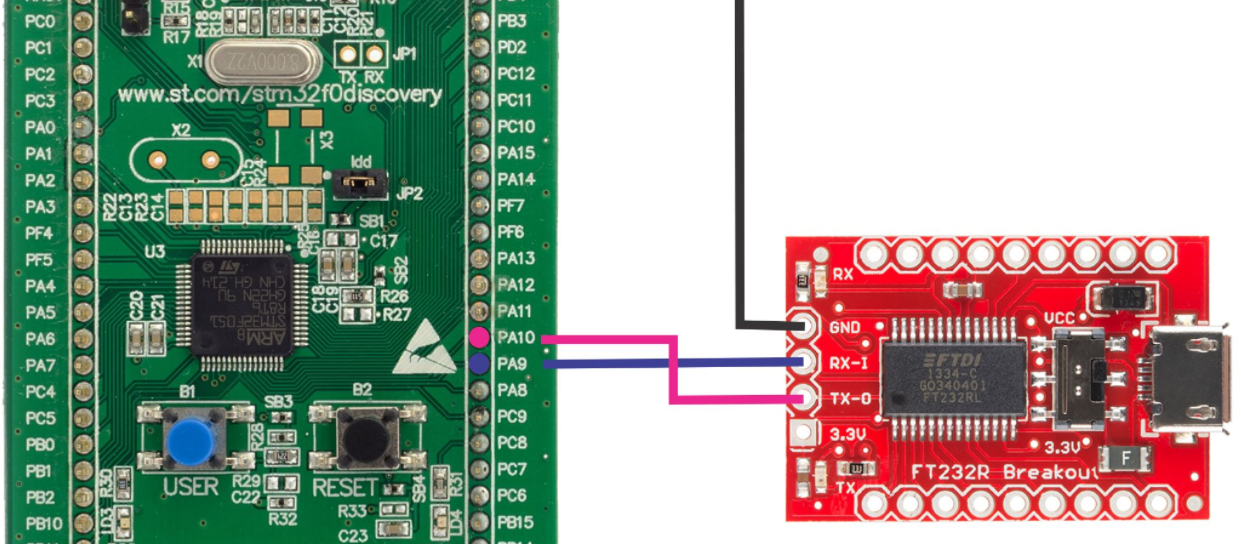
**STM32F0 UART – Tutorial**



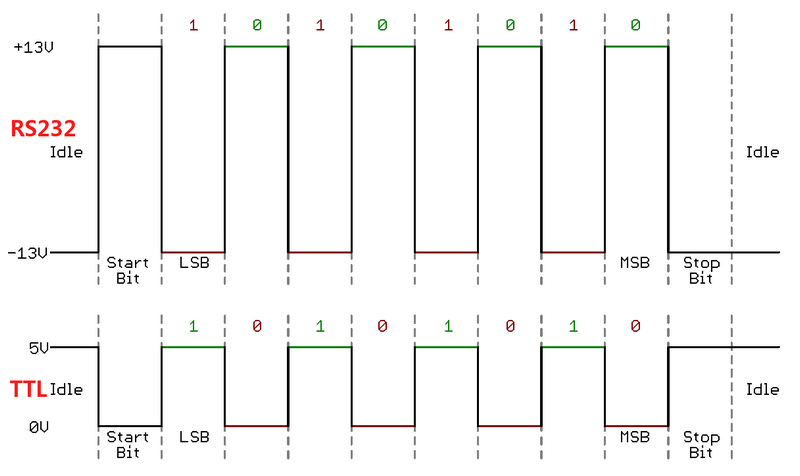
1. [Hardware connection](https://letanphuc.net/2015/09/stm32f0-uart-tutorial-5/#hardware-connection)
2. [Set up in CubeMX](https://letanphuc.net/2015/09/stm32f0-uart-tutorial-5/#set-up-in-cubemx)
3. [Send data to the computer](https://letanphuc.net/2015/09/stm32f0-uart-tutorial-5/#send-data-to-the-computer)
4. [**Receive data using Interrupt**](https://letanphuc.net/2015/09/stm32f0-uart-tutorial-5/#receive-data-using-interrupt)

It’s been a while since I posted the tutorial on using [STM32F0’s timer and counter](https://letanphuc.net/2015/06/stm32f0-timer-tutorial-and-counter-tutorial/). I’ve recently started my MSc course in NUS and found it quite challenging. I almost forgot all the backgrounds that I studied at university after 2 years of working. I’ve been spending most of my time reviewing old notes and working on new lectures. But today I’ll set them aside and enjoy my time here with the STM32F0 UART tutorial.

Ok, so if you have already worked with Arduino before, you must be very familiar with “Serial.print” command. For those who don’t know what it is, it’s simply a way of communication between two or more devices using a serial interface, which means data will be transmitted and received one by one bit sequentially using typically less than 4 wires. [Wikipedia](https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter) has very detailed knowledge on this if you want to understand more. In this tutorial, we will use this serial interface to transfer data, make a communication bridge between our device (STM32F0) to a computer or other devices such as Arduino, PIC, etc.

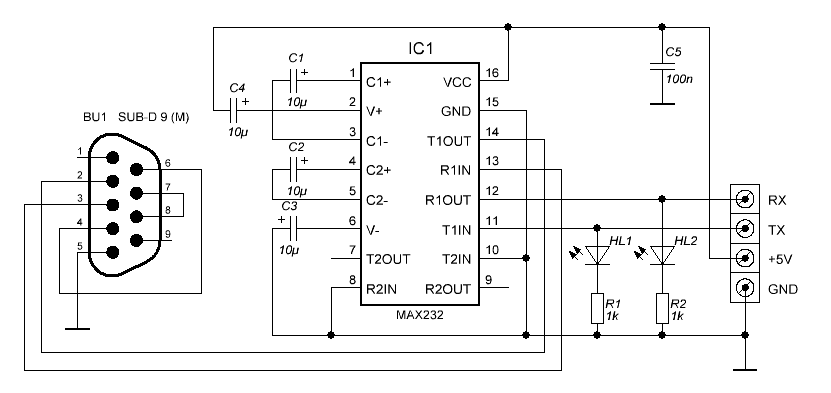
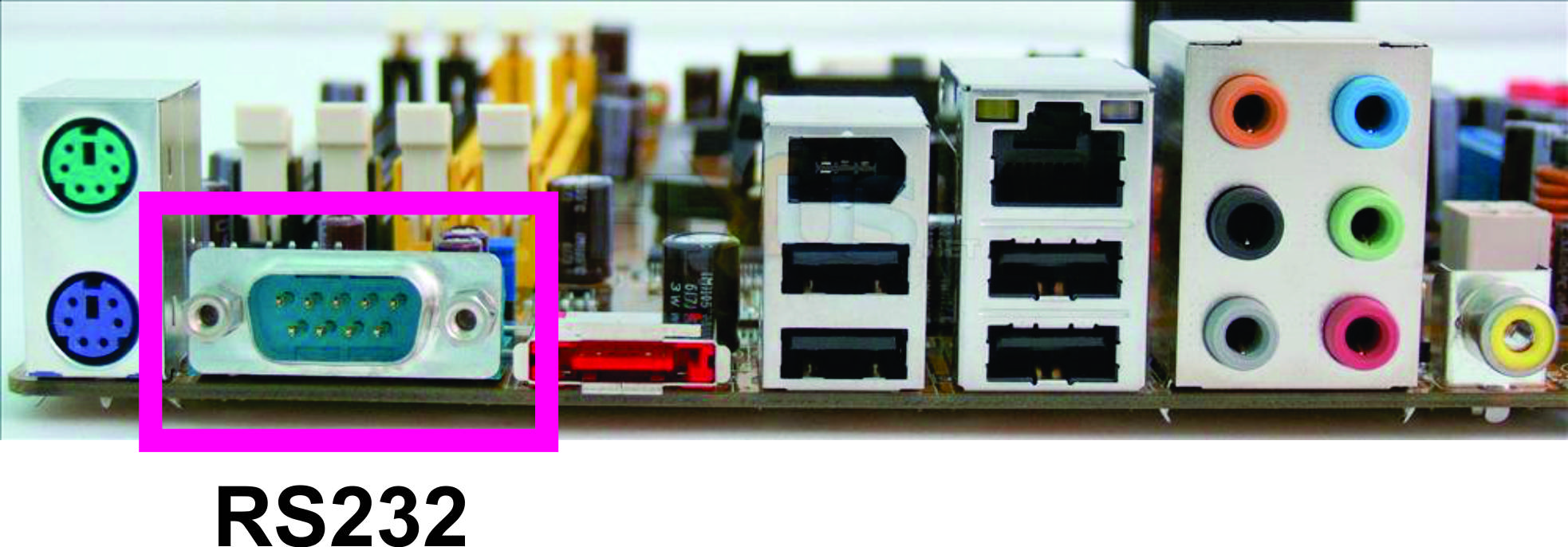
**Hardware connection**

Before going into details, there are two terms we need to clarify: RS232 and UART. What are the differences between the two? Take a look at the following picture:

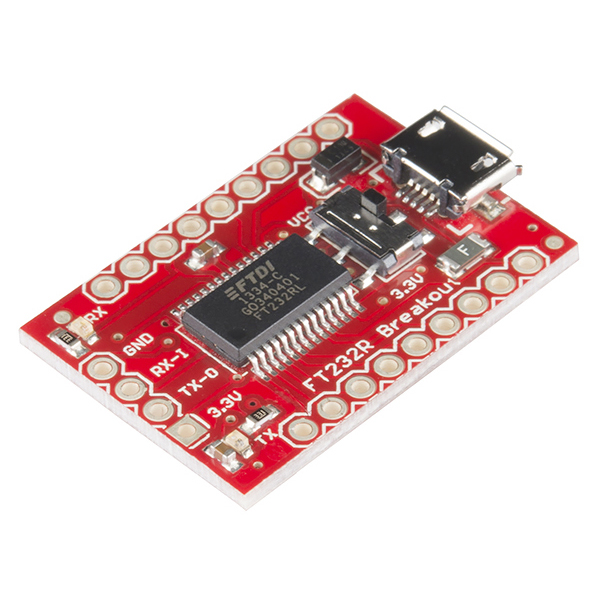


RS232 vs UART source: [http://www.crifan.com](http://www.crifan.com/)

We can clearly see the difference in voltage between the two, one is +-12V, one is 0-5V (or 0-3.3V). What your PC has (hopefully it still has nowadays) is an RS232 port (COM port shown in Manager), which is the 9-pin connector usually above those USB ports. To use this port to connect your STM32F0 to the computer, we need something to translate +-12V into 3.3V and MAX232 was born to do that job. **So, if you intend to use the RS232 port on your PC as a way to connect to STM32F0 or any kind of microcontroller, remember to use MAX232 or you will burn your devices.**

MAX232 – Source:robomart.com

However, this RS232 port is no longer exists on those new laptops, and tablets, but only a USB port. So, to make it easier, we try to use the next method which uses a USB port as a virtual RS232. All you need to do is to get one USB-UART converter to continue with this tutorial. This USB-UART converter can be found easily from one of those sources: [Ebay](https://www.ebay.com/itm/FT232RL-3-3V-5-5V-FTDI-USB-to-TTL-Serial-Adapter-Module-for-Arduino-Mini-Port/141724029351) or [Sparkfun](https://www.sparkfun.com/products/12731). This board will create a virtual serial port on your computer (COM port).

FT232 USB-UART

No matter what version of FT232 USB to serial board or any kinds of USB to UART board you have, PL2303, CP2012, or CH340, there will be those similar pins:

1. GND: Ground
2. VCC (5V or 3.3V): Power supply
3. TX (TXD): Transmit (from computer to device)
4. RX (RXD): Receive (from device to computer)

Since STM32F051 on the Discovery kit has 2 UART modules (UART1 and UART2), we can try using both in this tutorial. They are identical to each other in the sense of function, command, … and can be mapped to different pins easily. The picture below shows how to connect the STM32F0 kit to the FT232 Board. For the first time, we will use UART1 mapped with pin PA9 and PA10. In order to send or receive data through a Serial port on the computer, I would recommend Hercules Terminal which you can download here: <http://www.hw-group.com/products/hercules/index_en.html>

**Set up in CubeMX**

After having everything connected, we can start playing with the configuration part in CubeMX. The following video shows you how to initialize the typical configuration for the UART interface, and transmit and receive data.

There are a few things we need to pay attention to when setting up the UART function:

* Baud rate: this parameter decides how fast the communication is. The standard baud rates are: 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 250000, 230400, 460800, 921600, 1.5Mbps, 3Mbps, etc. The bigger the number is, the faster your communication is. FT232 supports up to 3Mbps and that is really fast. Normally, we just use up to 115200bps for all applications.
* Word length: usually 8bit will do the job, unless you want to do multi-slave communication then we can use 9bit length. Multi-slave communication can also be done with 8bit.

**Send data to the computer**

There are two common ways to send data to a computer or other devices: one, we can take advantage of the “[printf](http://www.cplusplus.com/reference/cstdio/printf/)” function that is available in Keil and two, use the built-in HAL library. Before using the HAL library with cubeMX, I personally preferred the first choice. Now I just change to use the second method, combining between HAL library and “[sprintf](http://www.cplusplus.com/reference/cstdio/sprintf/)” function. You should check both functions in [cplusplus.com](http://www.cplusplus.com/reference/cstdio/printf/) to know which parameter is used to transmit which type of variable. For example, “%i” is for integer; “%.2f” is for floating number with 2 digits after decimal point; “%s” for string; “\r” equal to ASCII 13; “\n” equal to ASCII 10…

This example was extracted from cplusplus.com for your reference on how to use ‘printf’ and ‘sprintf’

/\* printf example \*/

#include <stdio.h>

int main()

{

printf ("Characters: %c %c \n", 'a', 65);

printf ("Decimals: %d %ld\n", 1977, 650000L);

printf ("Preceding with blanks: %10d \n", 1977);

printf ("Preceding with zeros: %010d \n", 1977);

printf ("Some different radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);

printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);

printf ("Width trick: %\*d \n", 5, 10);

printf ("%s \n", "A string");

return 0;

}

The output will be

Characters: a A

Decimals: 1977 650000

Preceding with blanks: 1977

Preceding with zeros: 0000001977

Some different radices: 100 64 144 0x64 0144

floats: 3.14 +3e+000 3.141600E+000

Width trick: 10

A string

Back to the transmission part in the video clip, for the firstmethod, the configuration code is quite complicated, but the transmission part is very simple:

/\* Private function prototypes -----------------------------------------------\*/

#ifdef \_\_GNUC\_\_

/\* With GCC/RAISONANCE, small printf (option LD Linker->Libraries->Small printf

set to 'Yes') calls \_\_io\_putchar() \*/

#define PUTCHAR\_PROTOTYPE int \_\_io\_putchar(int ch)

#else

#define PUTCHAR\_PROTOTYPE int fputc(int ch, FILE \*f)

#endif /\* \_\_GNUC\_\_ \*/

/\*\*

\* @brief Retargets the C library printf function to the USART.

\* @param None

\* @retval None

\*/

PUTCHAR\_PROTOTYPE

{

/\* Place your implementation of fputc here \*/

/\* e.g. write a character to the USART \*/

HAL\_UART\_Transmit(&huart1, (uint8\_t \*)&ch, 1, 100);

return ch;

}

/\* USER CODE END PFP \*/

Wherever you want to send data to the computer, just use the following command

//For sending data out in while(1)

printf("Hello\r\n");

printf("This is i:%i\r\n",i);

For the **second method**, there is no configuration part, but the sending part needs to be:

//print out static text

sprintf(buffer,"Hello\r\n");

len=strlen(buffer);

HAL\_UART\_Transmit(&huart1, buffer, len, 1000);

//print out variable

sprintf(buffer,"This is i:%i\r\n",i);

len=strlen(buffer);

HAL\_UART\_Transmit(&huart1, buffer, len, 1000);

**Receive data using Interrupt**

In the video, I have shown you one possible way to use UART interrupt to receive data from a computer in a string finalized by a CR (ASCII 13 or Enter). There are also many ways to receive UART data using a timer or other methods. Here, for this method, the basic mechanism is storing each received byte into an array until receiving a CR. After that, a variable called “Transfer\_cplt*”*is triggered to announce that receiving is completed*.*Therefore, after reading out the data, you should reset this “Transfer\_cplt” for a new session.

The UART interrupt routine in the clip is as follows:

//variables need to be declared at the beginning

char Rx\_indx, Rx\_data[2], Rx\_Buffer[100], Transfer\_cplt;

//Interrupt callback routine

void HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart)

{

uint8\_t i;

if (huart->Instance == USART1) //current UART

{

if (Rx\_indx==0) {for (i=0;i<100;i++) Rx\_Buffer[i]=0;} //clear Rx\_Buffer before receiving new data

if (Rx\_data[0]!=13) //if received data different from ascii 13 (enter)

{

Rx\_Buffer[Rx\_indx++]=Rx\_data[0]; //add data to Rx\_Buffer

}

else //if received data = 13

{

Rx\_indx=0;

Transfer\_cplt=1;//transfer complete, data is ready to read

}

HAL\_UART\_Receive\_IT(&huart1, Rx\_data, 1); //activate UART receive interrupt every time

}

}

Rx\_Buffer is the array that stores received data.

Thanks you !!!

Refer: <https://letanphuc.net/2015/09/stm32f0-uart-tutorial-5/>

Video: https://www.youtube.com/watch?v=Gxcf4UFAOoE/