SendChar (':');**

**min = aTxBuffer [1];**

**min = RTC\_ConvertFromDec (min); // Convert to decimal format**

**LCD\_SendChar ((char) ((min / 10)% 10) + 0x30);**

**LCD\_SendChar ((char) (min% 10) + 0x30);**

**LCD\_SendChar (':');**

**sec = aTxBuffer [0];**

**sec = RTC\_ConvertFromDec (sec); // Convert to decimal format**

**LCD\_SendChar ((char) ((sec / 10)% 10) + 0x30);**

**LCD\_SendChar ((char) (sec% 10) + 0x30);**

**LCD\_SetPos (0,0);**

**LCD\_SendChar ((char) ((i / 100)% 10) + 0x30);**

**LCD\_SendChar ((char) ((i / 10)% 10) + 0x30);**

**LCD\_SendChar ((char) (i% 10) + 0x30);**

**LCD\_SetPos (6,2);**

**LCD\_SendChar ((char) (((i + 500) / 100)% 10) + 0x30);**

**LCD\_SendChar ((char) (((i + 500) / 10)% 10) + 0x30);**

**LCD\_SendChar ((char) ((i + 500)% 10) + 0x30);**

**LCD\_SetPos (9,3);**

**LCD\_SendChar ((char) (((i + 750) / 100)% 10) + 0x30);**

**LCD\_SendChar ((char) (((i + 750) / 10)% 10) + 0x30);**

**LCD\_SendChar ((char) ((i + 750)% 10) + 0x30);**

**HAL\_Delay (100);**

**i ++;**

  / \* USER CODE END WHILE \* /

In this long code, for all the registers in the chip, we first take the first step from the corresponding cell of the buffer into which we have read the registers, first take the readout in a variable, then convert it to a normal decimal form. Then convert tens and one units into symbols according to the ascii table, and then display. Then at the end we apply a delay of 100 milliseconds, then the entire process is repeated again.

The rest of the code from the infinite loop, left from the project for testing and connecting the display, we do not remove.

We will collect the code and let's say the controller. Let's see how all of this works for us.

