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

LAB report n°05

Title:

Arduino-Raspberry Wired Communications UART

Studied by:

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

A. Theory

1. Definition UART and particularly USB

UART, or **Universal Asynchronous Receiver/Transmitter**, is a prevalent communication protocol employed for serial communication among microcontrollers, computers, and various devices. Unlike synchronous methods, UART facilitates asynchronous data transmission, eliminating the necessity for a clock signal to synchronize sending and receiving devices. Its adaptable speed ranges from a few hundred bits per second to several megabits per second.

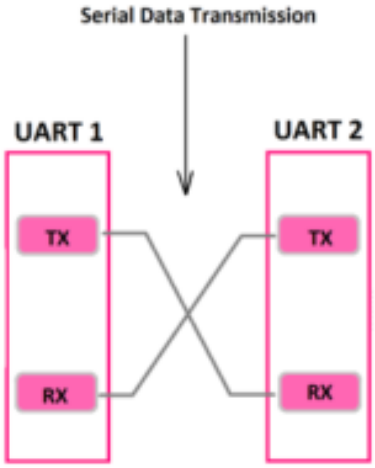


Figure 1: UART communication

On the other hand, USB, or **Universal Serial Bus**, serves as a widespread communication protocol for high-speed data transfer across devices like computers, smartphones, printers, and cameras. Noteworthy for its plug-and-play feature, USB enables the seamless connection and disconnection of devices without system shutdown.

With USB 3.2 Gen 2x2 being the fastest variant, achieving speeds up to 20 Gbps, USB finds utility in diverse applications, including:Data Transfer, Device Charging..

2. Introduction and Comparing Arduino vs Raspberry UART

2.1. Theoretical study of UART of an Arduino MKR1010 pins and software related to UART

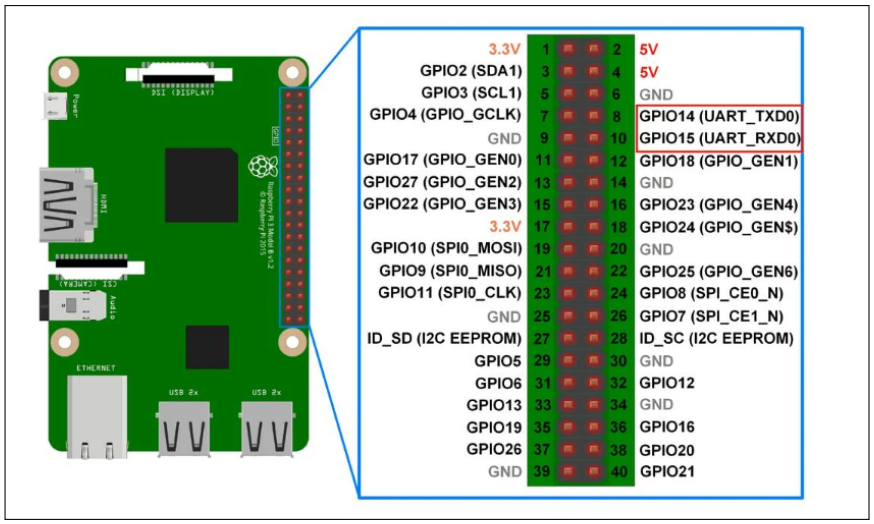
The **Arduino MKR1010** board provides two hardware UART interfaces: **Serial** (utilizing pins 0 for RX and 1 for TX) and **Serial1** (using pins 13 for RX and 14 for TX).

While **Serial** is commonly employed for programming and debugging, **Serial1** offers an additional UART channel for communication with external devices. In Arduino programming, built-in functions facilitate UART communication. The**Serial** object manages data transfer via the **Serial** interface, while the **Serial1** object is dedicated to communication through the **Serial1** interface. This abstraction simplifies the integration of UART functionality into projects, allowing developers to interact seamlessly with the MKR1010's hardware UART interfaces.

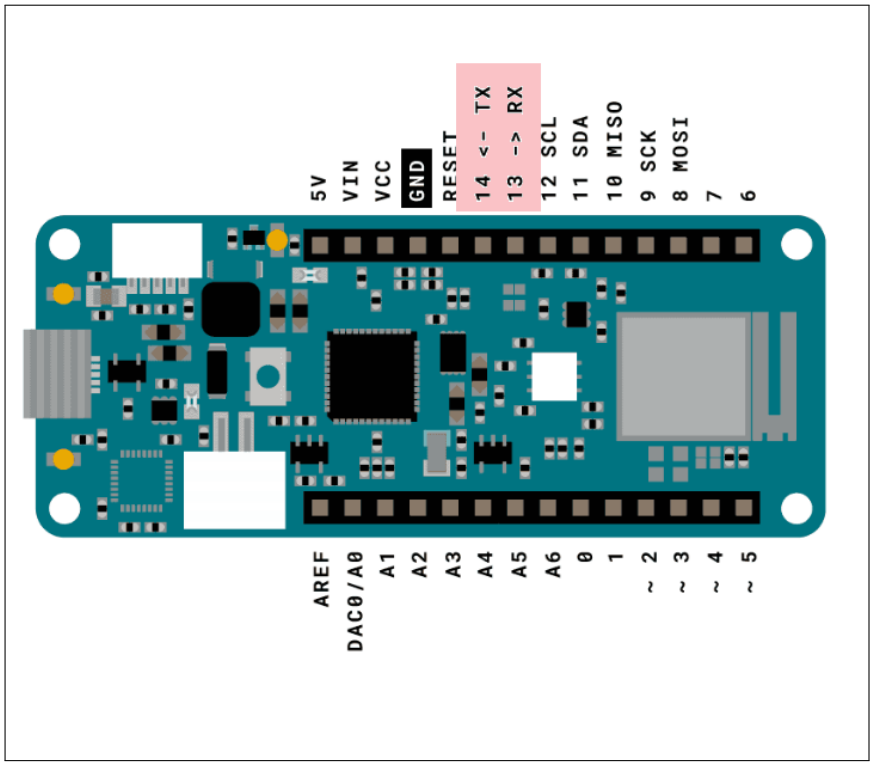
2.2. Theoretical study of UART of Raspberry pins and software related to UART

The Raspberry Pi 4 offers two hardware UART interfaces, namely **UART0** (utilizing pins 14 for TX and 15 for RX) and **UART1** (using pins 8 for TX and 10 for RX). **UART0** is linked to the built-in Bluetooth module, while **UART1** serves for additional UART communication.

For software, the Raspberry Pi supports UART through its built-in Linux terminal interface. The operating system features a default serial console using the **UART0** interface, adjustable via the Raspberry Pi configuration file. Beyond the Linux terminal, UART communication is facilitated through programming languages like Python. The ***pyserial*** library enhances Python's capability to send and receive data over the UART interface on the Raspberry Pi.



(a) Raspberry Pi 3 UART Pins



(b) UART pins in Arduino MKR 1010

Figure 2: LDR connection electronics views

B. Activity

1. Raspberry PUSH-BUTTON-LED

1.1. Blinked LED

Hardware

The setup involves a basic connection between an LED and a Raspberry.

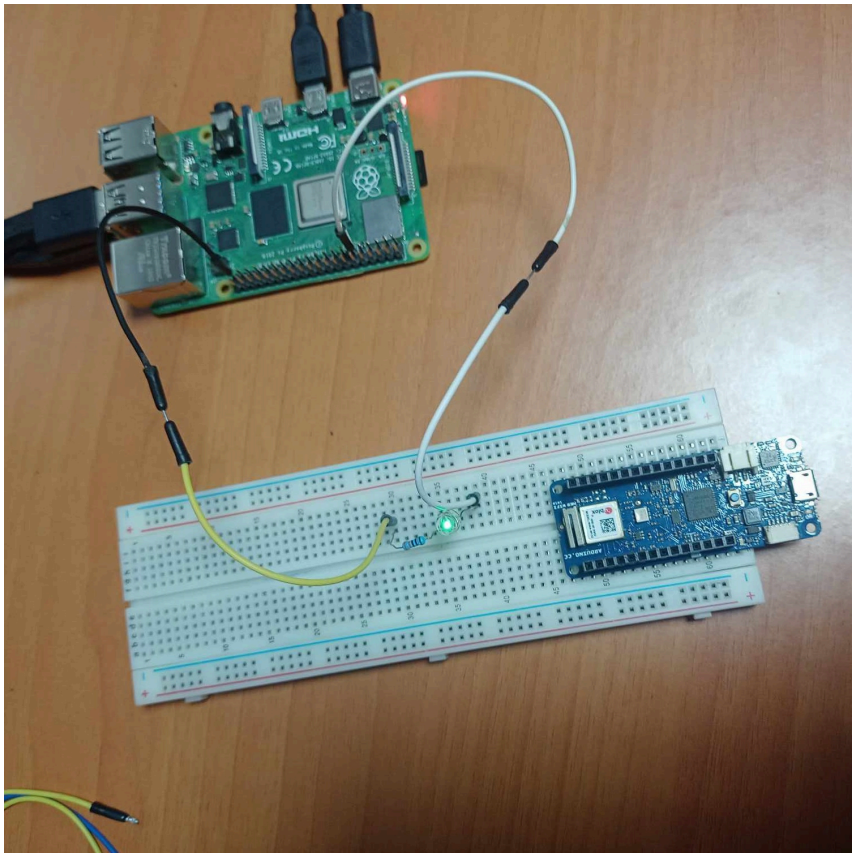


Figure 3: Blinked LED

Software

After reading the button state using `GPIO.input()` function, if the value is high, the LED is turned on using `GPIO.output()`.

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

```python
import time

LED_PIN = 23
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_PIN, GPIO.OUT)  # set the LED as an output

try:
    while True:
        GPIO.output(LED_PIN, GPIO.HIGH)  # turn on the LED
        time.sleep(1)
        GPIO.output(LED_PIN, GPIO.LOW)  # turn off the LED
        time.sleep(1)
except KeyboardInterrupt:
    GPIO.cleanup()
```

Listing 1: LDR connection Raspberry program

1.2. LED with Push Button

Hardware

The setup involves a basic connection between a push button, an LED, and a Raspberry.

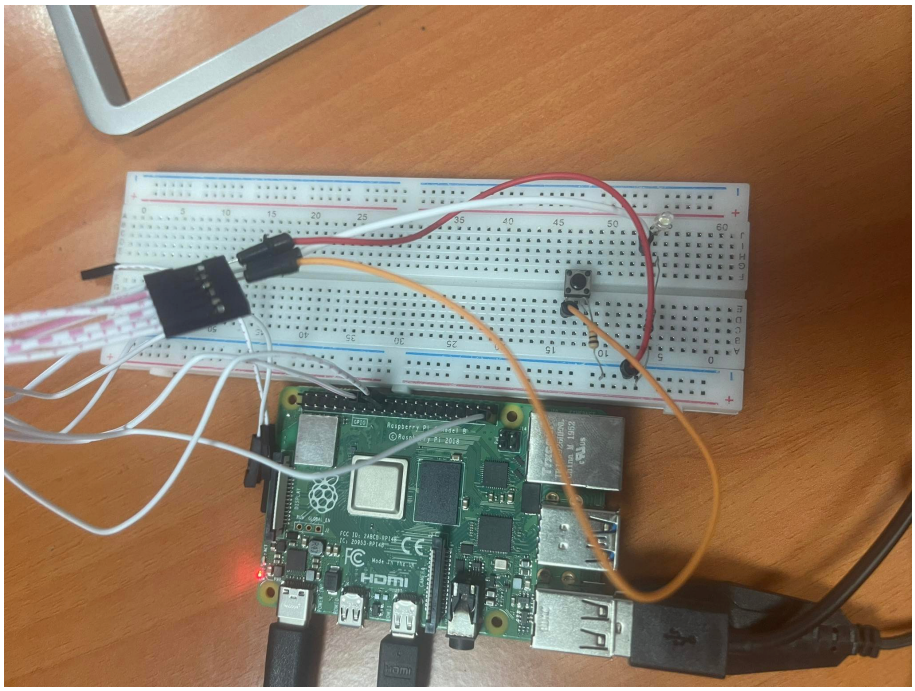


Figure 4: LED with Push Button

Software

After reading the button state using `GPIO.input()` function, if the value is high, the LED is turned on using `GPIO.output()`. This code controls an LED based on the state of a push button. If the push button is pressed, the LED turns on; otherwise, it turns off. The program runs continuously until a keyboard interruption (Ctrl+C) is detected, at which point GPIO cleanup is performed.

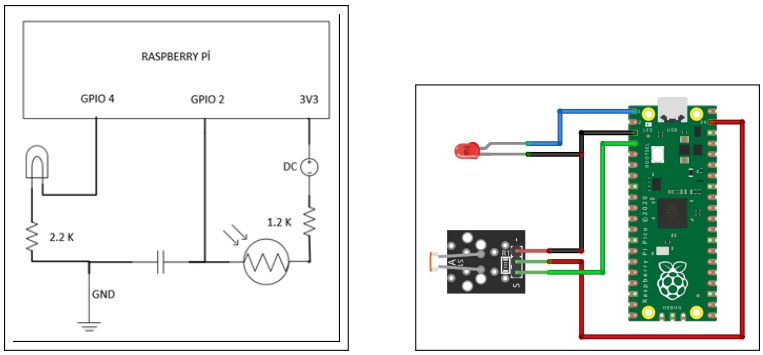
```python
# imports
import RPi.GPIO as GPIO
import time

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 2: LED with Push Button program

1.3. LED and LDR

Hardware

The setup involves a basic connection between a light sensor (LDR), an LED, and a Raspberry.



(a) Schematic view (b) Circuit view

Figure 5: LED and LDR connection electronics views

Software

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

# Set up GPIO
GPIO.setmode(GPIO.BCM)
LDR_PIN = 18
LED_PIN = 17

# Set up LDR pin as input and LED pin as output
GPIO.setup(LDR_PIN, GPIO.IN)
GPIO.setup(LED_PIN, GPIO.OUT)

try:
    while True:
        ldr_value = GPIO.input(LDR_PIN)
        if ldr_value == GPIO.LOW:  # LDR senses darkness
            GPIO.output(LED_PIN, GPIO.HIGH)
        else:
            GPIO.output(LED_PIN, GPIO.LOW)
        time.sleep(0.1)  # Adjust the sleep time as needed
except KeyboardInterrupt:
    GPIO.cleanup()
```

Listing 3: LED and LDR program

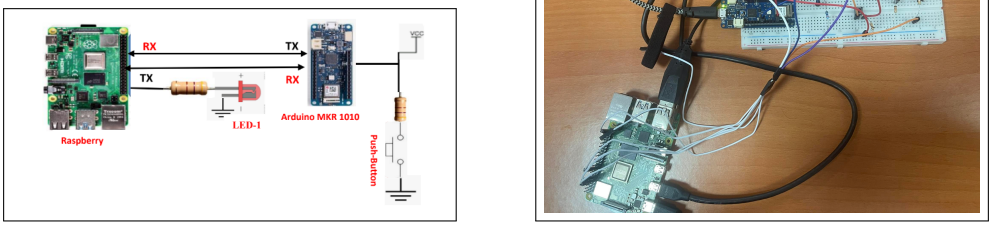
Analysis

The Raspberry Pi does not have built-in analog-to-digital converters (ADCs) on its GPIO pins, so you read the LDR as a digital signal (0 or 1) using a digital GPIO pin. The LDR, in conjunction with a resistor, forms a voltage divider. Depending on the amount of light falling on the LDR, the voltage at the junction of the LDR and the resistor changes. The GPIO pin reads either a HIGH (1) or LOW (0) value based on the voltage level.

2. Arduino-Raspberry-PUSH-BUTTON-LED

Hardware

The circuit consists of a Raspberry Pi and an Arduino connected via a UART serial bus. The LED connected to the Raspberry Pi is turned on or off each time the push button connected to the Arduino is pressed.



(a) Schematic view (b) Electronics mounting

Figure 6: Arduino-Raspberry-PUSH-BUTTON-LED

Software

Arduino

In this program, the button state is read by the Arduino board and sent to the Raspberry board through the UART connection.

```
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

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

Listing 4: Arduino-Raspberry-PUSH-BUTTON-LED program

Raspberry

The button state is received from the Arduino through the UART connection, and then the LED is turned off or on based on this state.

```python
import serial
import RPi.GPIO as GPIO
LED_PIN = 18
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_PIN, GPIO.OUT)

# initiate the serial connection
ser = serial.Serial('/dev/ttyS0', 9600)

try:
    while True:
        # reading the value from the serial
        buttonState = int(ser.readline().decode('ascii').strip())

        if buttonState == 1:
            GPIO.output(LED_PIN, GPIO.HIGH)  # turn on the LED
        else:
            GPIO.output(LED_PIN, GPIO.LOW)  # turn off the LED

except KeyboardInterrupt:
    GPIO.cleanup()
    ser.close()
```

Listing 5: Arduino-Raspberry-PUSH-BUTTON-LED program

3. Arduino-Raspberry-PUSH-BUTTON-LDR

Hardware

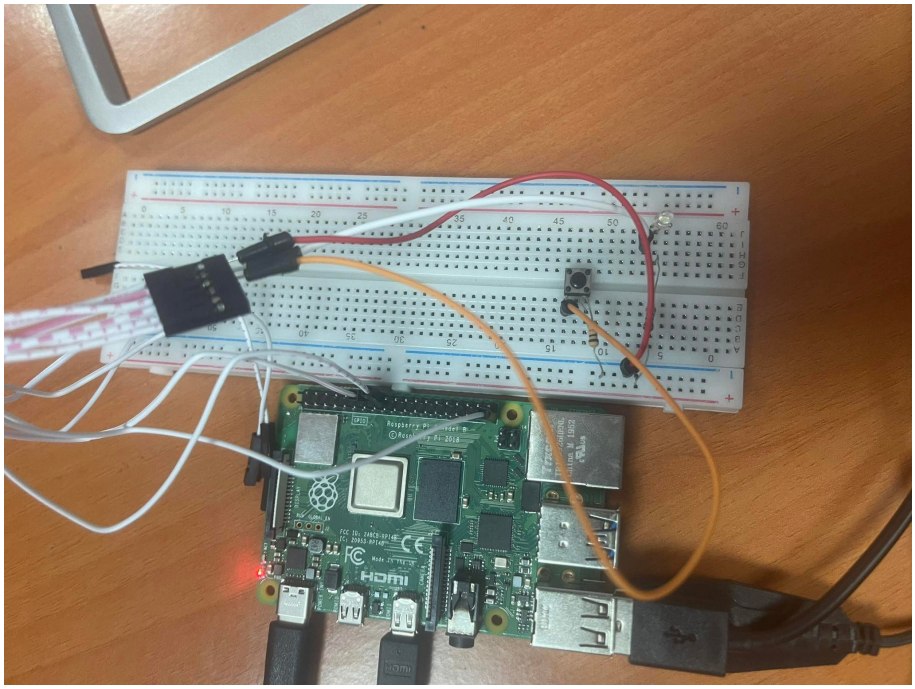


Figure 7: Enter Caption

3.1. Software

Arduino

```
int LDR_PIN = A0;
int ldrValue;

void setup() {
    Serial.begin(9600); // initiate the serial connection
}

void loop() {
    ldrValue = analogRead(LDR_PIN); // reading the light value
    Serial.println(ldrValue); // sending the value through the serial
    delay(100);
}
```

Listing 6: Arduino-Raspberry-PUSH-BUTTON-LDR program

Raspberry

```
import serial
import RPi.GPIO as GPIO
import time

```

```
LED_PIN = 18

GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_PIN, GPIO.OUT)

ser = serial.Serial('/dev/ttyS0', 9600)

# function to control led brightness
def set_led_brightness(brightness):
    duty_cycle = int((brightness / 1023) * 100)
    pwm = GPIO.PWM(LED_PIN, 100)
    pwm.start(duty_cycle)
    time.sleep(0.1)
    pwm.stop()

try:
    while True:
        # reading the light value from the serial connection
        ldrValue = int(ser.readline().decode('ascii').strip())

        # mapping the value
        ledValue = map(ldrValue, 0, 1023, 0, 255)

        set_led_brightness(ledValue)

except KeyboardInterrupt:
    GPIO.cleanup()
    ser.close()
```

Listing 7: Arduino-Raspberry-PUSH-BUTTON-LDR program

Conclusion

In conclusion, this lab provided valuable hands-on experience and insights into the practical applications and importance of UART (Universal Asynchronous Receiver/Transmitter) communication. The activities involved the communication between Arduino and Raspberry Pi using UART, demonstrating its significance in various scenarios.

The integration of sensors, such as LDRs (Light Dependent Resistors) and pushbuttons, with the Raspberry Pi showcased the versatility of UART communication. This capability was further highlighted in the Arduino-Raspberry-PUSH-BUTTON-LED activity, where the pushbutton press on the Arduino side triggered the LED state change on the Raspberry Pi. This interaction emphasizes the role of UART in enabling seamless communication between different microcontrollers for coordinated actions.

Additionally, the Arduino-Raspberry-PUSH-BUTTON-LDR activity expanded the scope by incorporating an LDR to control the LED based on ambient light levels. This practical implementation illustrated the flexibility and efficiency of UART in facilitating data exchange between devices, allowing for dynamic responses to environmental conditions.

The lessons drawn from this lab extend beyond the specific hardware connections and programming syntax. They encompass the broader understanding of UART as a reliable communication protocol for diverse applications in embedded systems. The ability to interface sensors, control LEDs, and exchange data between Arduino and Raspberry Pi reflects the real-world relevance of UART in IoT (Internet of Things) and embedded projects.