ECOLE NATIONALE
SUPÉRIEURE
D'INFORMATIQUE

الجـــمهوريـــة الجزائريـــة الديمـــقراطيـــة الشـعبيـة
وزارة التعليـــم العـــالي والبـــحث العلمـــي
People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

**Intelligent and Communicating Systems, ICS**

$2^{nd}$ Year Specialty SIQ G02, 2CS SIQ2

# SIMULATION

**Title:**

# Arduino-Raspberry Wired Communications UART

**Studied by:**

**First Name:** Nour el Imane
**Last Name:** HEDDADJI
**E-mail:** jn_heddadji@esi.dz

# Activity

## 1. Simulation with Proteus

### 1.1. Raspberry-LED-push Button

#### Hardware

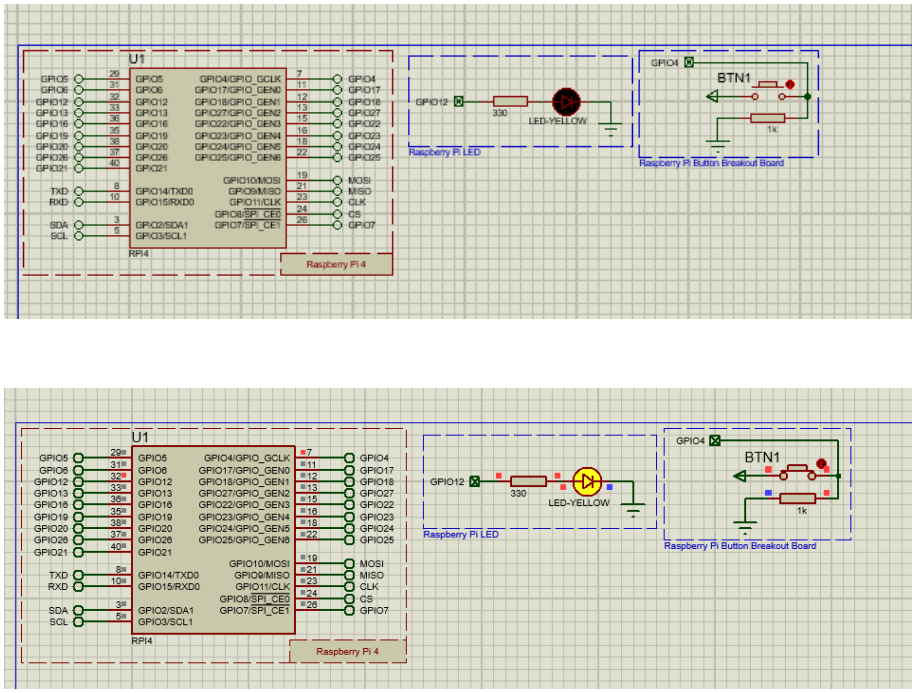



Figure 1: Raspberry-LED-push Button

#### Software

```python
# imports
import RPi.GPIO as GPIO
import time

LED_PIN = 23
PUSH_BUTTON = 24
VALUE_PUSH_BUTTON = 0

GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_PIN, GPIO.OUT)
GPIO.setup(PUSH_BUTTON, GPIO.IN)  # set the button as an input

try:
    while True:
        # Read the button state
```

```
        VALUE_PUSH_BUTTON = GPIO.input(PUSH_BUTTON)
        if VALUE_PUSH_BUTTON == GPIO.HIGH:
            GPIO.output(LED_PIN, GPIO.HIGH)
        else:
            GPIO.output(LED_PIN, GPIO.LOW)

except KeyboardInterrupt:
    GPIO.cleanup()
```

Listing 1: LED with Push Button program

### Analysis

The code continuously checks the state of the push button, and based on whether the button is pressed or not, it toggles the state of the connected LED—turning it on when the button is pressed and turning it off when the button is not pressed.

## 1.2. Raspberry-Arduino communication via UART using -LED-Push Button

### Hardware

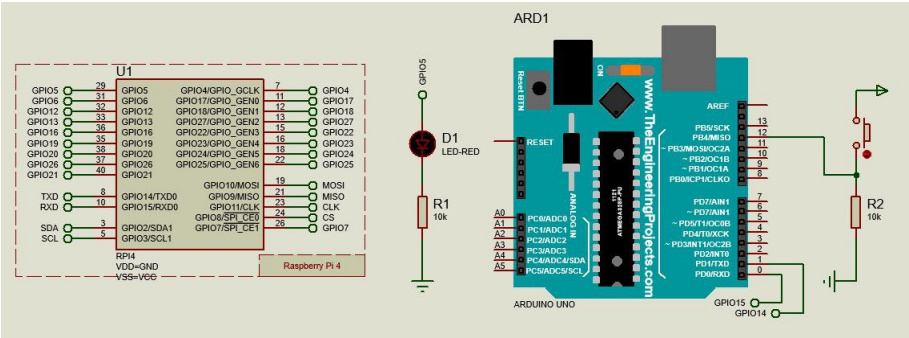



Figure 2: Raspberry-Arduino communication

### Software Arduino

```
int buttonPin = 2; // the digital pin connected to the push button
int lastButtonState = HIGH; // the previous button state

void setup() {
  pinMode(buttonPin, INPUT_PULLUP); // enable the internal pull-up resistor
   Serial.begin(9600);
   Serial1.begin(9600);
}
void loop() {
  int buttonState = digitalRead(buttonPin); // read the button state

```

```
12   if (buttonState == LOW && lastButtonState == HIGH) { // button was
      pressed
13     Serial1.println('1'); // send a message to the Raspberry Pi
14   } else {
15     Serial1.println('0');
16   }
17   lastButtonState = buttonState; // save the current button state
18   delay(100);
19 }
```

Listing 2: Arduino-Raspberry-PUSH-BUTTON-LED program

**Software Raspberry**

```
1
2  import serial
3  import RPi.GPIO as GPIO
4  LED_PIN = 18
5  GPIO.setmode(GPIO.BCM)
6  GPIO.setup(LED_PIN, GPIO.OUT)
7
8  # initiate the serial connection
9  ser = serial.Serial('/dev/ttyS0', 9600)
10
11 try:
12     while True:
13         # reading the value from the serial
14         buttonState = int(ser.readline().decode('ascii').strip())
15
16         if buttonState == 1:
17             GPIO.output(LED_PIN, GPIO.HIGH)  # turn on the LED
18         else:
19             GPIO.output(LED_PIN, GPIO.LOW)  # turn off the LED
20
21 except KeyboardInterrupt:
22     GPIO.cleanup()
23     ser.close()
```

Listing 3: Arduino-Raspberry-PUSH-BUTTON-LED program

**Analysis**

In this program, the button state is read by the Arduino board and sent to the Raspberry board through the UART connection. The raspberry then turns the LED on or off based on the button state.

# 2. Simulation with Thinkercad

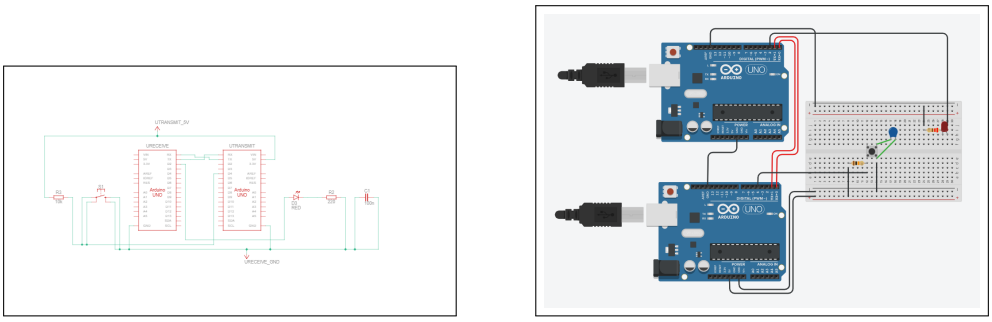
Simulation of “Arduino-Arduino communication” via UART using-LED-Push Button.

## 2.1. Components

| Name | Quantity | Component |
|---|---|---|
| UTransmit | 2 | Arduino Uno R3 |
| UReceive | 2 | Arduino Uno R3 |
| S1 | 1 | Pushbutton |
| R2 | 1 | 220 Ω Resistor |
| D3 | 1 | Red LED |
| R3 | 1 | 10 kΩ Resistor |
| C1 | 1 | 100 nF Capacitor |

Table 1: List of Components

## 2.2. Hardware



(a) Schematic view (b) Circuit view

Figure 3: Arduino-Arduino communication views

## 2.3. Software

### Sender Arduino (pushbutton)

```
// Sender Arduino
int buttonPin = 4;  // Assuming the button is connected to digital pin 4
int led = 2;
char mystring[3] = "np";
char mystring1[3] = "p";

void setup() {
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
  pinMode(led, OUTPUT);
}
void loop() {
  // Read the state of the button
  int buttonState = digitalRead(buttonPin);
  // Send the button state over UART
  if (buttonState == HIGH)
    Serial.write(mystring, 3);
  else
    Serial.write(mystring1, 3);
```

```
}
```

Listing 4: Sender Arduino program

### Receiver Arduino

```
// Receiver Arduino
int led = 2;
char mystring[15] = "np";
char mystring1[15] = "p";
char receivedString[15];

void setup() {
  Serial.begin(9600);
  pinMode(led, OUTPUT);
}

void loop() {

    // Read the string sent over UART
    Serial.readBytes(receivedString, 3);

    // Compare the received string with predefined strings
    if (strcmp(receivedString, mystring) == 0) {
      digitalWrite(led, LOW);  // Not pressed
    } else if (strcmp(receivedString, mystring1) == 0) {
      digitalWrite(led, HIGH);  // Pressed
    }

    // Display the received string
    Serial.print("Received Button State: ");
    Serial.println(receivedString);

}
```

Listing 5: Receiver Arduino program

## 2.4. Analysis

In this setup, one Arduino acts as *the sender*, while the other serves as *the receiver*. The sender Arduino (connected to a pushbutton) continuously reads the state of the button. If the button is pressed, it sends a predefined string "p" over the UART connection; otherwise, it sends the string "np." On the receiver side, the Arduino continuously reads the incoming serial data. Upon receiving a string, it compares it with the predefined strings "p" and "np." If a match is found, the receiver Arduino toggles the state of an LED accordingly, indicating whether the button was pressed or not. This communication protocol enables the exchange of information between the two Arduinos, allowing them to synchronize actions based on the state of the pushbutton, as illustrated in Figure 4.

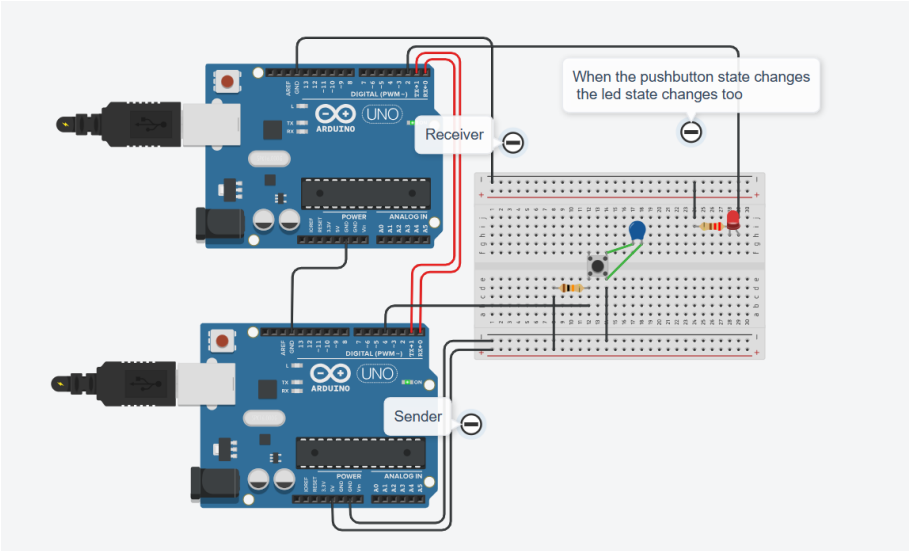



Figure 4: Arduino-Arduino communication

# Conclusion

The Raspberry Pi tasks, such as LED control based on push button states, highlighted the simplicity of GPIO programming. Introducing UART communication between Raspberry Pi and Arduino expanded our knowledge of seamless information exchange between platforms.

The Arduino-Arduino UART communication demonstrated the practical application of serial communication, showcasing bidirectional data transfer. This emphasized the importance of UART in enabling synchronized actions between connected devices.