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 SIL G1

Lab Report n°04

Title:

Arduino Communications PWM-Sensors-Actuators-Raspberry (Intro)

Studied by:

First Name: Yasmine

Last Name: DINARI

E_mail: ly_dinari@esi.dz

A. Theory

1. PWM

1.1. Definition

Pulse Width Modulation (PWM) is a technique used to simulate analog signals using digital outputs by varying the width of pulses in a signal. The duty cycle, expressed as a percentage, determines the proportion of time the signal remains ”on” during each cycle. For example, a 50% duty cycle means the signal is ”on” half the time, resulting in an average output voltage that is half of the maximum voltage. PWM is used in applications like controlling LED brightness, motor speed regulation, and audio signal generation.

1.2. Comparing Arduino vs Raspberry GPIO, PWM, and Int. Pins

1.2.1. Raspberry Pi: Installation and Configuration

The Raspberry Pi is a small, affordable single-board computer designed for educational and IoT applications. It runs on Linux-based operating systems, such as Raspberry Pi OS, and features GPIO pins for hardware interfacing. To set it up, Raspberry Pi OS is installed on a microSD card using the Raspberry Pi Imager, then the device is powered on with connected peripherals. The setup wizard guides users through configuring Wi-Fi, locale, and user credentials.

1.2.2. Comparing Arduino Analog GPIO with A0 and PWM

Arduino analog GPIO pins (A0-A5) are equipped with an ADC (Analog-to-Digital Converter) that reads continuous voltage signals with 10-bit resolution (0-1023). These pins are ideal for sensors requiring precise analog input. On the other hand, PWM pins simulate analog output by toggling between HIGH and LOW states at varying duty cycles. While analog pins provide true voltage readings, PWM approximates analog behavior by averaging digital signals over time.

1.2.3. Theoretical study of Analog, PWM and interrupt of an Arduino pins

Arduino supports three key functionalities:

Analog Pins: Measure continuous voltage signals via ADC with 10-bit resolution.

PWM Pins: Generate simulated analog signals by varying duty cycles for applications like LED dimming or motor speed control.

Interrupt Pins: Respond to external events in real-time by executing predefined routines when triggered by rising/falling edges or level changes on specific pins.

1.2.4. Theoretical study of Analog, PWM and interrupt of a Raspberry pins

Raspberry Pi GPIO pins offer multiple functionality:

Analog Signals: Not natively supported; external ADCs are required for reading analog inputs.

PWM: Software-based PWM is available for all GPIO pins using libraries like pigpio, while hardware PWM is limited to specific pins (e.g., GPIO12, GPIO13).

Interrupts: All GPIO pins can act as interrupt sources, supporting edge or level-triggered events for responsive applications

1.2.5. Comparing Arduino vs Raspberry GPIO, PWM, and Int. Pins

Feature	**Arduino**	**Raspberry Pi**
Analog GPIO	Native support with ADC (10-bit resolution)	Requires external ADC
PWM	Hardware-based on specific pins	Limited hardware PWM; software-based possible
Interrupts	Supported on select pins	Available on all GPIO pins
Real-time Control	Excellent due to microcontroller design	Limited due to OS overhead

Table 1: Comparing Arduino and Raspberry Pi on GPIO, PWM, and Interrupts

B. ACTIVITY:

1. Arduino

1.1. PWM usage

Using Pulse Width Modulation (PWM), the brightness of an LED can be controlled efficiently. PWM allows us to simulate analog output by varying the duty cycle of a digital signal. The Arduino functions analogWrite(), analogWriteResolution(), and map() are key tools for implementing this functionality.

PWM controls the average voltage delivered to a device by rapidly switching between HIGH and LOW states.

The duty cycle determines the brightness of the LED: higher duty cycles correspond to brighter LEDs.

analogWrite():

This function generates a PWM signal on specified pins. It accepts values between 0 (always off) and 255 (always on) by default.

Syntax:analogWrite(pin, value).

analogWriteResolution():

By default, Arduino uses 8-bit resolution (0-255). This can be increased using analogWriteResolution to allow finer control over brightness (e.g., 12-bit resolution provides a range of 0-4095).

map():

The map() function scales values from one range to another. For example, it can convert an 8-bit range (0-255) to a 12-bit range (0-4095).

Syntax: map(value, fromLow, fromHigh, toLow, toHigh).

Steps to Control LED Brightness

- Connect the cathode (-) of the LED to GND.
- Connect the anode (+) of the LED to a PWM-capable pin on the Arduino via a resistor.
- Use pinMode() to set the LED pin as output.
- Set the PWM resolution using analogWriteResolution(bits).

- Gradually increase and decrease brightness using loops and map values between ranges.

```
const int ledPin = 0;  // PWM pin
int brightness = 0;    // brightness

void setup() {
  pinMode(ledPin, OUTPUT);  // Set LED pin as output
  analogWriteResolution(12);  // Set PWM resolution to 12-bit (0-4095)
}

void loop() {
  // Increase brightness
  for (int i = 0; i <= 255; i++) {
    brightness = map(i, 0, 255, 0, 4095);  // Convert 8-bit to 12-bit
    analogWrite(ledPin, brightness);  // Set brightness
    delay(20);  // transition
  }

  // Decrease brightness
  for (int i = 255; i >= 0; i--) {
    brightness = map(i, 0, 255, 0, 4095);  // Convert 8-bit to 12-bit
    analogWrite(ledPin, brightness);  // Set brightness
    delay(20);
  }
}
```

Listing 1: LED Dimming Using PWM

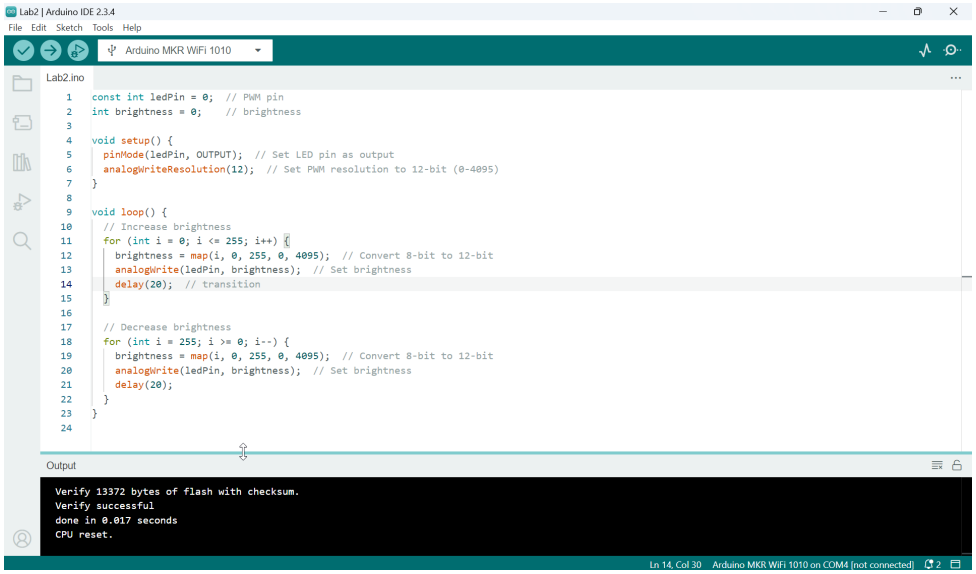

Figure 1: Simulation of the PWM usage Circuit in Arduino IDE

PWM Signal Generation: The analogWrite() function generates a PWM signal with a duty cycle proportional to the input value.

Increasing or decreasing this value adjusts the LED's perceived brightness.

Resolution Adjustment: Using analogWriteResolution(12) increases control granularity by allowing values up to 4095 instead of the default maximum of 255.

Value Mapping: The map() function converts values from one range (e.g., sensor input or default resolution) to another range suitable for higher-resolution PWM signals.

Smooth Transitions: Loops with incremental changes and delays ensure gradual adjustments in brightness for visual appeal.

1.2. PWM-LDR-LED Usage

Using a Light Dependent Resistor (LDR) in combination with Pulse Width Modulation (PWM), the brightness of an LED can be dynamically controlled based on ambient light levels. The LDR is a sensor whose resistance decreases as the light intensity increases, producing an analog voltage that can be read using the ‘analogRead()‘ function. This value is then mapped to a suitable range using the ‘map()‘ function and used as input to the ‘analogWrite()‘ function to control the LED’s brightness.

The process involves reading the LDR’s analog output, mapping its value (0-1023) to a range compatible with PWM (e.g., 0-255), and writing the resulting value to a PWM-capable pin connected to the LED. This creates a circuit where the LED’s brightness adjusts automatically based on the level of light detected.

analogRead(): This function reads an analog voltage from a specified pin and returns a value between 0 and 1023 (10-bit resolution). It is used here to measure the light level detected by the LDR.

map(): The ‘map()‘ function scales values from one range to another. In this case, it converts the LDR’s 10-bit output (0-1023) to an 8-bit range (0-255) suitable for PWM.

analogWrite(): This function generates a PWM signal on a specified pin with a duty cycle proportional to the input value. It is used here to control the brightness of the LED.

Steps to Control LED Brightness Using LDR

- Connect the LDR in a voltage divider configuration with one end connected to 5V, the other end connected to an analog pin (e.g., A1), and a resistor connected between the analog pin and GND.
- Connect the cathode (-) of the LED to GND.
- Connect the anode (+) of the LED to a PWM-capable digital pin (e.g., pin 0) via a current-limiting resistor.
- Use ‘analogRead()‘ to read the LDR’s output, map its value using ‘map()‘, and write it to the LED using ‘analogWrite()‘.

```
const int ledPin = 0;       // PWM pin for LED
const int ldrPin = A1;      // Analog pin for LDR
```

```
3
4  void setup() {
5    pinMode(ledPin, OUTPUT);  // Set LED pin as output
6  }
7
8  void loop() {
9    int lightLevel = analogRead(ldrPin);  // Read light level from LDR
       (0-1023)
10
11   // Map light level to LED brightness (8-bit range)
12   int brightness = map(lightLevel, 0, 1023, 0, 255);
13
14   // Set LED brightness using PWM
15   analogWrite(ledPin, brightness);
16
17   delay(5); // Small delay for stability
18 }
```

Listing 2: Controlling LED Brightness Using an LDR

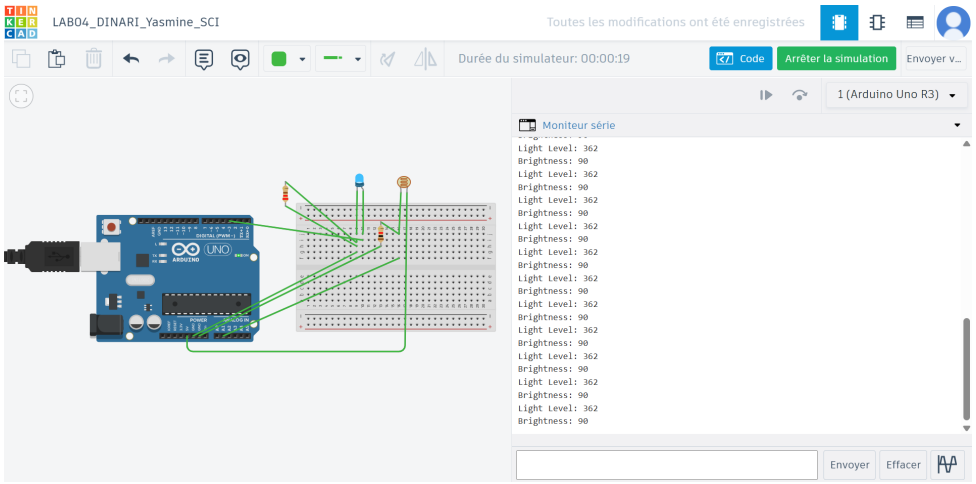

Figure 2: Simulation of the PWM-LDR-LED usage Circuit in Tinkercad

Comparison: PWM Pin vs Analog GPIO PWM pins and analog GPIO pins serve different purposes:

Feature	**PWM Pin**	**Analog GPIO**
Functionality	Generates digital signals with varying duty cycles	Reads continuous voltage signals
Output Type	Simulated analog output	True analog input
Applications	Controlling brightness or speed	Reading sensor values
Resolution	Typically 8-bit (0-255)	Typically 10-bit or higher (e.g., 0-1023)

Table 2: Comparison of PWM Pin and Analog GPIO

1.3. Applications of PWM

Beyond the common applications like LED dimming and motor speed control, PWM can be employed in innovative ways:

- **Smart Home Systems:** PWM can regulate the brightness of smart lighting or control the speed of motors in devices like automated curtains and smart clothes hangers.
- **Temperature Control:** In heating systems, PWM adjusts the power delivered to heating elements for precise temperature regulation.
- **Audio Processing:** Class-D audio amplifiers use PWM for efficient sound reproduction with minimal heat generation.
- **Automotive Applications:** PWM controls engine systems, interior lighting, and power windows, offering smooth operation and energy efficiency.

These examples highlight the adaptability of PWM across industries, making it indispensable in modern electronics.

2. CONCLUSION

In this lab, we explored Pulse Width Modulation (PWM) using Arduino. Through hands-on activities, we demonstrated how PWM controls LED brightness and adjusts it based on ambient light using an LDR. The use of functions like ‘analogWrite()‘, ‘map()‘, and ‘analogRead()‘ highlighted Arduino’s efficiency in signal control.

We also examined the differences between Arduino and Raspberry Pi regarding GPIO, PWM, and interrupts, emphasizing Arduino’s responsiveness in real-time applications and Raspberry Pi’s processing capabilities for more complex tasks.

This lab reinforced our understanding of PWM, further illustrating its relevance in various control systems.