Experiment #3
Serial Communication with Arduino

Birzeit University

Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

Objectives:

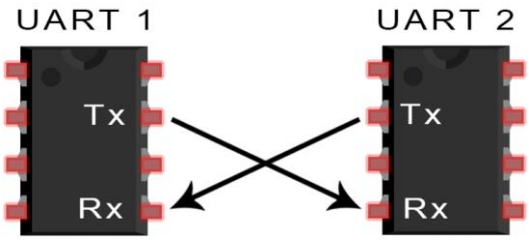
The objectives of this experiment are getting to know the basic concepts of serial communication and to practically apply it using Arduino.

Theoretical Introduction

I. UART

UART stands for Universal Asynchronous Receiver/Transmitter. It's not a communication protocol like SPI and I2C, but a physical circuit in a microcontroller, or a stand-alone IC. **A UART's main purpose is to transmit and receive serial data.**

In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. **Only two wires are needed to transmit data between two UARTs**. Data flows from the **Tx** pin of the transmitting UART to the **Rx** pin of the receiving UART.



UARTs transmit data **asynchronously**, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds **start and stop bits** to the data packet being transferred. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits.

When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the baud rate. **Baud rate** is a measure of the speed of data transfer, expressed in bits per second (bps). **Both** UARTs must operate at about the same baud rate.

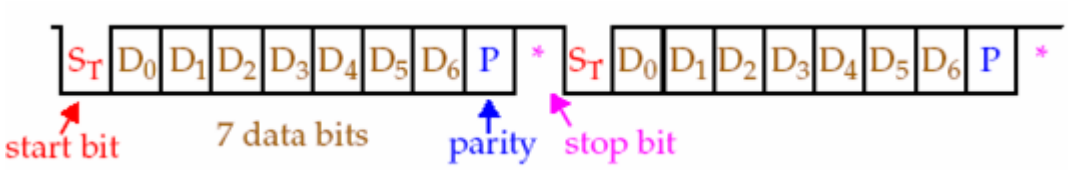


Figure 1: Serial Communication

- **START BIT**

The UART data transmission line is normally held at a high voltage level when it's not transmitting data. To start the transfer of data**, the transmitting UART pulls the transmission line from high to low for one clock cycle.** When the receiving UART detects the high to low voltage transition, it begins reading the bits in the data frame at the frequency of the baud rate.

- **DATA FRAME**

The data frame contains the actual data being transferred. It can be **5 bits up to 8 bits** long if a parity bit is used. If no parity bit is used, the data frame can be 9 bits long. In most cases, the data is sent with the least significant bit first.

- **PARITY**

Parity describes the **evenness** or **oddness** of a number. The parity bit is a way for the receiving UART to tell if any data has changed during transmission. Bits can be changed by electromagnetic radiation, mismatched baud rates, or long-distance data transfers. After the receiving UART reads the data frame, it counts the number of bits with a value of 1 and checks if the total is an even or odd number. If the parity bit is a 0 (even parity), the 1-bits in the data frame should total to an even number. If the parity bit is a 1 (odd parity), the 1-bits in the data frame should total to an odd number. When the parity bit matches the data, the UART knows that the transmission was free of errors. But if the parity bit is a 0, and the total is odd; or the parity bit is a 1, and the total is even, the UART knows that bits in the data frame have changed.

- **STOP BITS**

To signal the end of the data packet**, the sending UART drives the data transmission line from a low voltage to a high voltage for at least two-bit durations.**

II. Serial communication with Arduino [T_x, R_x]

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.

Serial is used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART): Serial. It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output.

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().

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. For extra information, please refer to this link.

- *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

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.

Procedure

The following parts are different applications on using Serial communication between 2 devices.

Part1: Basic communication between Arduino & PC

The idea of this part is sending a simple information between the Arduino & PC [“Hello World!”] in order to become familiar with the connections & the setup.

I. Required Components:

- PC
- Arduino UNO Board
- USB Cable

II. Circuit

III. Code

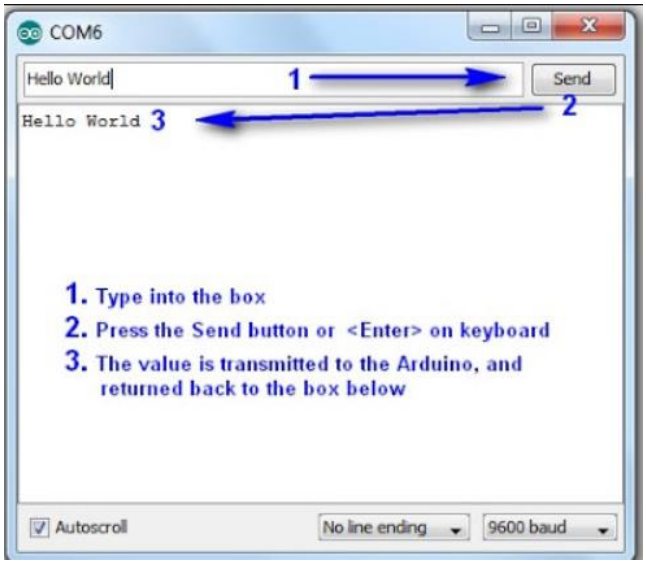
The following code will receive data from the PC and then write it back to the serial port.

```
/* 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);
  }
}
```

IV. Download & Run

- Connect the Arduino with the PC using the USB cable and then download the code.
- Open the serial monitor.

- Send a value to Arduino as seen in the following screenshot



Part2: Basic communication between 2 Arduinos

The idea of this part is sending a simple information between the 2 Arduinos [“Hello World!”] in order to become familiar with the connections & the setup.

I. Required Components:

- Arduino UNO Board X 2
- Jumper Wires

II. Circuit

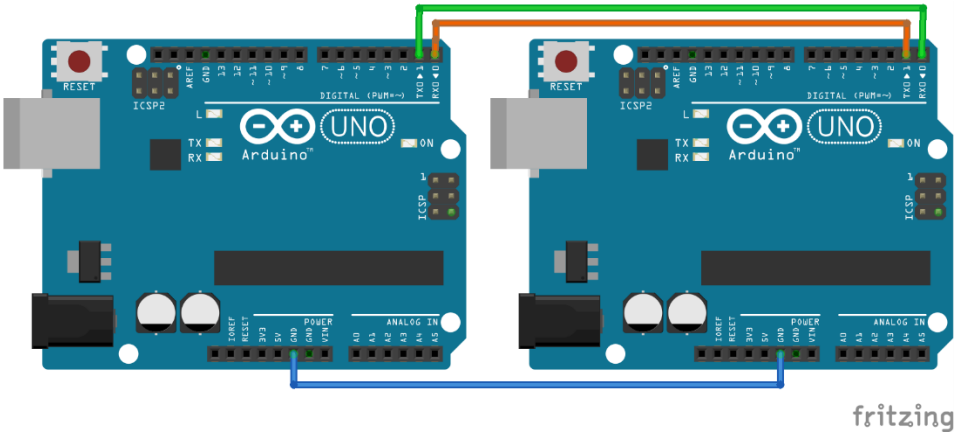


Figure 2: Connecting 2 Arduinos using TX & RX

We connect **TX** in the First Arduino with **RX** in the second one and Vice versa [**$TX_1 \leftrightarrow RX_2$ & $RX_1 \leftrightarrow TX_2$**]. The ground must be common between both of them.

III. Code

Since we are dealing with 2 Arduinos, a code for each is needed.

Sender Code:

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

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

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

Receiver Code:

The receiver 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!

```
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
  }
}
```

IV. Download & Run

- Download Both codes on the Transmitter & Receiver Arduinos.

  Note: If it shows error while uploading code to Arduino. Then, disconnect the connection between Arduino and try to upload program on it.

- Open Serial monitor on the receiver and observe the printed data.

Part3: Push Button & LED using 2 Arduinos

The idea of this part is to connect a push button with **Arduino$_1$** and whenever it's pushed, the value 1 is sent to the **Arduino$_2$**. In its part, **Arduino$_2$** receives the data and whenever logic 1 is read, a LED will be turned on.

I. Required Components:

- Arduino UNO Board X 2
- Push Button
- LED
- Resistance X 2

II. Circuit

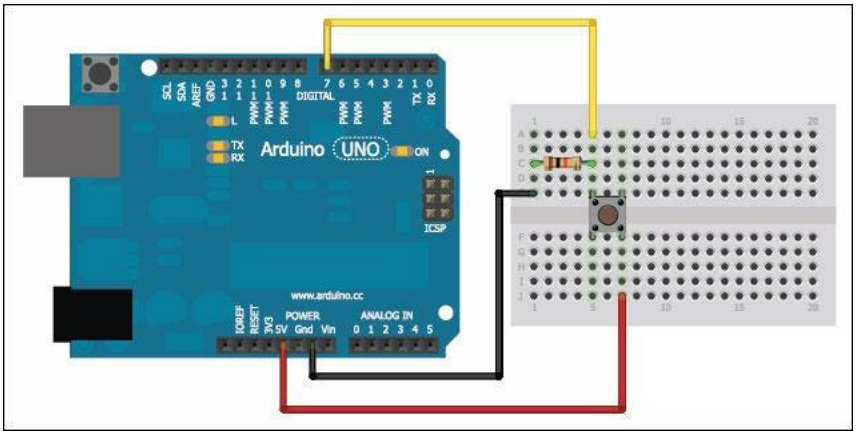


Figure 3: Arduino1 connected with Push button

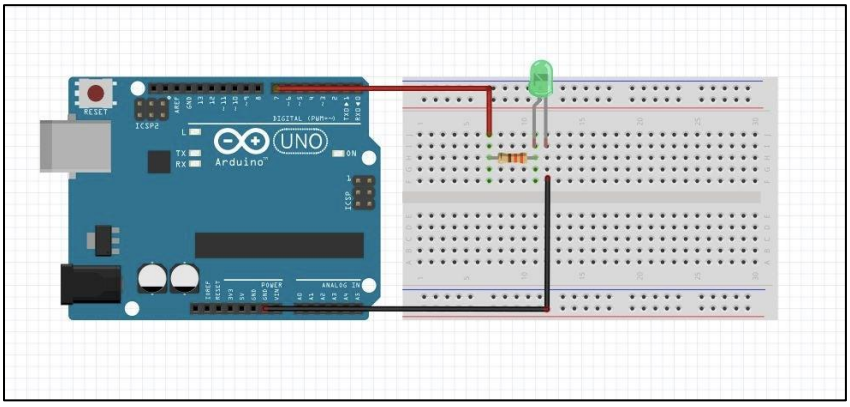


Figure 4: Arduino2 connected with Led

Note: We connect **TX** in the First Arduino with **RX** in the second one and Vice versa [**$TX_1 \leftrightarrow RX_2$ & $RX_1 \leftrightarrow TX_2$**]. The ground must be common between both of them.

III. Code

Sender Code:

Arduino$_1$ should send 1 to **Arduino$_2$** whenever the push button is pushed. If not pushed, nothing should be sent.

```
void setup() {
  // TODO[1]: Setup the Serial at 9600 Baud

}
void loop() {
  // TODO[2]: Check if the push button is pushed & Send 1 if pushed

}
```

Receiver Code:

Arduino$_2$ should receive 1 from **Arduino$_1$** whenever the push button is pushed. In this case, the Led must be turned on.

```
void setup() {
  // TODO[1]: Setup the Serial at 9600 Baud

}
void loop() {
/* Check if data has been sent from the computer: */
  if (Serial.available()) {
  // TODO[2]: Check if 1 is sent, and turn on the LED if yes.

  }
}
```

IV. Download & run

- Download Both codes on the Transmitter & Receiver Arduinos.
- Open the Serial monitor to make sure that the receiver sends the data correctly.

Part4: Visualization of serial communication using **Serial Plotter**

The idea of this part is to see how the serial data are transferred as start-data-parity-stop bits. This can be done using Serial plotter in Arduino IDE.

I. Required Components:

- Arduino UNO Board X 2
- PC
- USB Cable
- Wires

II. Circuit

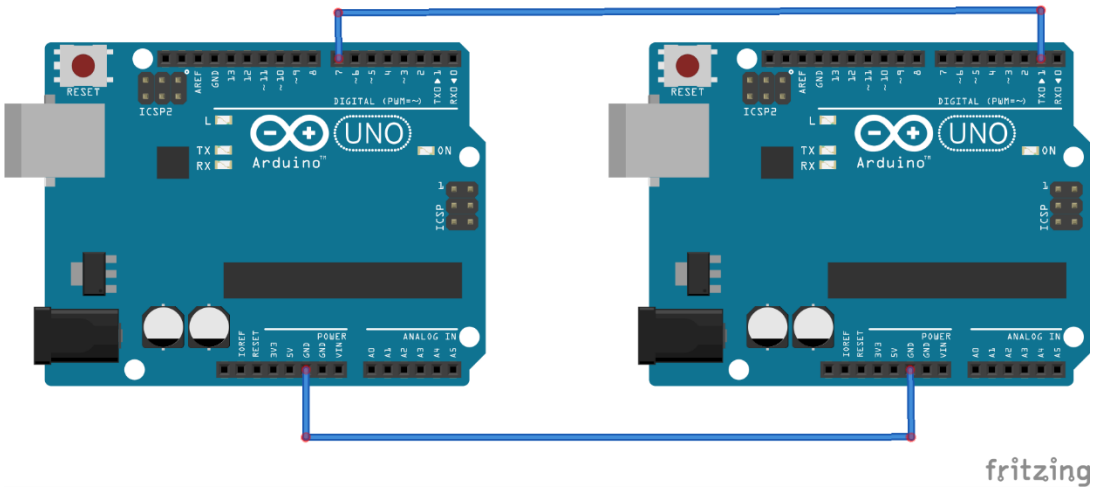


Figure 5: Arduino1 TX connected with Arduino2 PIN_7 with common GND

III. Code

Sender Code:

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.

```
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);
}
```

Receiver Code:

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.

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

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

IV. Download & run

- Download Both codes on the Transmitter & Receiver Arduinos.
- Open the Serial Plotter [Tools -> Serial Plotter] in the Receiver to observe the bits received.

V. Tasks

- Draw the theoretical waveform for the start-data-parity-stop bits and compare it with what is plotted.
- Try Sending different data. [Read the data from the user]
- Try changing the serial setup:
    - Even Parity
    - Odd Parity

References:

- https://www.arduino.cc/reference/en/language/functions/communication/serial/