6a laboratory work

UART WITE

1. Aim

- Learn how to use STM32CubeIDE for programming of STM32 microcontrollers.
- Use basic in Embedded C language.
- Learn how to send data Using UART.

1. Theory

A universal asynchronous receiver-transmitter (UART) is a hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable.

The electric signaling levels and methods are handled by a driver circuit external to the UART. A UART is usually an individual (or part of an) integrated circuit (IC) used for serial communications over a computer or peripheral device serial port. One or more UART peripherals are commonly integrated in microcontroller chips. A related device, the universal synchronous and asynchronous receiver-transmitter (USART) also supports synchronous operation.

The universal asynchronous receiver-transmitter (UART) takes bytes of data and transmits the individual bits in a sequential fashion

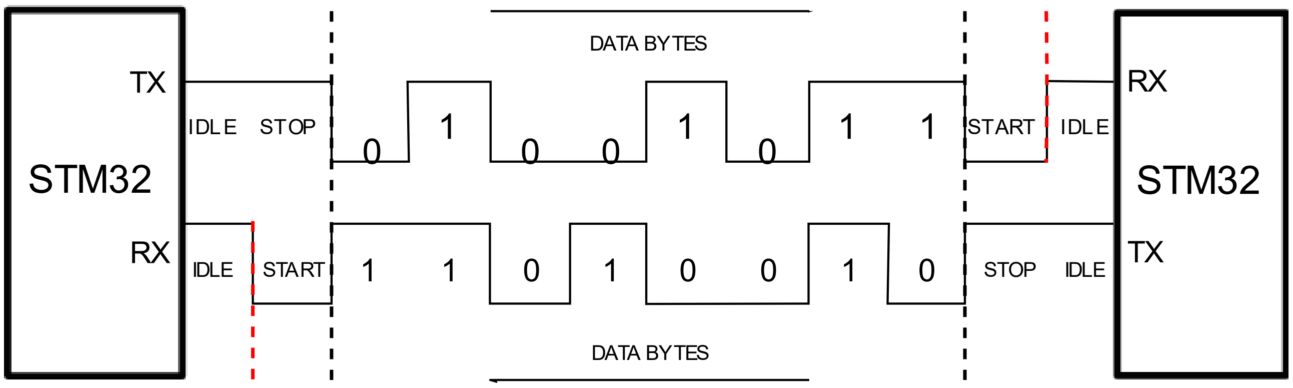


Fig 1. UART communication

The image above shows a typical UART connection. The transmitting UART receives data from the controlling device through the data bus. The controlling device can be anything like a CPU of a microprocessor or a microcontroller, memory unit like a RAM or ROM, etc. The data received by the transmitting UART from the data bus is parallel data.

To this data, the UART adds Start, Parity and Stop bits in order to convert it into a data packet. The data packet is then converted from parallel to serial with the help of shift register and is transmitted bit – by – bit from the TX pin.

The receiving UART receives this serial data at the RX pin and detects the actual data by identifying the start and stop bits. Parity bit is used to check the integrity of the data.

Up on separating the start, parity and stop bits from the data packet, the data is converted to parallel data with the help of shift register. This parallel data is sent to the controller at the receiving end through a data bus.

Structure of Data Packet or Frame

The data in UART serial communication is organised in to blocks called Packets or Frames.

Frame	START	DATA	PARITY	STOP
Length (Size)	1 bit	5-9 bits	0-1 bit	1-2 bits

Start Bit: Start bit is a synchronisation bit that is added before the actual data. Start bit marks the beginning of the data packet. Usually, an idle data line i.e. when the data transmission line is not transmitting any data, it is held at a high voltage level (1).

In order to start the data transfer, the transmitting UART pulls the data line from high voltage level to low voltage level (from 1 to 0). The receiving UART detects this change from high to low on the data line and begins reading the actual data. Usually, there is only one start bit.

Stop Bit: The Stop Bit, as the name suggests, marks the end of the data packet. It is usually two bits long but often only on bit is used. In order to end the transmission, the UART maintains the data line at high voltage (1).

Parity Bit: Parity allows the receiver to check whether the received data is correct or not. Parity is a low – level error checking system and comes in two varieties: Even Parity and Odd Parity. Parity bit is optional and it is actually not that widely used.

Data Bits: Data bits are the actual data being transmitted from sender to receiver. The length of the data frame can be anywhere between 5 and 9 (9 bits if parity is not used and only 8 bits if parity is used). Usually, the LSB is the first bit of data to be transmitted (unless otherwise specified).

Rules of UART

As mentioned earlier, there is no clock signal in UART and the transmitter and receiver must agree on some rules of serial communication for error free transfer of data. The rules include:

- Synchronisation Bits (Start and Stop bits)

- Parity Bit
- Data Bits and
- Baud Rate

We have seen about synchronisation bits, parity bit and data bits. Another important parameter is the Baud Rate.

Baud Rate: The speed at which the data is transmitted is mentioned using Baud Rate. Both the transmitting UART and Receiving UART must agree on the Baud Rate for a successful data transmission.

Baud Rate is measured in bits per second. Some of the standard baud rates are 4800 bps, 9600 bps, 19200 bps, 115200 bps etc. Out of these 9600 bps baud rate is the most commonly used one.

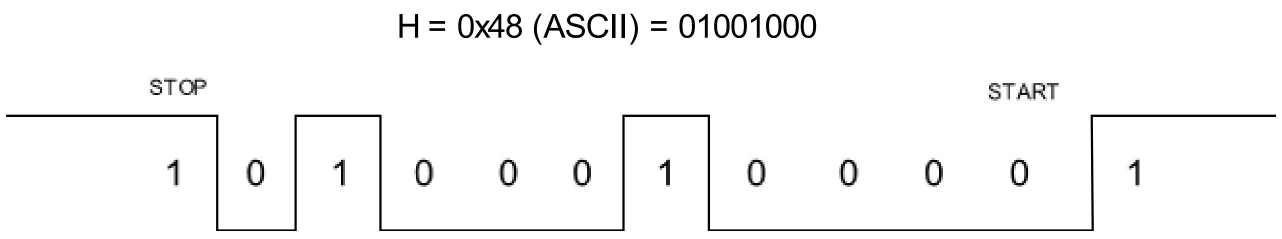
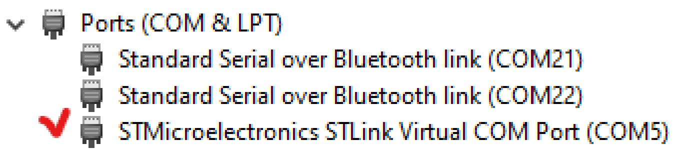


Fig 2. Example of UART packet

In this laboratory work we will send data from NUCLEO STM32 microcontroller to the PC, using virtual UART possibility.

If drivers of Nucleo board are properly installed, during connection of your board to the PC onboard ST link programmer creates virtual COM. You can find it in Device manager.

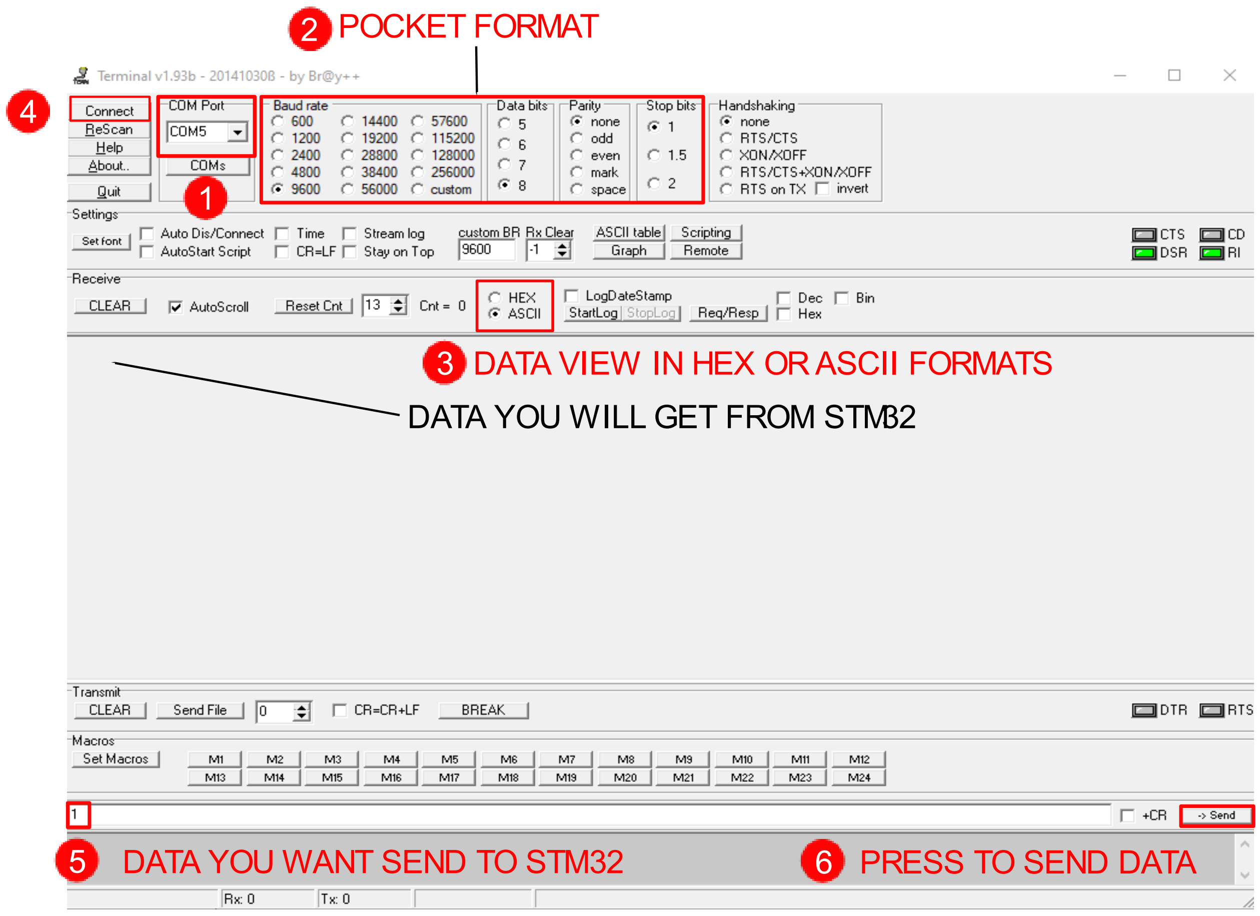


Ports (COM & LPT)
Standard Serial over Bluetooth link (COM21)
Standard Serial over Bluetooth link (COM22)
STMicroelectronics STLink Virtual COM Port (COM5)

Fig 3. Nucleo virtual com port in device manager of your PC

This gives you possibility to send/read data to/from your PC using terminal software. One we will use is open software Terminal created by Br@y (https://sites.google.com/site/terminalbpp/).

For start using it – you nee to select COM port of your STlink device, select baud rate and pocket format of your communication. By default is 9600, 8 data bits, No parity, One stop bit.


2 POCKET FORMAT
Terminal v1.93b - 20141030B - by Br@y++
4
Connect
ReScan
Help
About..
Quit
1
3 DATA VIEW IN HEX OR ASCII FORMATS
DATA YOU WILL GET FROM STM32
5 DATA YOU WANT SEND TO STM32
6 PRESS TO SEND DATA
-> Send

COM Port: COM5
Baud rate: [✓ 9600]
Data bits: [✓ 8]
Parity: [✓ none]
Stop bits: [✓ 1]
Handshaking: [✓ none]
Receive: HEX [✓ ASCII]

Fig 4. Serial Terminal interface

Setup STM32G0 for UART communication.

Create new project for STM32G070RB and setup TX (PA2) and RX(PA3) pins for UAR communication.

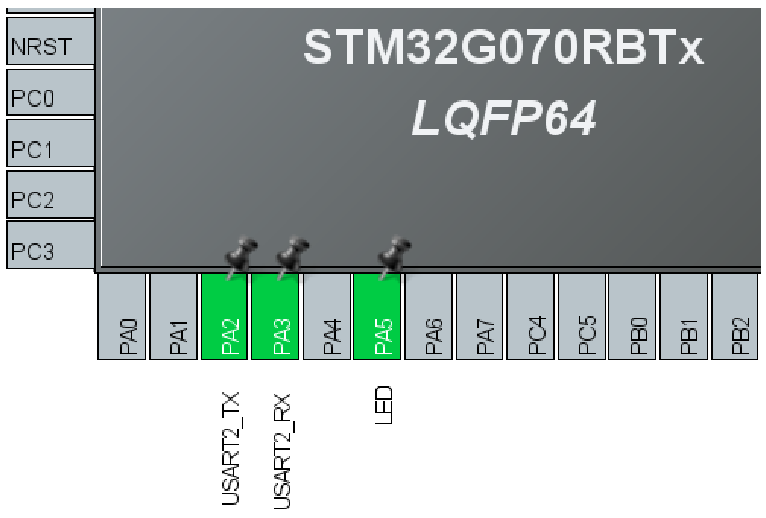
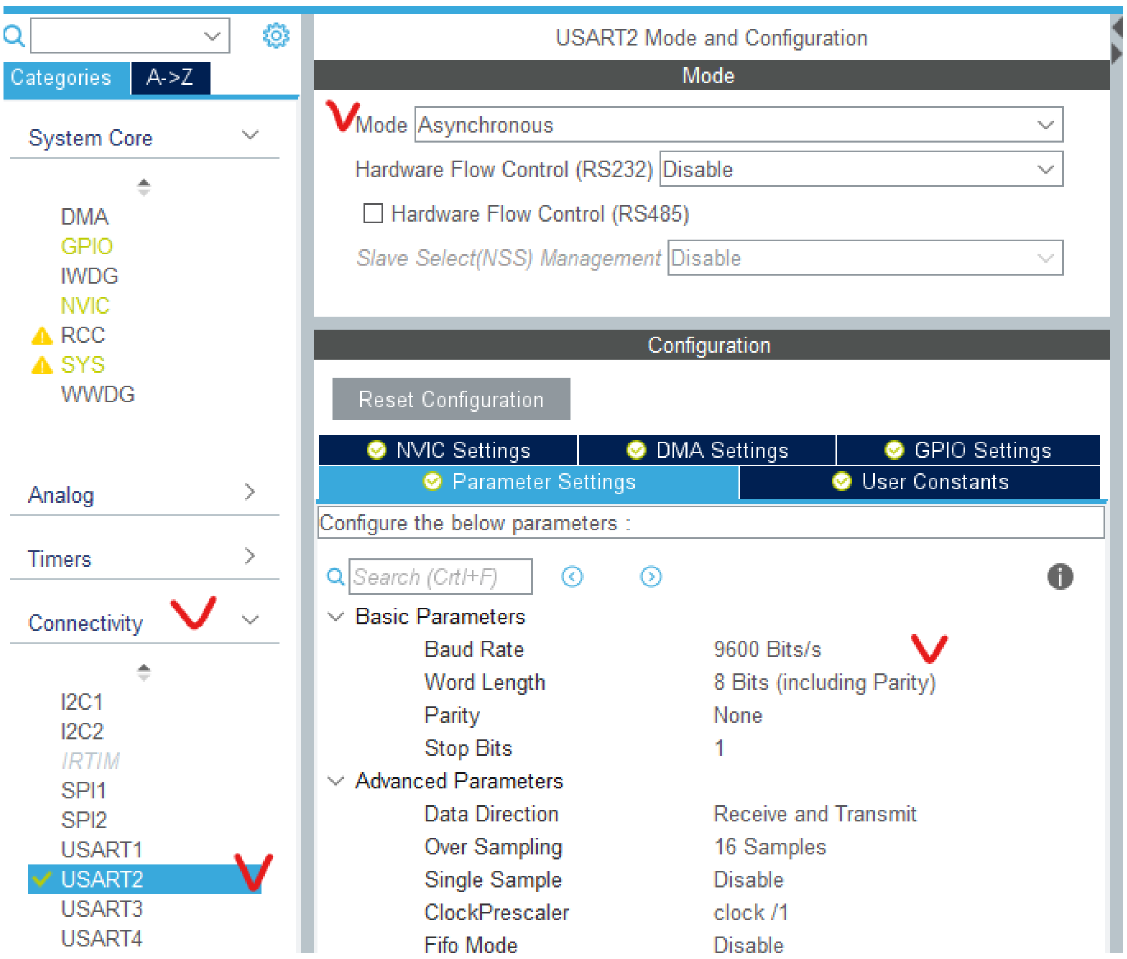


Fig 5. UART pins

Setup UART communication baud rate and data format.



Mode: Asynchronous
Hardware Flow Control (RS232): Disable
Hardware Flow Control (RS485)
Slave Select(NSS) Management: Disable

Basic Parameters	
Baud Rate	9600 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1
Advanced Parameters	
Data Direction	Receive and Transmit
Over Sampling	16 Samples
Single Sample	Disable
ClockPrescaler	clock /1
Fifo Mode	Disable

Fig 5. UART Settings

Save changes and go to the main.c and modify it using code below.

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "string.h"
```

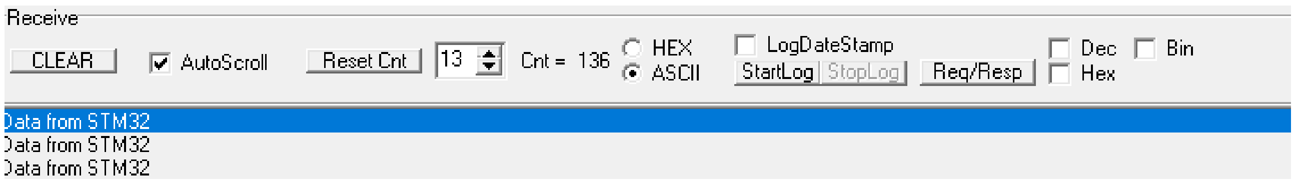
```
/* USER CODE BEGIN WHILE */
  while (1)
  {
		char *msg = "Data from STM32 \r\n";
		HAL_UART_Transmit(&huart2, (uint8_t*) msg, strlen(msg), 0xFFFF);
		HAL_Delay(1000);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
```

UART Transmit sample code

Start debug and press run.

Open terminal and press Connect. You shoud see string coming from STM32 to you terminal each second.



```
Data from STM32
Data from STM32
Data from STM32
```

Fig 6. Serial Terminal output

Feature that will help you to print strings and values in different format as string to UART

For print some text messages into UART you will need special functions, which located in C libraries **#include** "string.h" and **#include** "stdio.h". To use functions, you will need to add these libraries into your main.c.

```
/* Private includes ----------------------------------------------------------*/
#include "string.h"
#include "stdio.h"
```

Libraries for sprint functions

Function needed for combine string message and integer or float numbers is sprintf().

```
char stringR[10]; // declare array of char's
uint8_t R= 100; // declare uint8_t variable

sprintf(stringR,"R= %d Ohm",R);// combine string R=  Ohm with integer using %d special char

// send string to UART
HAL_UART_Transmit(&huart2, (uint8_t*) stringR, strlen(stringR), 0xFFFF);
```

Sprint() ussage

You will get this output in your terminal:

```
R= 100 Ohm
```

Character	Description
%	Prints a literal % character (this type doesn't accept any flags, width, precision, length fields).
%d	int as a signed **integer**. %d and %i are synonymous for output, but are different when used with scanf() for input (where using %i will interpret a number as hexadecimal if it's preceded by 0x, and octal if it's preceded by 0.)
u	Print decimal unsigned int.
f, F	double in normal (**fixed-point**) notation. f and F only differs in how the strings for an infinite number or NaN are printed (inf, infinity and nan for f; INF, INFINITY and NAN for F).
e, E	double value in standard form ([-]d.ddd e[+/-]ddd). An E conversion uses the letter E (rather than e) to introduce the exponent. The exponent always contains at least two digits; if the value is zero, the exponent is 00. In Windows, the exponent contains three digits by default, e.g. 1.5e002, but this can be altered by Microsoft-specific _set_output_format function.
g, G	double in either normal or exponential notation, whichever is more appropriate for its magnitude. g uses lower-case letters, G uses upper-case letters. This type differs slightly from fixed-point notation in that insignificant zeroes to the right of the decimal point are not included. Also, the decimal point is not included on whole numbers.
x, X	unsigned int as a **hexadecimal** number. x uses lower-case letters and X uses upper-case.

o	unsigned int in octal.
s	null-terminated string.
c	char (character).
p	void * (pointer to void) in an implementation-defined format.
a , A	double in hexadecimal notation, starting with 0x or 0X. a uses lower-case letters, A uses upper-case letters.[4][5] (C++11 iostreams have a hexfloat that works the same).
n	Print nothing, but writes the number of characters successfully written so far into an integer pointer parameter.

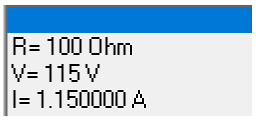
2. Tasks

2.1. Create and setup STM32G070RB project.

2.2. Analyse Nucleo board schematic form DM00452640 pdf file.

2.3. Write code for Send numbers from "1" to "10" into your terminal in decimal and hex formats.

2.4. Create variables for resistance and voltage in your code and write software wich calculates current and send into UART this output:



```
R= 100 Ohm
V= 115 V
I= 1.150000 A
```

3. Report content

1) Title.
2) Main blocks of source code for tasks with comments.
3) Conclusions.

1. References

1. https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-mainstream-mcus/stm32g0-series/stm32g0x0-value-line/stm32g070rb.html
2. https://www.electronicshub.org/basics-uart-communication/
3. https://sites.google.com/site/terminalbpp/