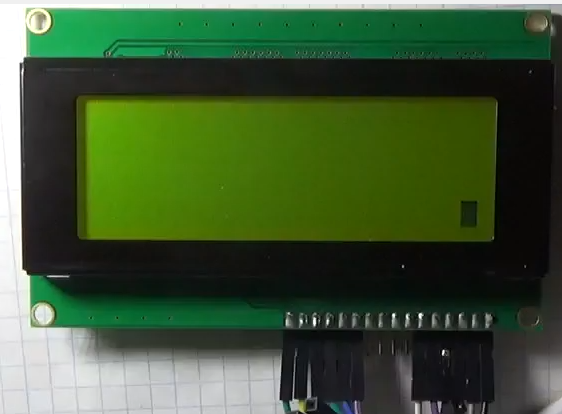
# HAL Library. LCD 20 × 4. 4-bit mode

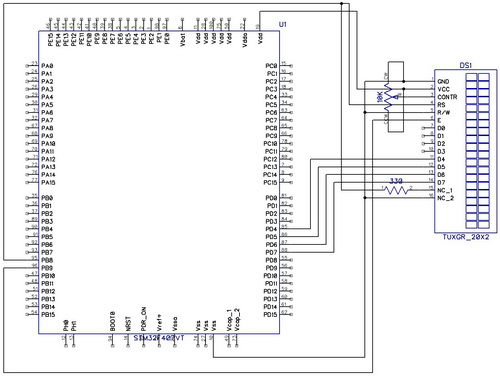
Today we start working with the display connection. Since the LEDs - it's very beautiful, but you still need to somehow look at the information.

We will connect the display to the controller  **HD44780** , although the data in the datasheet is **S6A0069** , which is also the analog of the first one. The display displays character information with a dimension of 4 lines of 20 characters. And we will connect this display 4-bit way. So we can save 4 port legs. In this mode, we transfer the data by half the byte, first the higher part, then the younger one.

This display looks like this:



The display to the microcontroller **STM32F407VG**  we connect here this way (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2016/12/Image01_1472.jpg)

The diagram shows an indicator 20x2, but the meaning is the same, just in the program there was no 20x4 display.

By connecting the display and working with it in 4-bit mode, you can look in detail in the [lesson on AVR](http://narodstream.ru/avr-urok-12-lcd-indikator-16x2/), in which we examined the display 16x2.

But all the same, I must remind you that GND and VCC can often change places depending on the type of display, so look at the documentation on your display, otherwise it will inevitably fail.

In principle, we have already dismantled the purpose of the 1 and 2 legs of the display module.

Assignment of the following module feet:

3 - V0 is the foot with which the contrast of the display is adjusted. That is, the contrast of the display will depend on the voltage applied to the given leg. As a rule, a variable resistor is used for 10 kilos, connected by the extreme legs to the common wire and to the power supply, and from the central leg of this resistor the wire goes just to the foot V0 and by adjusting the resistor we adjust the contrast of the display in the module.

4 - RS - this is such a cunning leg, with which the display controller will "know" which data is on the data bus. If we give a logical 0 to the given leg, then there will be a command, if 1 - then this data.

5 - RW - this leg, depending on the logical state on it, tells the display controller, we will read it from him or we will write data to it. If there is 0 - then we will write to the display controller, and if 1 - we will read the data from the display controller. This function is rarely used. As a rule, we always just write the data into the display. Reading is usually required in order to determine that the display has accepted our data, or to determine the state. But there are certain timings that allow us to "believe" the display kettle, that it has received and processed our data. We can also read from the memory of the display, which, in principle, there is no need. Therefore, we usually connect this contact to a common wire.

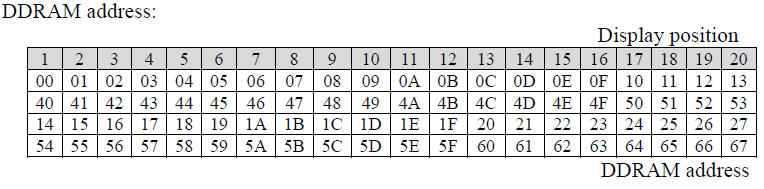
6-E is the so-called gating bus, on the falling edge (when 1 changes to 0) at which the display controller understands that it is now the time of data reading on the data legs D0 to D7, or the data transmission from the module, depending also on the state legs RW.

The legs D0 to D7 are a parallel 8-bit data bus through which data is transmitted or received. Numbers 0 - 7 correspond to the same name bits in the data byte. But there is also a 4-bit way to transfer data to the controller and from the display controller when only the D4-D7 data feet are used, and the D0-D3 feet are no longer used. Typically, this method is used to save the legs of the port and this is the way we will use today, as we lose speed twice, but we have a symbolic display and we have nowhere to hurry. In 4-bit mode, we transmit or receive a byte in 2 steps in half, first the highest part of the byte, then the lower one.

The legs A and K are the anode and the cathode for supplying voltage to power the LED backlight of the display. Typically, you can feed from 5 volts, and from 3 volts, but it is desirable to put a current-limiting resistor at 100 ohms and most likely then the display backlight will "live" longer. All this is usually indicated in the technical documentation on the display. We will deliver at 330.

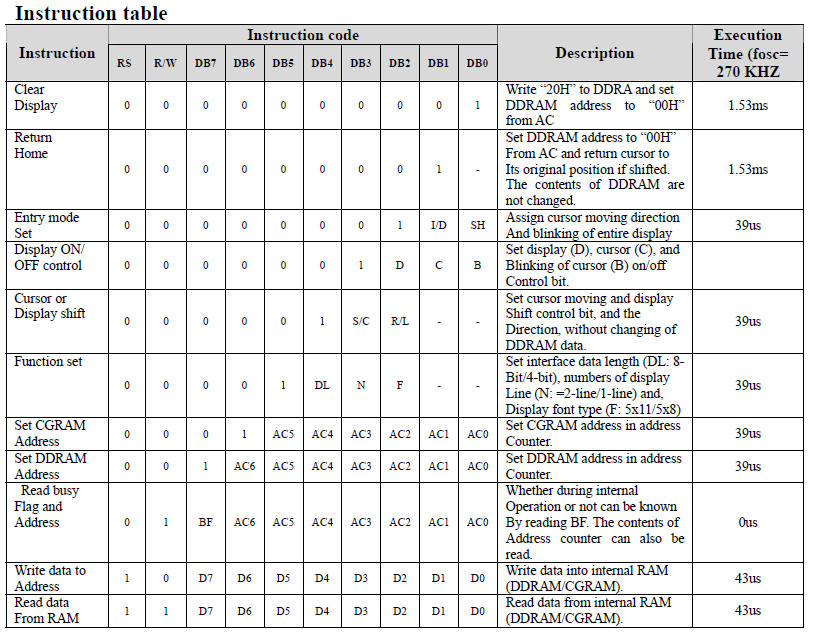
For the data, we will use port D, and for servicing - 8 and 9 of the foot of port B.

To get a certain symbol in a certain place on the display, we need to send the code of this symbol to a specific address in DDRAM memory. For this purpose in the technical documentation there is such information



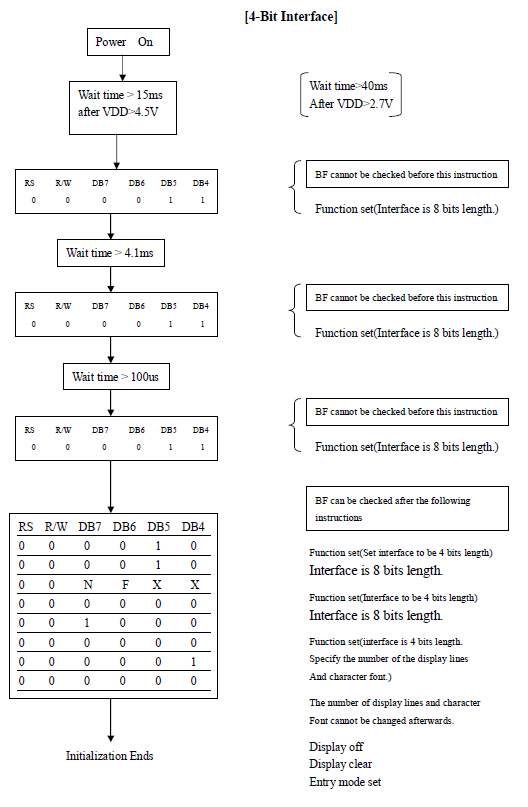
Here we see at what address we will need to send a byte with the character code.

Also in the documentation we are interested in the display control commands. There is a table explaining the command data, for what it is needed, and also how long it takes to execute this command or instruction



Also, the condition for executing a command with such a time interval is the timing of the 270 kHz display controller.

Also, to launch our display into operation, since there is a controller there, initial initialization will be necessary. In the documentation, the truth in the other - the display 1602 is the order of initialization. It does not differ from the initialization of the 2004 display. For a 4-bit connection method, it is

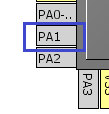
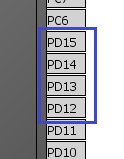


I made the **MYLCD80** project from **TEST002** in the same way as in the last lesson.

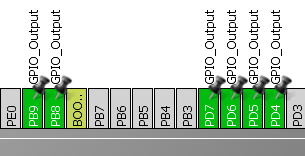
Open MYLCD80 in the cube. Disable timer TIM6



Also, as unnecessary to disconnect the tabs PA1, PD12, PD13, PD14, PD15

Include the output of the tabs PB8, PB9, PD0, PD4, PD5, PD6, PD7



In the settings do not touch anything. We generate the project, we collect it.

We see the errors on our manual code on the timer, remove the line

~~tim6\_counter = 0;~~

and also everything we wrote from an infinite cycle.

From the main.h variable

~~uint8\_t tim6\_counter;~~

We connect the new files lcd.h and lcd.c with the contents:

lcd.h:

#ifndef LCD\_H\_

#define LCD\_H\_

#include "stm32f4xx\_hal.h"

#endif / \* LCD\_H\_ \* /

lcd.c:

#include "lcd.h"

Then we connect lcd.h to main.h

main.h:

#include "lcd.h"

We will write the defines to the lcd.h file, for convenient control of the levels on the data legs

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

**#define d4\_set () HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_4, GPIO\_PIN\_SET)**

**#define d5\_set () HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_5, GPIO\_PIN\_SET)**

**#define d6\_set () HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_6, GPIO\_PIN\_SET)**

**#define d7\_set () HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_7, GPIO\_PIN\_SET)**

**#define d4\_reset () HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_4, GPIO\_PIN\_RESET)**

**#define d5\_reset () HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_5, GPIO\_PIN\_RESET)**

**#define d6\_reset () HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_6, GPIO\_PIN\_RESET)**

**#define d7\_reset () HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_7, GPIO\_PIN\_RESET)**

**#define e1 HAL\_GPIO\_WritePin (GPIOB, GPIO\_PIN\_9, GPIO\_PIN\_SET) // set the line E to 1**

**#define e0 HAL\_GPIO\_WritePin (GPIOB, GPIO\_PIN\_9, GPIO\_PIN\_RESET) // set the line E to 0**

**#define rs1 HAL\_GPIO\_WritePin (GPIOB, GPIO\_PIN\_8, GPIO\_PIN\_SET) // set the RS line to 1 (data)**

**#define rs0 HAL\_GPIO\_WritePin (GPIOB, GPIO\_PIN\_8, GPIO\_PIN\_RESET) // set the RS line to 0 (command)**

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

We look above the table how to initialize the 4-bit mode and actually write the initialization.

We begin with a delay

**void LCD\_ini (void)**

**{**

**HAL\_Delay (40);**

**}**

We write a prototype on it in the header file.

Then we need to write the function of writing data to the memory of the display

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

**{**

**if (((dt >> 3) & 0x01) == 1) {d7\_set ();} else {d7\_reset ();}**

**if (((dt >> 2) & 0x01) == 1) {d6\_set ();} else {d6\_reset ();}**

**if (((dt >> 1) & 0x01) == 1) {d5\_set ();} else {d5\_reset ();}**

**if ((dt & 0x01) == 1) {d4\_set ();} else {d4\_reset ();}**

**}**

Write a function for a small delay

**void delay (void)**

**{**

**uint16\_t i;**

**for (i = 0; i <1000; i ++)**

**{**

**}**

**}**

Now write the function for the commands

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

**{**

**rs0;**

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

**e1;**

**delay ();**

**e0;**

**LCD\_WriteData (dt);**

**e1;**

**delay ();**

**e0;**

**}**

Then we continue to write the initialization function of the display, glancing at times in the technical documentation and in the [lesson on AVR](http://narodstream.ru/avr-urok-12-lcd-indikator-16x2/) , where it was told already about each team

void LCD\_ini (void)

{

        HAL\_Delay (40);

**rs0;**

**LCD\_WriteData (3);**

**e1;**

**delay ();**

**e0;**

**HAL\_Delay (1);**

**LCD\_WriteData (3);**

**e1;**

**delay ();**

**e0;**

**HAL\_Delay (1);**

**LCD\_WriteData (3);**

**e1;**

**delay ();**

**e0;**

**HAL\_Delay (1);**

**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 (0x0F); // the display is turned on (D = 1), we also turn on all cursors**

**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**

**HAL\_Delay (2);**

}

We call the initialization from the main module

  / \* USER CODE BEGIN 2 \* /

**LCD\_ini ();**

  / \* USER CODE END 2 \* /

Stitch, watch.

