**HAL. LCD 20 × 4. 4-bit mode. Displaying information**

Today we will continue to work with the 20 × 4 display, with which we started working on the [**last lesson**](http://narodstream.ru/stm-urok-6-biblioteka-hal-lcd-20x4-4-bitnyj-rezhim/). We wrote the initialization and tested the code in practice.

Now we are trying to bring something to the display.

To do this, we first write a function for writing data to the display, it will be slightly different from the commands (only with the foot **rs**)

**void LCD\_Data (uint8\_t dt)**

**{**

**rs1;**

**LCD\_WriteData (dt >> 4);**

**e1;**

**delay ();**

**e0;**

**LCD\_WriteData (dt);**

**e1;**

**delay ();**

**e0;**

**}**

We'll also write a cleaning function for the display

**void LCD\_Clear (void)**

**{**

**LCD\_SendCommand4 (0x01);**

**HAL\_Delay (2);**

**}**

Also, in fact, write the function of outputting a symbol on the display

**void LCD\_SendChar (char ch)**

**{**

**LCD\_Data ((uint8\_t) ch);**

**delay ();**

**}**

Let's make it visible by writing a prototype in lcd.h.

And we will try finally to deduce a symbol in the main function of the program

        LCD\_ini ();

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

Let's sew and see



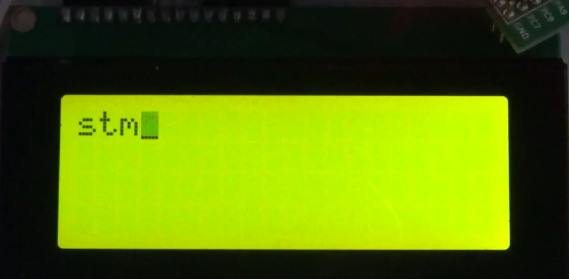
Let's try something else on the screen

        LCD\_SendChar ('s');

**LCD\_SendChar ('t');**

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

Stitch, watch



Everything works, but, you see, it's not very convenient to write a program, using a separate command for each character. Let's therefore write a function to output an entire string

**void LCD\_String (char \* st)**

**{**

**uint8\_t i = 0;**

**while (st [i]! = 0)**

**{**

**LCD\_Data (st [i]);**

**delay ();**

**i ++;**

**}**

**}**

We will write a prototype for this function and call it from the main module, trying to display a line on the display.

First, declare the array and initialize it

int main (void)

{

  / \* USER CODE BEGIN 1 \* /

**char str [] = "Stm32F407VG";**

  / \* USER CODE END 1 \* /

Then we call the function, and by the character output for the moment we comment

        LCD\_ini ();

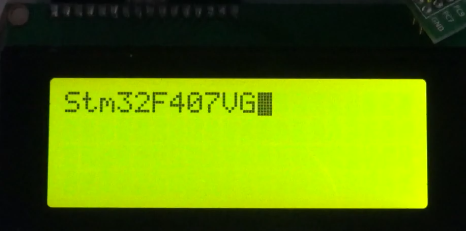
**LCD\_String (str);**

// LCD\_SendChar ('s');

// LCD\_SendChar ('t');

// LCD\_SendChar ('m');

We will collect the code, we will tell the controller and we will see the result



Our entire line is visible on the display.

Also, we are not interested in writing in a row, so we need to create a function for output to the display in the desired place.

For this, let's look at the datasheet (by opening a specialized for LCD2004)

We see there (on page 8) that the lines have their addresses in DDRAM memory and all the characters also:

1 line - 0x00

2nd line - 0x40

3 line - 0x14

4 line - 0x54

And each character is an offset relative to the beginning of the line.

Having understood this, we write the positioning function by column and line

**void LCD\_SetPos (uint8\_t x, uint8\_t y)**

**{**

**switch (y)**

**{**

**case 0:**

**LCD\_Command (x | 0x80);**

**HAL\_Delay (1);**

**break;**

**case 1:**

**LCD\_Command ((0x40 + x) | 0x80);**

**HAL\_Delay (1);**

**break;**

**case 2:**

**LCD\_Command ((0x14 + x) | 0x80);**

**HAL\_Delay (1);**

**break;**

**case 3:**

**LCD\_Command ((0x54 + x) | 0x80);**

**HAL\_Delay (1);**

**break;**

**}**

**}**

Do not forget about the prototype.

Let's try to write something to another position.

Let's change the code in main.c as follows. First, declare a different (without initialization) string array

  / \* USER CODE BEGIN 1 \* /

**char str [100];**

  / \* USER CODE END 1 \* /

Then change the output code

LCD\_ini ();

**sprintf (str, "Stm32F407VG");**

LCD\_String (str);

**LCD\_SetPos (10, 2);**

**sprintf (str, "ARM mc");**

**LCD\_String (str);**

Also our display should be able to be completely cleaned. To do this, we write the purge function.

**void LCD\_Clear (void)**

**{**

**LCD\_Command (0x01);**

**HAL\_Delay (2);**

**}**

It will also need a prototype.

Let's try after some delay to clear our screen

sprintf (str, "ARM mc");

LCD\_String (str);

**HAL\_Delay (2000);**

**LCD\_Clear ();**

We will sew and look. In the video version attached at the bottom of the lesson, it will be clearer, so I do not show it here.

We add a variable to main () and at last a bit poprikalyvayemsya

  / \* USER CODE BEGIN 1 \* /

**uint32\_t i = 0;**

        char str [100];

  / \* USER CODE END 1 \* /

Add the code

  sprintf (str, "ARM mc");

        LCD\_String (str);

        HAL\_Delay (2000);

        LCD\_Clear ();

**LCD\_SetPos (4.0);**

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

**LCD\_SetPos (8.1);**

**LCD\_SendChar ('t');**

**LCD\_SetPos (12.2);**

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

**LCD\_SetPos (16,3);**

**LCD\_SendChar ('3');**

**LCD\_SendChar ('2');**

**HAL\_Delay (2000);**

  / \* USER CODE END 2 \* /

Also add code to the infinite loop

  / \* USER CODE BEGIN WHILE \* /

  while (1)

  {

**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 (16,1);**

**LCD\_SendChar ((char) (((i + 250) / 100)% 10) + 0x30);**

**LCD\_SendChar ((char) (((i + 250) / 10)% 10) + 0x30);**

**LCD\_SendChar ((char) ((i + 250)% 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 \* /

Well, it's absolutely on the stick to remove the annoying black flasher (or cursor). To do this, we change only one letter in the initialization

        LCD\_Command (0x28); // 4 bit mode, 2 lines (for our large display it's 4 lines), font 5х8

        HAL\_Delay (1);

        LCD\_Command (0x28); // again for fidelity

        HAL\_Delay (1);

        LCD\_Command (0x0 **C**); // display is turned on (D = 1), **no cursors are needed**

        HAL\_Delay (1);

        LCD\_Command (0x01); // remove the garbage

        HAL\_Delay (2);

        LCD\_Command (0x06); // write to the left.

        HAL\_Delay (1);

        LCD\_Command (0x02); // return the cursor to the zero position

That's what we got (respectively, in a video version look better)

