GOOD PRACTICE FOR UART COMMUNICATION USING ARDUINO WITH APPLICATIONS

Preprint · May 2020

DOI: 10.13140/RG.2.2.29680.48649

2 authors:

Mohamed Fezari
Badji Mokhtar - Annaba University
127 PUBLICATIONS **336** CITATIONS

SEE PROFILE

Ali Al Dahoud
Al-Zaytoonah University of Jordan
80 PUBLICATIONS **241** CITATIONS

SEE PROFILE

GOOD PRACTICE FOR UART COMMUNICATION USING ARDUINO WITH APPLICATIONS

Mohamed FEZARI*, Ali AL-Dahoud**
*Badji Mokhtar Annaba University Faculty of Engineering, Dept. Electronics
**AL Zaytoonah University Amman, Faculty of IT ,Amman, Jordan

Abstract: UART is one of the most useful serial interface, it is presented in microcontrollers and in single board computers.

It allows user to communicate in serial mode or connect a device in wireless mode. In this preprint we give an introduction on UARTs basically in Arduino modules. The protocol of communication is well explained and we finally, present three examples of using serial communication UART with Bluetooth, GPS sensor and for communication with another processor raspberry Pi.

1) Introduction:

In this work, we show what is UART communication and how it works. We will also write a simple sketch to explore serial interface for Arduino modules and present three applications with their code.

UART stands for **Universal Asynchronous Receiver/Transmitter**. It is a hardware device (or circuit) used for serial communication between two devices.

Here, we present connecting two UART devices together is simple and straightforward. Figure 1 shows a basic UART connection diagram. In order to get good results we need also to connect GNDs

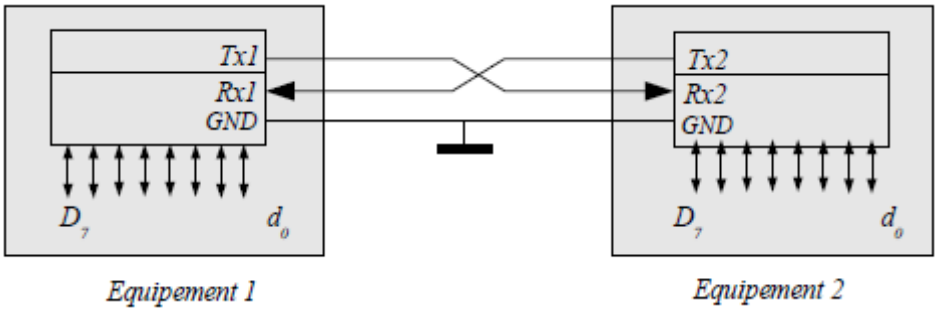

Figure 1: Basic UART Connection Diagram

One wire is for transmitting data (called the TX pin) and the other is for receiving data (called the RX pin). We can only connect two UART devices together.

UART works by converting data into packets for sending or rebuilding data from packets received.

2) TRANSMISSION DATA IN SERIAL

SENDING SERIAL DATA

Before the UART device can send data, the transmitting device converts the data bytes to bits. After converting the data into bits, the UART device then splits them into packets for transmission. Each packet contains a *start bit*, a *data frame*, *parity bit*, and the *stop bits*. Figure 2 shows a sample data packet.

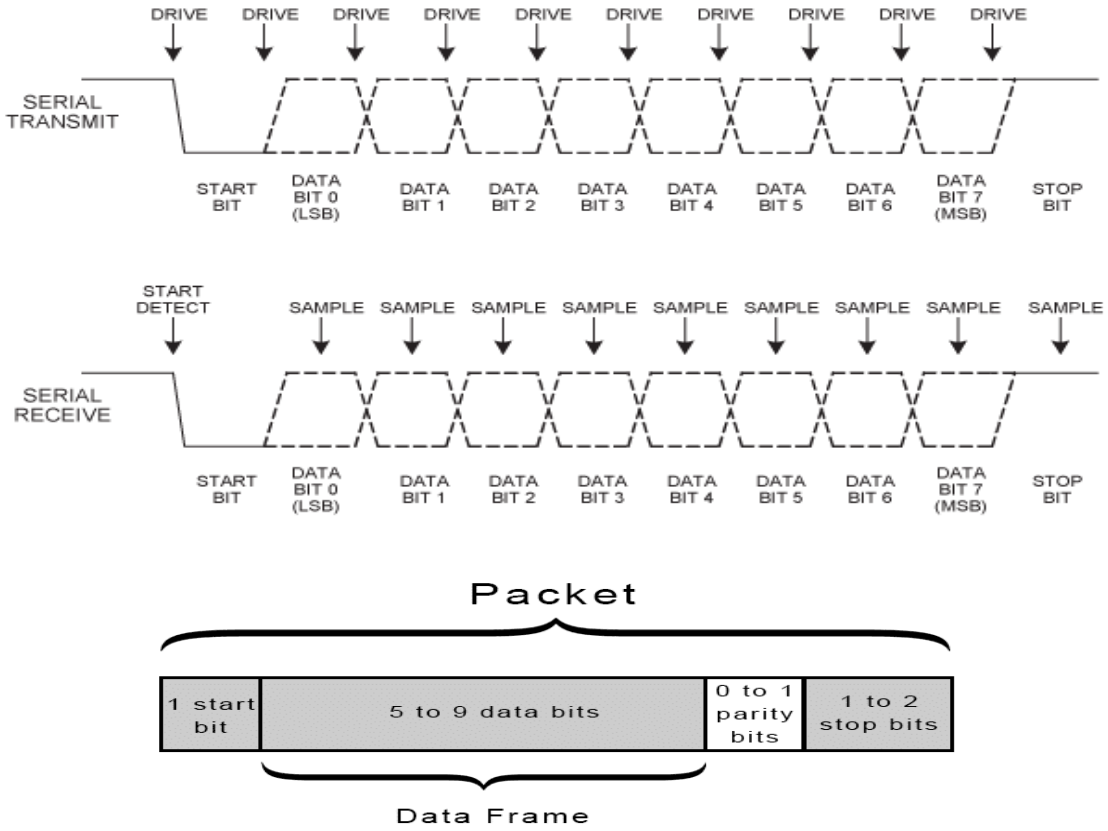

Figure 2: Data Packet elements

After preparing the packet, the UART circuit then sends it out via the TX pin.

RECEIVING SERIAL DATA

The receiving UART device checks the received packet (via RX pin) for errors by calculating the number of 1's and comparing it with the value of the parity bit contained in the packet. If there are no errors in transmission, it will then proceed to strip the start bit, stop bits, and parity bit to get the data frame. It may need to receive several packets before it can rebuild the whole data byte from the data frames. After rebuilding the byte, it is stored in the UART buffer.

The receiving UART device uses the parity bit to determine if there was a data loss during transmission. Data loss in transmission happens when a bit changed its state while being transmitted. Bits can change because of the transmission distance, magnetic radiation, and mismatch baud rates, among other things.

3) UART PARAMETERS

UART has settings that need to be the same on both devices to have proper communication. These UART settings are the *baud rate*, *data length*, *parity bit*, *number of stop bits*, and *flow control*.

A)BAUD RATE

Baud rate is the number of *bits per second (bps)* a UART device can transmit/receive. We need to set both UART devices with the same baud rate to have the proper transmission of data. Common values for baud rate are 9600, 1200, 2400, 4800 , 19200, 38400, 57600, and 115200 bps.

B)DATA LENGTH

Data length refers to the number of bits per byte of data.

C)PARITY BIT

The parity bit is a bit added to the transmitted data and tells the receiver if the number of 1's in the data transmitted is odd or even. The possible setting for Parity Bit is *Odd* or *Even*.

- **ODD** – the parity bit is '1' if there is an *odd* number of 1's in the data. Otherwise, the parity bit is set to zero.
- **EVEN** – the parity bit is '1' if there is an even number of 1's in the data. Otherwise, the parity bit is set to zero.

D)NUMBER OF STOP BITS

UART devices can use *none*, *one* or *two* stop bits to mark the end of a set of bits (called packets) transmitted.

E)FLOW CONTROL

Flow Control is the method to avoid the risk of losing data when transmitting data over UART. The UART device uses special characters as flow control to start/stop transmission.

4) ARDUINO UART INTERFACE

Arduino has one or more UART pins depending on the board. For our project, we will use an Arduino Uno which has only one UART interface found on pin **0** (RX0) and pin **1** (TX0)**.** The Arduino pins **0** and **1** are also used for communicating with the Arduino IDE via the USB. So if you will upload sketches to your UNO, be sure to first disconnect any wires on pins **0** and **1**. Figure 3 shows the location of the UART TX and RX pins.

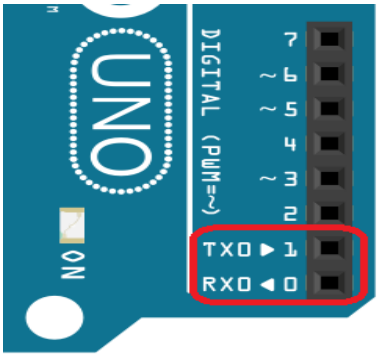

Figure 3: Arduino Uno TX/RX pins

UART LOGIC LEVEL

The UART logic levels may differ between manufacturers. For example, an Arduino Uno has a 5-V logic level but a computer's RS232 port has a +/-12-V logic level. Connecting an Arduino Uno directly to an RS232 port will damage the Arduino. If both UART devices don't have the same logic levels, a suitable logic level converter circuit is needed to connect the devices.

5) A SIMPLE UART PROJECT

After learning how the UART works, let us now build a simple sketch demonstrating how to use UART communication using Arduino Uno.

Our project is about controlling the built-in LED of an Arduino remotely via UART. A push-button wired to the first Uno board will control the built-in LED of the second Uno board and vice versa.

COMPONENTS REQUIRED

To build our project, we need the following components:

- Arduino Uno (2 pcs.), Pushbuttons (2 pcs.); Breadboard ; Jumper wires

5.1 CONNECTION DIAGRAM

Figure 4 shows how to connect the components used in our project.

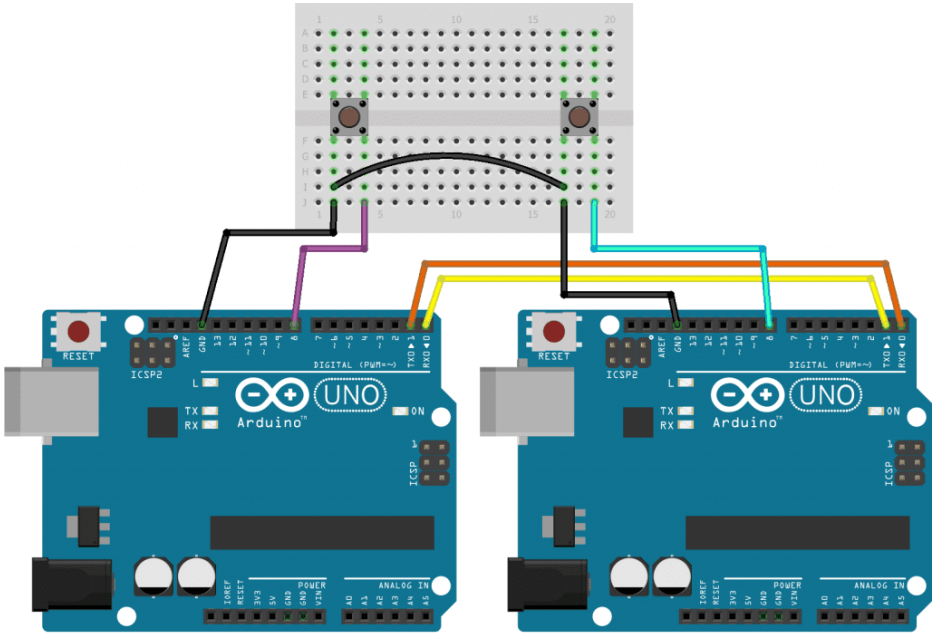

Figure 4: Connection Diagram

5.2 ARDUINO SKETCH

After gathering and assembling the hardware, we are now ready to program our boards. For this project, both boards will have identical sketches. First, we set **pin 8** (push button) pin mode to INPUT_PULLUP, set **pin 13** (LED) pin mode to OUTPUT and set the initial state of **pin 13** to LOW (LED off).

```
// Arduino UART Tutorial

void setup() {
  pinMode(8, INPUT_PULLUP); // set push button pin as input
  pinMode(13, OUTPUT);      // set LED pin as output
  digitalWrite(13, LOW);    // switch off LED pin
}

void loop() {

}
```

5.3 SERIAL OBJECT

As always, the Arduino makes it easy for us to use the built-in UART hardware by using the serial object. The serial object has the necessary functions for an easy use of the Arduino's UART interface.

5.4 SERIAL.BEGIN()

To communicate via the UART interface, we need to configure it first. The easiest way to configure the Arduino's UART is by using the function Serial.begin(*speed*). The *speed* parameter is the baud rate that we want the UART to run. Using this function will set the remaining UART parameters to default values (*Data length*=8, *Parity bit*=1, *Number of Stop Bits=None*).

If the default settings don't work for you, use the function Serial.begin(*speed,config*) instead of Serial.begin(*speed*). The additional parameter *config* is used to change the settings for data length, parity bit, number of stop bits. Defined values for the parameter *config* can be found here.

The code below adds Serial.begin(9600); inside setup() to initialize the Arduino Uno UART with a baud rate of 9600 bps and other parameters set to default values.

```
// Arduino UART Tutorial

void setup() {
  pinMode(8, INPUT_PULLUP); // set push button pin as input
  pinMode(13, OUTPUT);      // set LED pin as output
  digitalWrite(13, LOW);    // switch off LED pin

  Serial.begin(9600);       // initialize UART with baud rate of 9600 bps
}

void loop() {

}
```

The next part to code is to read and save a value received from the serial. To do this, we will use the function Serial.available() together with an If statement to check if there is data received. We will then call Serial.read() to get one byte of received data and save the value to the variable data_rcvd. The value of data_rcvd controls the On/Off of the built-in LED.

5.5 SERIAL.AVAILABLE()

To check if there is data waiting to be read in the UART (or serial) buffer, we will use the function Serial.available(). Serial.available() returns the number of bytes waiting in the buffer.

5.6 SERIAL.READ()

To read the data waiting in the serial buffer, we will use the function Serial.read(). This function returns one byte of data read from the buffer.

```
// Arduino UART Tutorial

void setup() {
  pinMode(8, INPUT_PULLUP); // set push button pin as input
  pinMode(13, OUTPUT);      // set LED pin as output
  digitalWrite(13, LOW);    // switch off LED pin

  Serial.begin(9600);       // initialize UART with baud rate of 9600 bps
}

void loop() {
  if(Serial.available()) {
    char data_rcvd = Serial.read();   // read one byte from serial buffer and save to data_rcvd

    if(data_rcvd == '1') digitalWrite(13, HIGH);  // switch LED On
    if(data_rcvd == '0') digitalWrite(13, LOW);   // switch LED Off
  }
}
```

5.7 SERIAL.WRITE()

For us to send data via Arduino's TX0 pins, we will use the function Serial.write(*val*). The *val* parameter is the byte (or series of bytes) to be sent.

In our sketch, we will send a char value depending on the state of **pin 8**. We will send the char value of '1' if **pin 8** is HIGH or a char value of '0' if the **pin 8** is LOW.

```
// Arduino UART Tutorial

void setup() {
  pinMode(8, INPUT_PULLUP); // set push button pin as input
  pinMode(13, OUTPUT);      // set LED pin as output
  digitalWrite(13, LOW);    // switch off LED pin

  Serial.begin(9600);       // initialize UART with baud rate of 9600 bps
}

void loop() {
  if (Serial.available()) {
    char data_rcvd = Serial.read();   // read one byte from serial buffer and save to data_rcvd
```

```
  if (data_rcvd == '1') digitalWrite(13, HIGH); // switch LED On
  if (data_rcvd == '0') digitalWrite(13, LOW);  // switch LED Off
 }

 if (digitalRead(8) == HIGH) Serial.write('0');    // send the char '0' to serial if button is not pressed.
 else Serial.write('1');                           // send the char '1' to serial if button is pressed.
}
```

5.8 SKETCH UPLOAD AND TESTING

Save the sketch as *arduino_uart_tutorial.ino*. The remaining step is to upload the sketch to both Arduino Uno boards. Please remember to disconnect the wires connected to **TX0** and **RX0** pins before uploading the sketch. After uploading successfully, reconnect the wires on **TX0** and **RX0** pins.

After uploading, we can control the built-in LED of one of the boards by pressing the push button connected to the other board. Our sketch checks the button state and sends a corresponding char value of'0' or '1' to the other board to control the built-in LED.

6. Applications

6.1 sing Bluetooth HC-05 module fro wireless transmission

For this module we made two example, controlling the Arduino using a smartphone and controlling the Arduino using a laptop or a PC. In order not to overload this tutorial, in my next tutorial we will learn how we can configure the HC-05 Bluetooth module and make a Bluetooth communication between two separate Arduino Boards as master and slave devices. Before we start with the first example, controlling an Arduino using a smartphone, let's take a closer look at the HC-05 Bluetooth module. Comparing it to the HC-06 module, which can only be set as a Slave, the HC-05 can be set as Master as well which enables making a communication between two separate Arduino Boards. There are several different versions of this this module but I recommend the one that comes on a breakout board because in that way it's much easier to be connected. The HC-05 module is a Bluetooth SPP (Serial Port Protocol) module, which means it communicates with the Arduino via the Serial Communication.

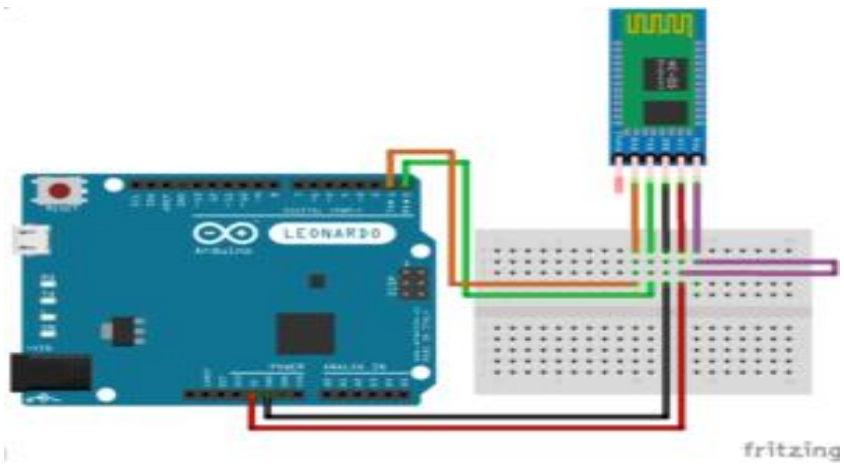

Figure 5: ARDUINO Bluetooth connections

So, now we are ready to make the Arduino code for enabling the communication between the Arduino board and the smartphone. We will make a simple example, just turning on and off a LED but it will be good enough for understanding the communication.

```
1. #define ledPin 7
2. int state = 0;
3.
4. void setup() {
5. pinMode(ledPin, OUTPUT);
6. digitalWrite(ledPin, LOW);
7. Serial.begin(38400); // Default communication rate of the Bluetooth module
8. }
9.
10. void loop() {
11. if(Serial.available() > 0){ // Checks whether data is comming from the serial port
12. state = Serial.read(); // Reads the data from the serial port
13. }
14.
15. if (state == '0') {
16. digitalWrite(ledPin, LOW); // Turn LED OFF
17. Serial.println("LED: OFF"); // Send back, to the phone, the String "LED: ON"
18. state = 0;
19. }
20. else if (state == '1') {
21. digitalWrite(ledPin, HIGH);
22. Serial.println("LED: ON");;
23. state = 0;
24. }
25. }
```

6.2 Using the UART to Communicate with Other Devices Arduino-RPI [3]

we can't use the */dev/serial0* UART to communicate with other devices if it's tied to the console. Luckily, there is a way to unlink this port from the Linux console. Just we follow these steps:

1. On terminal, we run *sudo raspi-config*

2. we Select *8 Advanced Options*

3. then we select *A8 Serial*

4. and we choose <*No*> when asked "*Would you like a login shell to be accessible over serial?*"

5. finally, we choose <*Ok*> and reboot

If we followed the steps above correctly, we can now use the Raspberry Pi UART to talk to microcontrollers, GPS devices or other serial-enabled peripherals!

Here's a simple python script that sends "Hello" to an Arduino UNO via serial.

```
#!/usr/bin/env python

import time
import serial

ser = serial.Serial(
    port='/dev/serial0',
```

```
    baudrate = 9600,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
    timeout=1
)

while 1:
    ser.write('Hello\n')
    time.sleep(1)
```

Here I used Python's serial library which is one of the built-in libraries. The serial port is initialized through the lines:

```
1  ser = serial.Serial(
2      port='/dev/serial0',
3      baudrate = 9600,
4      parity=serial.PARITY_NONE,
5      stopbits=serial.STOPBITS_ONE,
6      bytesize=serial.EIGHTBITS,
7      timeout=1
8  )
```

Here the Raspberry Pi serial port parameters are specified.

Writing to the serial port simply requires the .write() function:

```
1 ser.write('Hello\n')
```

I also added a line feed character (\n) at the end of "Hello".

Note that we need a logic level converter for this because of the different voltage levels! Follow this wiring diagram:

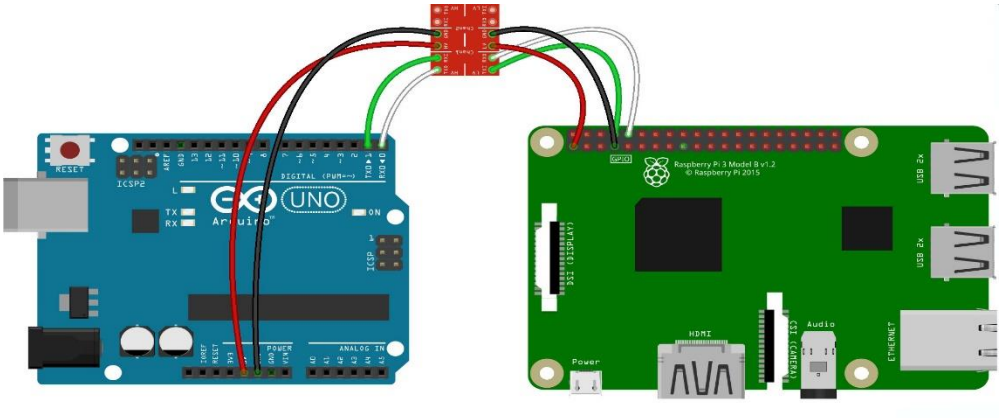

Figure 6: raspberry Pi and Arduino interconnection in serial mode UART

6.2 GPS with Arduino in serial connection

Once we have gotten the GPS module tested with direct wiring, we can go forward and wire it up to a microcontroller. We'll be using an Arduino but you can adapt our code to any other microcontroller that can receive TTL serial at 9600 baud.

Connect **VIN** to +5V, **GND** to Ground, **RX** to digital 2 and **TX** to digital 3.

Next up, download the Adafruit GPS library. This library does a lot of the 'heavy lifting' required for receiving data from GPS modules, such as reading the streaming data in a background interrupt and auto-magically parsing it. To download it, visit the GitHub repository [5].

Open up the **File→Examples→Adafruit_GPS→echo** sketch and upload it to the Arduino. Then open up the serial monitor. This sketch simply reads data from the software serial port (pins 2&3) and outputs that to the hardware serial port connected to USB.

Open up the Arduino IDE Serial Console and make sure to set the Serial baud rate to **115200**

You can configure the GPS output you see by commenting/uncommenting lines in the **setup()** procedure. For example, we can ask the GPS to send different sentences, and change how often it sends data. 10 Hz (10 times a second) is the max speed, and is a lot of data. You may not be able to output "all data" at that speed because the 9600 baud rate is not fast enough.

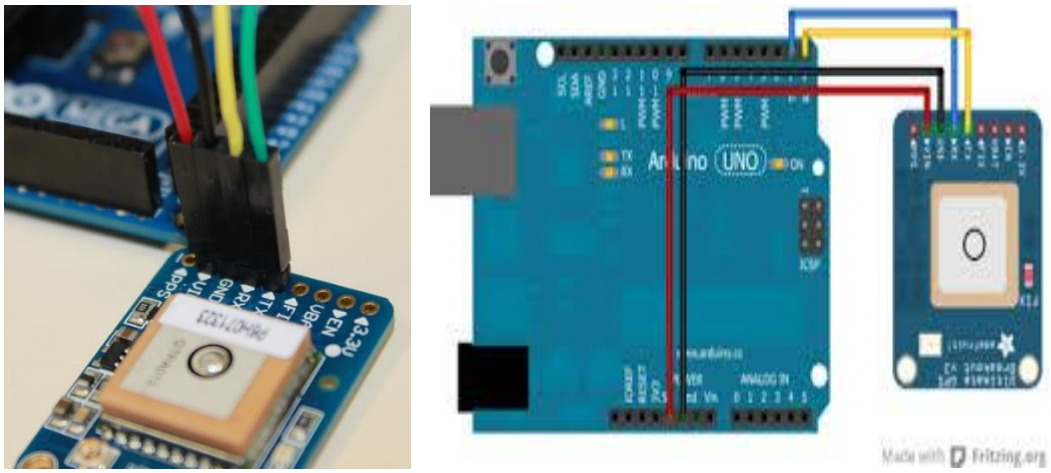

Figure 7. Connection GPS sensor in Serial mode using UARt.

The code

```
#include <TinyGPS.h> // La bibliothèque TinyGPS va analyser les trames GPS

TinyGPS gps; // Création de l'objet GPS
int a;
float lat, lon;

void setup() {
  Serial.begin(9600);
 }

void loop() {
    if (Serial.available()){
      a = Serial.read();
      // Serial.write(a); // Pour Debuger et voir les trames GPS
      if (gps.encode(a)) lireGPS();   // Si TinyGPS a découvert une nouvelle trame, afficher les coordonnées
          }
  }

void lireGPS() {
      gps.f_get_position(&lat, &lon);   // appel fonction de la bibliothèque TinyGPS
      Serial.println();                 // Sauter une ligne
      Serial.print(lat,5);              // voir les coordonnées GPS
      Serial.print(",");Serial.println(lon,5);
    }  // D'autres fonctions sont disponibles : lire la vitesse, la date, l'altitude...
```

references

[1] https://www.circuitbasics.com/how-to-set-up-uart-communication-for-arduino/

[2] http://tpil.projet.free.fr/TP_Arduino/01_UART.html

[3] https://www.teachmemicro.com/raspberry-pi-serial-uart-tutorial/

[4] http://tpil.projet.free.fr/TP_Arduino/01_UART.html

[5] https://github.com/adafruit/Adafruit-GPS-Library

[6] https://howtomechatronics.com/tutorials/arduino/arduino-and-hc-05-bluetooth-module-tutorial/