Faculty of Engineering & Technology Electrical & Computer Engineering Department

Computer Design Laboratory ENCS 4110

Experiment #3

Serial Communication with Arduino

Prepared by:
Hasan Hamed 1190496

Instructor: Dr. Abualseoud Hanani
Assistant: Eng. Moutasem Diab

Section: 1
Date: 6/8/2021

1. Abstract

The aim of the experiment is to understand the concepts of serial communication in Arduino UNO board and to learn the UART communication concept and how to use it to transfer data between two connected Arduino's and finally to apply this concept in several applications such as data transferring, controlling motors and using serial plotter to see how data get transferred.

Table of Contents

2. Theory

2.1. UART

Universal asynchronous receiver-transmitter (UART) is one of the simplest and oldest forms of device-to-device digital communication. You can find UART devices as a part of integrated circuits (ICs) or as individual components. UARTs communicate between two separate nodes using a pair of wires and a common ground.

Data transmission takes place in the form of data packets, beginning with a start bit, where the ordinarily high line is pulled to ground. After the start bit, five to nine data bits transmit in what is known as the packet's data frame, followed by an optional parity bit to verify proper data transmission. Finally, one or more stop bits are transmitted where the line is set to high. This ends a packet.

As an asynchronous protocol, no clock line regulates data transmission speed and users must set both devices to communicate at the same speed. This speed is known as the baud rate, expressed in bits per second, or bps. Transmission speeds vary dramatically, from the typical 9600 baud setting to 115200 and beyond.

While something of an "ancient" protocol, and one that can only communicate between a single master and slave, UART is well known, easy to set up, and extremely versatile.[1]

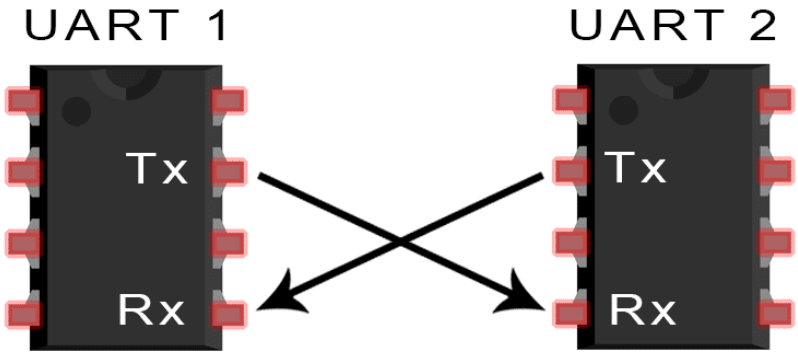

Figure 2.1: Two connected UART's.

UART transmitted data is organized into packets. Each packet contains 1 start bit, 5 to 9 data bits (depending on the UART), an optional parity bit, and 1 or 2 stop bits.

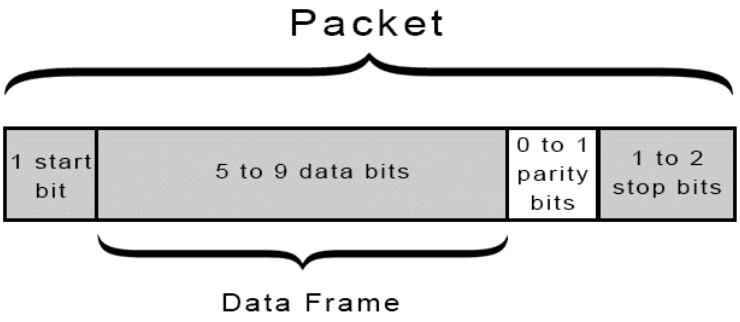

Figure 2.2: Packets Bits.

Start Bit

When there is no data transmission, the UART transmission line is held at high voltage. To initiate the data transfer, the voltage is pulled from high to low for one clock cycle. This transition acts as the start bit. And when the receiving UART detects the high to low voltage transition, it begins reading the data frame at the frequency of the baud rate.

Data frame

The data bits are usually 5 to 8 in number. If no parity bit is used, it can be 9-bit long. In general case, the least significant bit of the data is transmitted first. It is the useful data we're actually transmitting.

Parity

A parity bit is used to indicate the change in data during transmission. There are several reasons for the change in data including mismatched baud rates, electromagnetism or long-distance data transfer. After receiving the data frame, UART checks if the total number of bits in the data frame is an even or odd number. This is done by counting the number of bits with a value of 1. If the parity bit is 0, the number of bits in the data frame is an even number. If the parity bit is 1, the number of bits in the data frame is odd. If the parity bit does not match with the data, UART identifies an error in the data frame.

Stop bit

To mark the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage for a minimum of two-bit duration.

UART does not process the start, parity and stop bits and it automatically discards these bits, no matter the data was received correctly or not. Since the asynchronous data transmission is 'self-synchronizing', if no data is transmitted at a particular time, the transmission line stays idle.[2]

2.2. Serial communication with Arduino [Tx, Rx]

On Uno, Nano, Mini, and Mega, pins 0 and 1 are used for communication with the computer. Connecting anything to these pins can interfere with that communication, including causing failed uploads to the board.

You can use the Arduino environment's built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to begin().

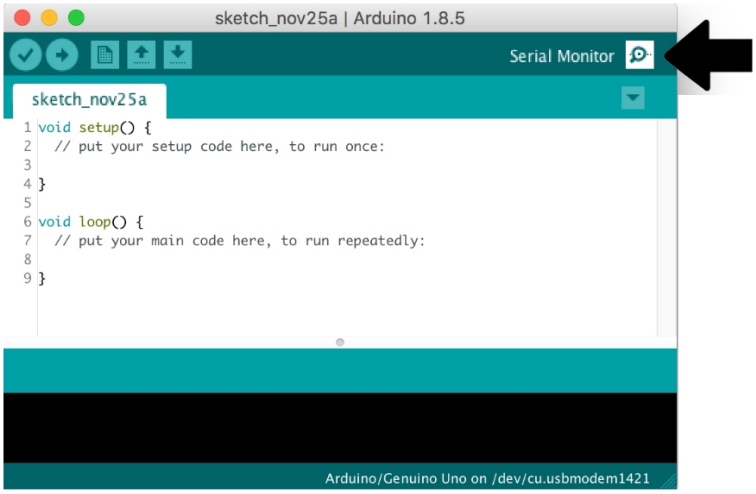

Figure 2.3: Serial Monitor.

Serial communication on pins TX/RX uses TTL logic levels (5V or 3.3V depending on the board). Don't connect these pins directly to an RS232 serial port; they operate at +/- 12V and can damage your Arduino board.

To use these extra serial ports to communicate with your personal computer, you will need an additional USB-to-serial adaptor, as they are not connected to the Mega's USB-to-serial adaptor. To use them to communicate with an external TTL serial device, connect the TX pin to your device's RX pin, the RX to your device's TX pin, and the ground of your Mega to your device's ground.

In order to handle the serial communication, a lot of built-in functions are implemented to make your life easier. The following functions are used mainly.

Serial.available()

Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer.

Serial.begin()

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with Serial Monitor, make sure to use one of the baud rates listed in the menu at the bottom right corner of its screen. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate. [Must be used in setup()]

An optional second argument configures the data, parity, and stop bits. The default is 8 data bits, no parity, one stop bit. Syntax:

Serial.begin(speed)

Serial.begin(speed, config)

Where:

speed: in bits per second (baud). Allowed data types: long. [ex: 4800, 9600, 19200]

config: sets data, parity, and stop bits. Valid values are:

No Parity:	Even Parity:	Odd Parity:
SERIAL_5N1	SERIAL_5E1	SERIAL_5O1
SERIAL_6N1	SERIAL_6E1	SERIAL_6O1
SERIAL_7N1	SERIAL_7E1	SERIAL_7O1
SERIAL_8N1 (the default)	SERIAL_8E1	SERIAL_8O1
SERIAL_5N2	SERIAL_5E2	SERIAL_5O2
SERIAL_6N2	SERIAL_6E2	SERIAL_6O2
SERIAL_7N2	SERIAL_7E2	SERIAL_7O2
SERIAL_8N2	SERIAL_8E2	SERIAL_8O2

Figure 2.4: Valid Values for Configure Argument

Read Data:

Serial.read()

Reads incoming serial data. Returns: The first byte of incoming serial data available (or -1 if no data is available). Data type: int.

Serial.readBytes(buffer, length)

Reads characters from the serial port into a buffer. The function terminates if the determined length has been read, or it times out. Returns: The number of bytes placed in the buffer. Data type: size_t.

Serial.readString()

Reads characters from the serial buffer into a String. The function terminates if it times out. Returns: A String read from the serial buffer.

Serial.parseInt()

Looks for the next valid integer in the incoming serial.

Serial.parseFloat()

returns the first valid floating-point number from the Serial buffer. It is terminated by the first character that is not a floating-point number.

Write Data:

Serial.print()

Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is.

Serial.write()

Writes binary data to the serial port. This data is sent as a byte or series of bytes; to send the characters representing the digits of a number use the print() function instead.[3]

3. Procedure

3.1. Basic communication between Arduino & PC.

1) An Arduino UNO were used to communicate between the PC and the Arduino using Tinkercard website.

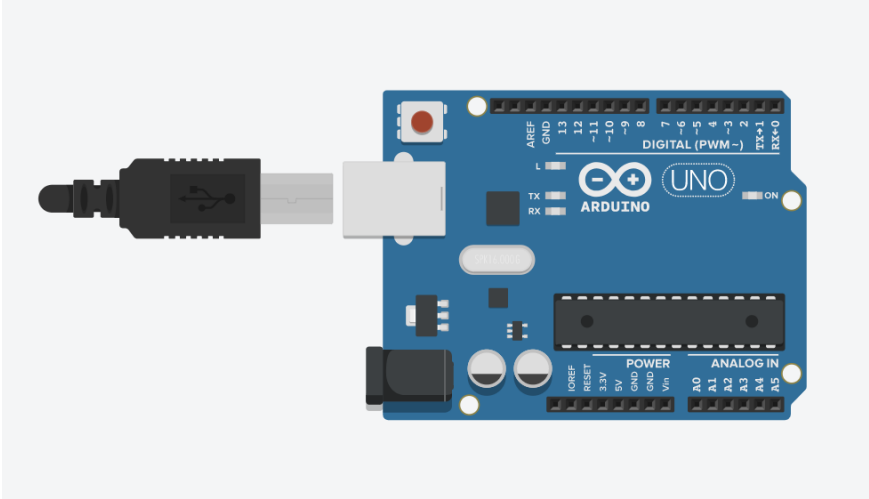

Figure 3.1: Arduino UNO board in Tinkercad.

2) The following code was used to run the Arduino.

```
/* Use a variable called byteRead to temporarily store
   the data coming from the computer */
byte byteRead;

void setup() {
// Turn the Serial Protocol ON
  Serial.begin(9600);
}

void loop() {
   /*  check if data has been sent from the computer: */
  if (Serial.available()) {
    /* read the most recent byte */
    byteRead = Serial.read();
    /*ECHO the value that was read, back to the serial port. */
    Serial.write(byteRead);
  }
}
```

Figure 3.2: Basic Communication Code.

3) The simulation was started using Start Simulation button.

This code will receive data from the serial monitor in the Tinkecad and then write it back to the serial port and print it on the screen as shown below.

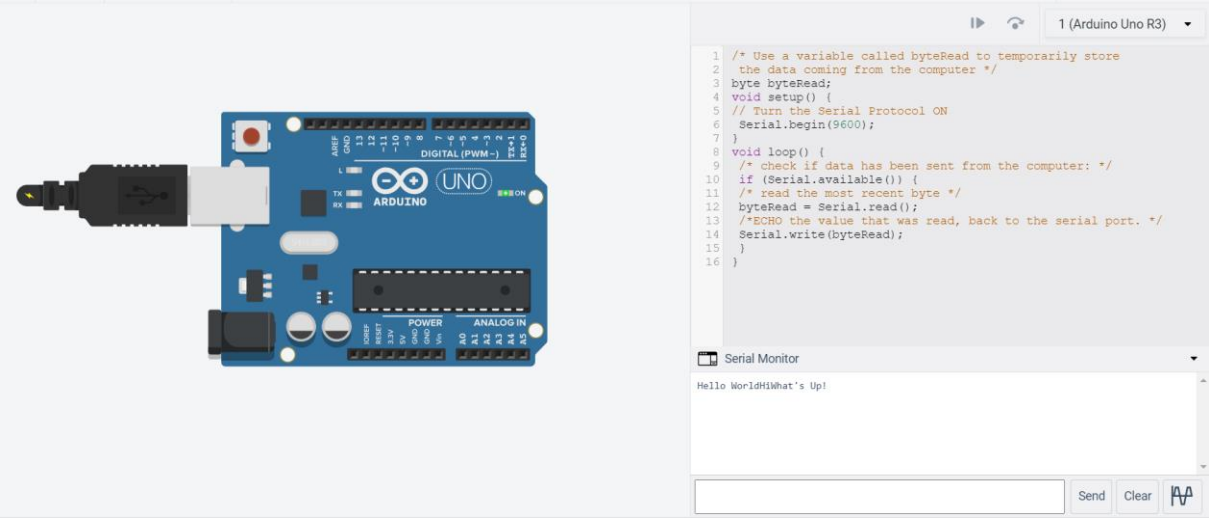

Figure 3.3: Simulation Result.

3.2. Basic communication between 2 Arduinos.

1) The following circuit was implemented using Tinkercad.

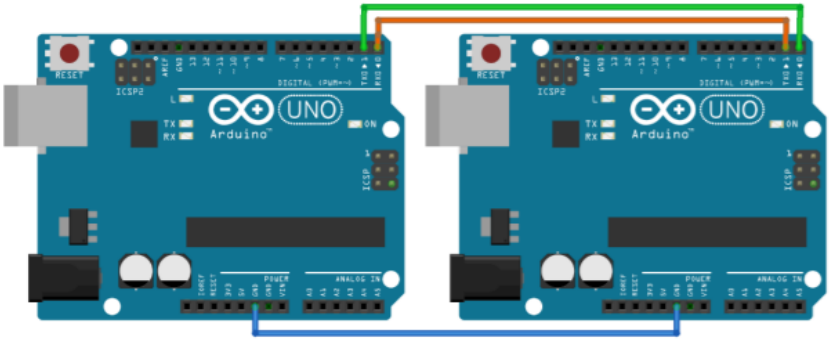

Figure 3.4: Connecting two Arduinos using TX & RX.

2) Two codes were used for both the Sender and Receiver Arduinos.

```
void setup() {
  // Setup the Serial at 9600 Baud
  Serial.begin(9600);
}

void loop() {
  Serial.println("Hello World!"); //Write the serial data
  delay(1000);
}
```

Figure 3.5: Sender Code.

```
void setup() {
  // Begin the Serial at 9600 Baud
  Serial.begin(9600);
}

void loop() {
  if (Serial.available()) {
    String data = Serial.readString();//Read the serial data and store in var
    Serial.println(data);//Print data on Serial Monitor
  }
```

Figure 3.6: Receiver Code.

3) The simulation was started using Start Simulation button.

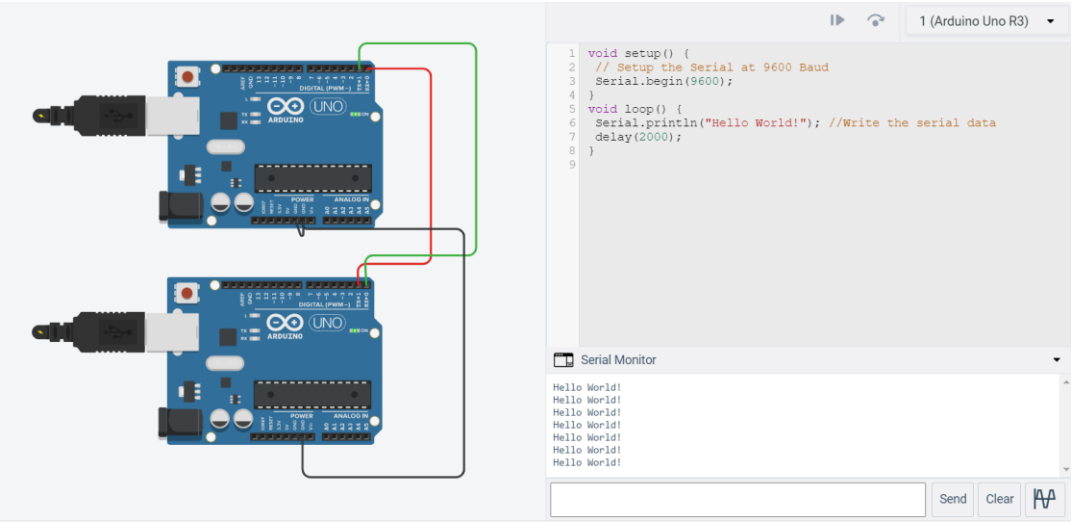

Figure 3.7: Sender Arduino Monitor.

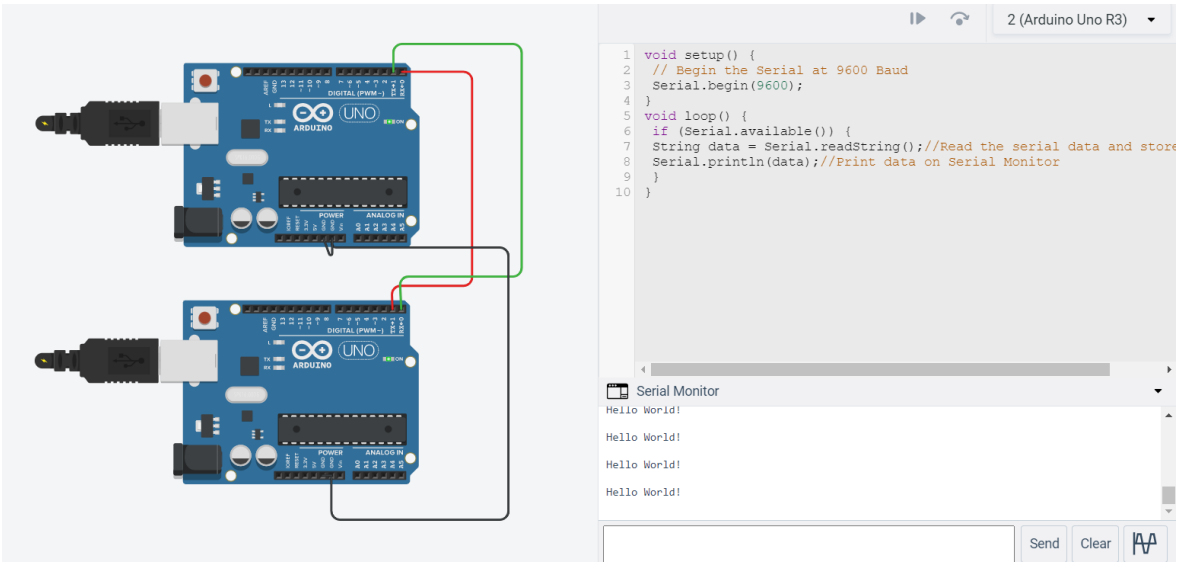

Figure 3.8: Receiver Arduino Monitor.

The sender Arduino will be responsible of sending “Hello World!” to the receiver Arduino after specifying serial communication Setup [Baud rate, stop bits, …]. The receiver Arduino will be responsible of receiving the data and then displaying it. First, we should specify the serial communication Setup [Baud rate, stop bits, …] and then read the data.

3.3. Push Button & LED using 2 Arduinos.

1) The Push button was connected to Arduino 1 as shown.

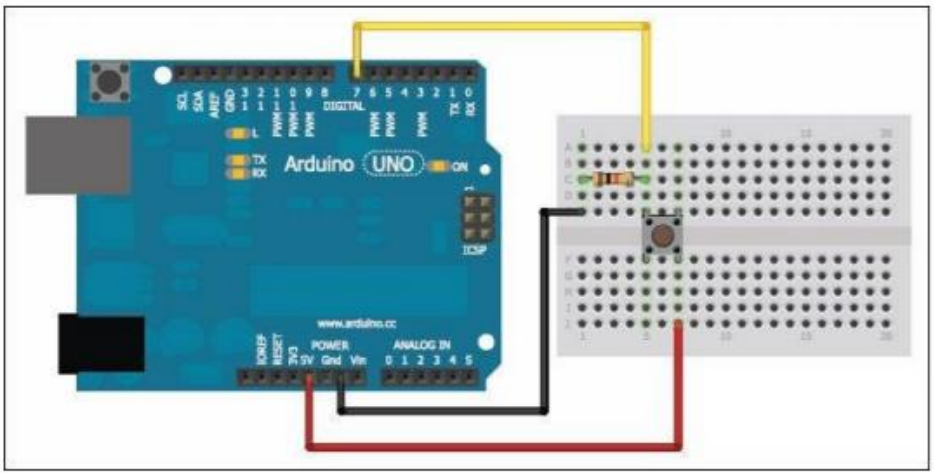

Figure 3.9: Push button connected to Arduino 1.

2) The LED was connected to Arduino 2 as shown.

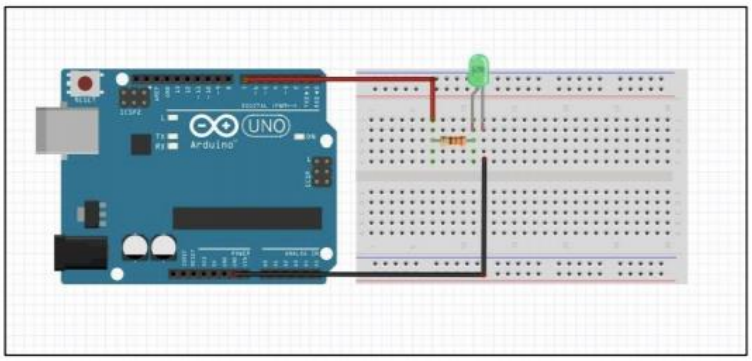

Figure 3.10: LED Connected to Arduino 2.

3) The following code was uploaded to Arduino 1.

```
int pin =8;
int n;
void setup(){
pinMode(pin, INPUT);
Serial.begin(9600);
}
void loop() {
n = digitalRead(8);
if(n == HIGH ){
Serial.println('1');
}
delay(2000);
}
```

Figure 3.11: Arduino 1 Code.

3) The following code was uploaded to Arduino 2.

```
int pinn =8;
void setup() {
Serial.begin(9600);
pinMode(pinn, OUTPUT);}
void loop() {
if (Serial.available()){
char data = Serial.read();
Serial.println(data);
if( data == '1'){
digitalWrite(pinn,HIGH);
}
delay(500);
digitalWrite(pinn,LOW);
}
}
```

Figure 3.12: Arduino 2 Code.

4) The simulation was started using the Start Simulation Button.

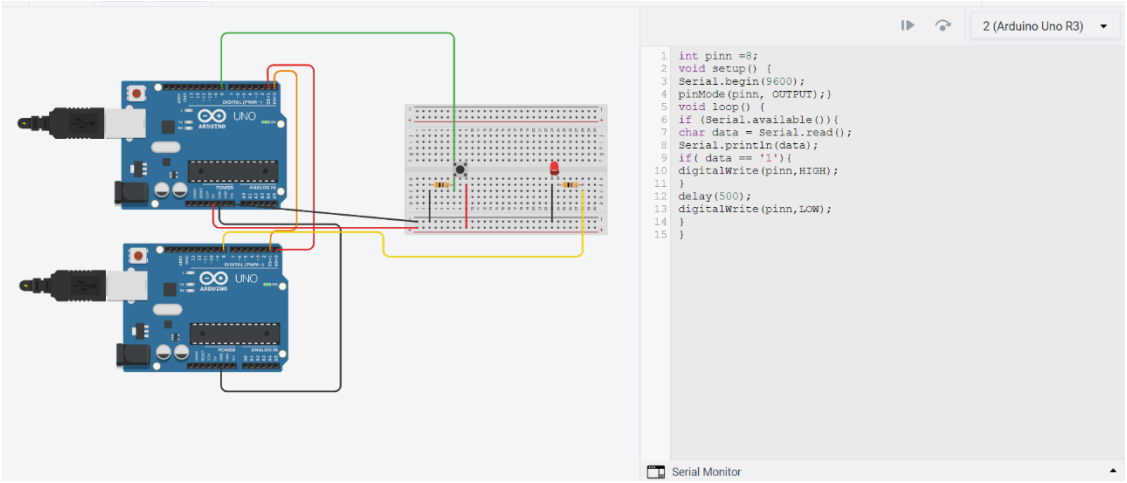

Figure 3.13: Simulation Result.

Arduino 1 will send the value '1' whenever the push button is pushed, and then Arduino 2 will receive the data and turn on the LED whenever logic '1' is read.

3.4. Visualization of serial communication using Serial Plotter.

1) The following circuit was implemented using Tinkercad.

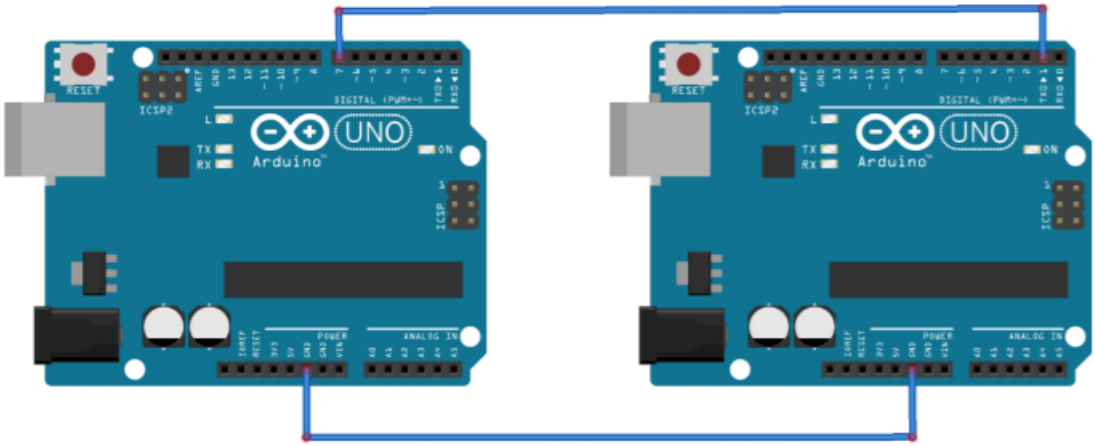

Figure 3.14: Two Connected Arduinos.

2) The Sender code was uploaded to the first Arduino.

```
void setup() {
  // Setup the Serial at 300 Baud
  Serial.begin(300);
}

void loop() {
  Serial.write('A'); //Write the character A => will be transmitted as Byte [41H]
  delay(500);
}
```

Figure 3.15: Sender Code.

3) The Receiver code was uploaded to the second Arduino.

```
int readPin = 7;
void setup() {
  // Begin the Serial at 19200 Baud
  Serial.begin(19200);
  pinMode(readPin, INPUT);
}

void loop() {
  Serial.println(digitalRead(readPin));
}
```

Figure 3.16: Receiver Code.

4) The simulation was started using the Start Simulation button.

The sender will be responsible of sending a character to the receiver Arduino. First, we have to setup the Baud Rate to be very small [for example, 300] and then sending a character with a delay.

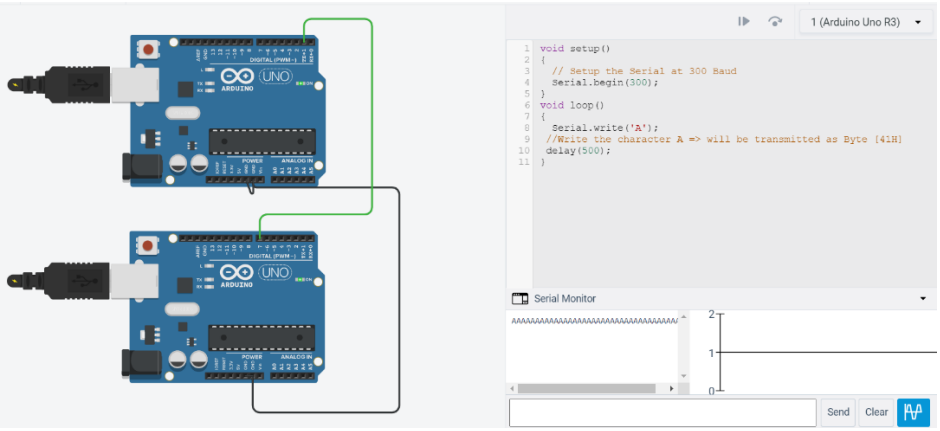

Figure 3.17: Sender Arduino Screen.

The receiver will be responsible of reading the data sent using digitalRead with one of the digital pins in order to plot the serial data as bits. A high Baud rate was used in order to capture the bits.

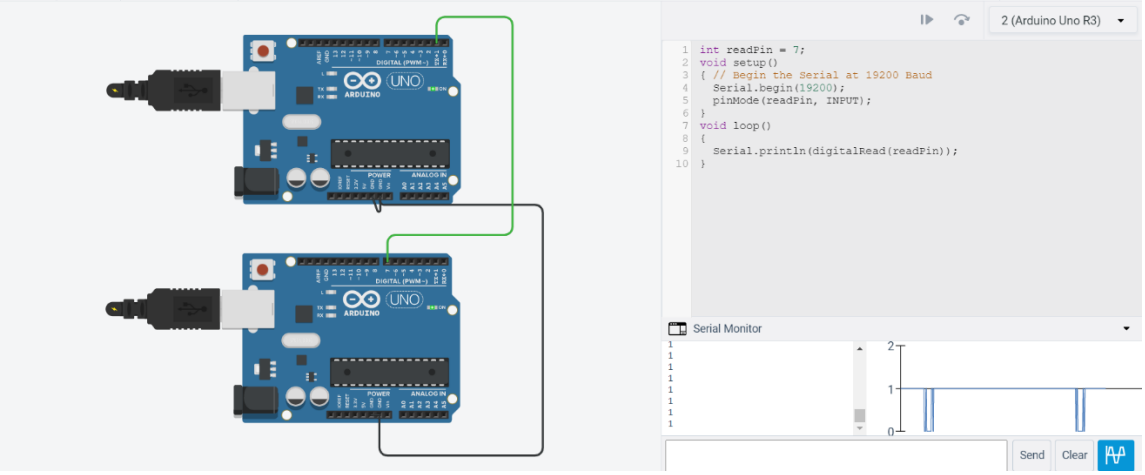

Figure 3.18: Receiver Arduino Screen.

4. Conclusion

In conclusion, we understood how the UART concept works and how we can use it in serial transfer for data between two Arduino's, and we understood that we can use Arduino serial monitor communicate between the Arduino and the PC and to graph how data are transferred using serial plotter and finally, we understood the importance of the serial functions and how we can use it in a lot of real-life applications.

5. References

[1] https://www.arrow.com/en/research-and-events/articles/what-is-uart-protocol-uart-communication-explained

[2] https://openlabpro.com/guide/basics-of-uart/

[3] Computer Design Laboratory Manual