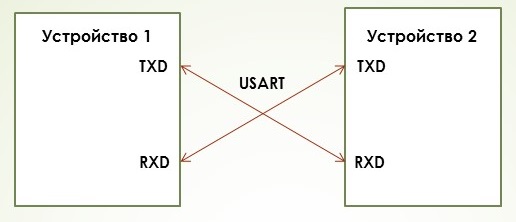
**STM Lesson 13. HAL. USART. Data transfer**

Today I will tell you what **USART is** and how to use it in STM32 controllers.

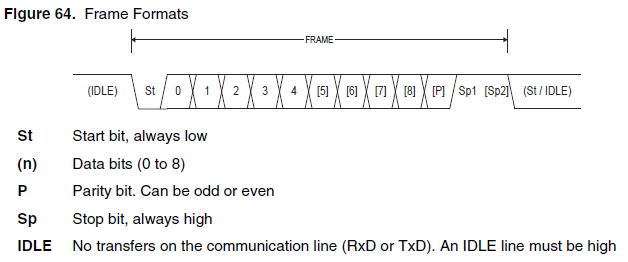
**USART** is a data transfer interface that takes place over two wires between two devices, in which data transmission and reception occurs over two separate wires.

And the data transfer contact of one device is connected to the data receiving terminal of the other and vice versa.



In detail we will not here study this interface, its protocol and other subtleties, this is all detailed in the [**lesson for AVR**](http://narodstream.ru/avr-urok-14-usart-svyaz-mk-s-pk-chast-1/) .

In general, USART data transfer occurs in this way



Those bits, the designations of which are given in braces, may not be used in certain modes.

At the beginning of the diagram, we see that when the bus is not used and the data for a particular wire is not transmitted or received, depending on the purpose of the contact, this contact is in a high state. As it is commonly said, the passive state on the bus is high. Next is the mandatory start bit. As we see, it is directed downwards, that is, at this time the contact passes from a high state to a low state. How long this bit lasts, it's no wonder. It is necessary to divide 1 second into the speed set for data transmission and reception, which is measured in bits per second well or, as they are also called, in bodah. After the time of the stop bit expires, the controller will consider the following bits as information bits. They can be from five to nine depending on the regime. In general, most often it is used 8, since it is most convenient to transfer data bytes. Well here the state of the leg during the transmission of a certain information bit will be dictated by the information bit itself. That is, if you need to transfer a unit - then high, if zero - low. It is also important to note that the transmission of information bits begins with the youngest, and then the older ones.

The next bit of **P** is the parity bit. Depending on the mode, it can be, and also it may not be. That is, when we transfer one block, it will be exposed, and the next one will be reset, for even greater synchronization.

Then there are stop bits saying that the transmission of the parcel is over. The data bits can be one or two also depending on the mode. These bits are transmitted using a high logic state on the leg. And then the process is repeated again, that is, we pass the next parcel.

How can we do all this in practice to connect our controller with a PC, provided that this interface does not exist on the PC. Such things are solved by means of interface converters or adapters.

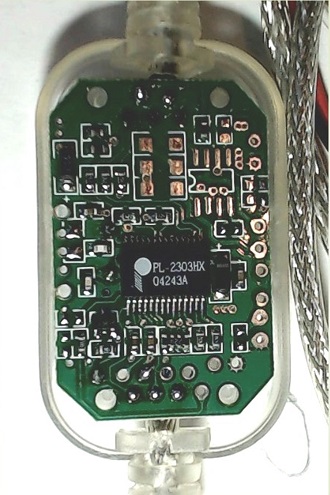
These adapters are of different types depending on their manufacturers

I have a self-made adapter made of an old telephone data cable on a PL2303HX chip. Here and so it looks like



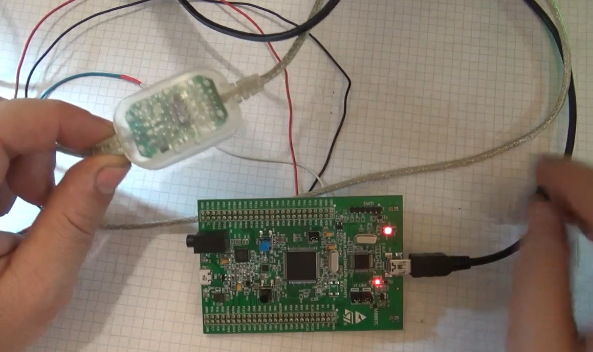
Here its internal view from the chip side



I brought out the necessary wires from the corresponding contacts and soldered them to the tips. I soldered the wires in such a way

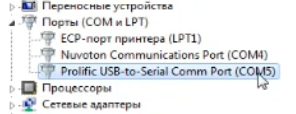


And so this adapter looks like in the state connected to the Discocery board



And to which board legs the adapter is connected, we will see in the Cube MX program, when we will set up our project there.

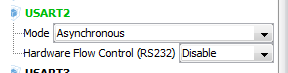
The installed driver for the adapter looks on the Windows system like this



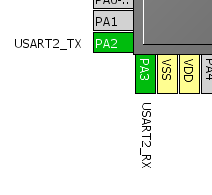
In order for us to see the data on the PC, we need a terminal program. We will use the **Terminal 1.9b**program , the link to which will be given at the bottom of the page.

Just as before, the project is created from TEST001, call it USART\_TRANSMIT

Start the cube, turn off all the ports for the LEDs, they do not need us yet, the button port is also disabled we turn USART2 into the Asynchronous position

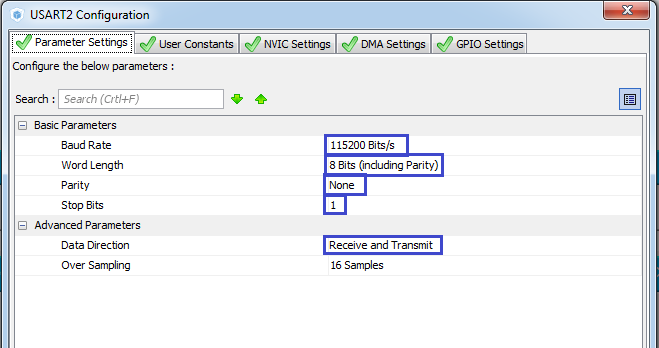


We see that we have also included the legs of the ports in certain alternative modes.



Accordingly, we will connect the wires of our adapter to these legs.

We do not touch anything in the Configuration settings at all, just check to see if we have turned on correctly, as some versions of Cube MX can by default enable 7-bit data transfer mode, we need 8-bit



We need to set the same settings in the terminal program.

We generate the project.

In the main function main () we add a variable

  / \* USER CODE BEGIN 1 \* /

**uint8\_t str [] = "USART Transmit \ r \ n";**

  / \* USER CODE END 1 \* /

In an infinite loop, we add

  while (1)

  {

**HAL\_UART\_Transmit (& huart2, str, 16.0xFFFF);**

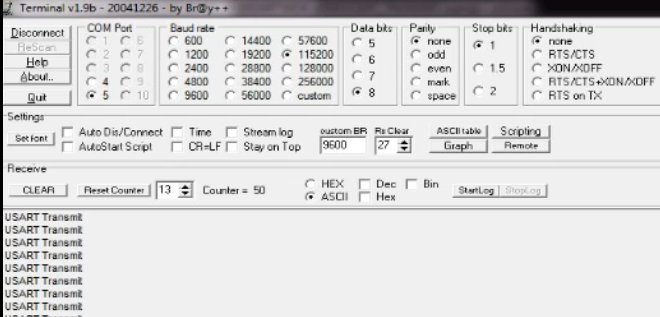
**HAL\_Delay (100);**

  / \* USER CODE END WHILE \* /

In this function, as input parameters, we feed a reference to the periphery, to the data array, the number of bytes transferred and the timeout.

And then add a delay, otherwise without it our terminal program will hang.

We collect the project, open the terminal program, set all the settings there, press the **Connect** button, flash , watch



Everything is transmitted from us. But here it is easier.

With reception it will be more difficult, for this purpose it will be necessary to connect the display, to process interrupts.

Therefore, we will postpone this matter until the [next session](http://narodstream.ru/stm-urok-14-hal-usart-priem-dannyx/) .