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°04

Title:

Interrupts-PWM-Sensors-Actuators

(Introduction to raspberry)

Studied by:

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

A. Theory

1. PWM

1.1. Definition

PWM, or Pulse Width Modulation, is a technique for controlling electrical power by rapidly turning it on and off at a fixed rate. The duty cycle, the ratio of time the power is on to off, is adjusted to regulate the power delivered to the load. For instance, a 50% duty cycle means the power is on half the time, delivering half the maximum power. The accuracy of PWM refers to how closely the actual duty cycle matches the desired one, measured as a percentage. Factors affecting accuracy include signal frequency, generator resolution, component quality, and system noise. Higher frequency and resolution typically result in better accuracy.

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

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

Arduino boards are equipped with a specialized Analog-to-Digital Converter (ADC) designed to precisely read analog voltages from dedicated input pins. The ADC, with a 10-bit resolution, ensures high precision by breaking the 0 to 5-volt range into 1024 discrete steps. Additionally, Arduino boards feature multiple pins capable of generating Pulse Width Modulation (PWM) signals, crucial for controlling power delivery to devices like motors or LEDs. These PWM pins are easily identifiable by the tilde ( ) symbol. Moreover, Arduino provides various pins designated for interrupt functionality, enabling the detection of signal level changes and triggering interrupt signals to the microcontroller. The Arduino Integrated Development Environment (IDE) further simplifies interrupt management with a dedicated library, facilitating the seamless incorporation of interrupt-driven features into Arduino projects.

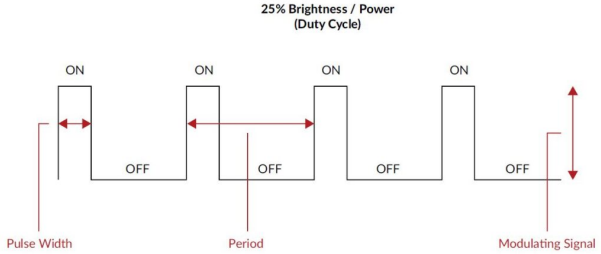


Figure 1: PWM generated signal

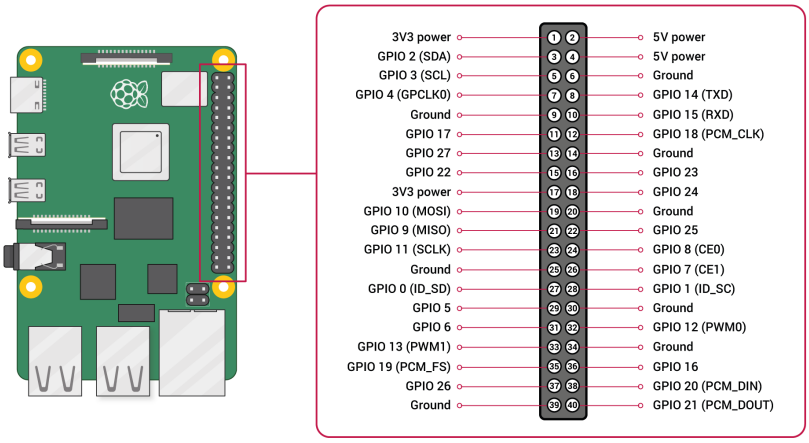


Figure 2: GPIO Pinout Diagram

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

Raspberry Pi boards lack dedicated analog input pins, but users can employ General Purpose Input/Output (GPIO) pins along with an external Analog-to-Digital Converter (ADC) to read analog voltages. Additionally, Raspberry Pi GPIO pins have the capability to generate Pulse Width Modulation (PWM) signals. Unlike Arduino, Raspberry Pi doesn't incorporate specific PWM pins; instead, PWM functionality is implemented in software using Pulse-Width Modulation (PWM) modules. It's important to note that although GPIO pins can function as interrupt pins, Raspberry Pi does not provide a dedicated library for efficient interrupt management. This distinction underscores the need for a more manual approach when incorporating interrupt-driven functionalities in Raspberry Pi projects.

2. Raspberry

2.1. Introducing Raspberry and its Pins

The Raspberry Pi is a compact and versatile single-board computer designed for diverse applications, from DIY projects to education. Despite its small size, it delivers robust computing capabilities. The board includes GPIO (General-Purpose Input/Output) pins, functioning as a physical interface for external connections. These pins act as switches, controlled by external input or the Pi itself, facilitating interaction with electronic circuits for tasks like LED and motor control. Additionally, the GPIO pins enable the Raspberry Pi to detect changes in the environment, such as button presses, temperature, and light levels, expanding its utility in physical computing. The Raspberry Pi has a total of 40 pins (26 on early models), each serving different functions.

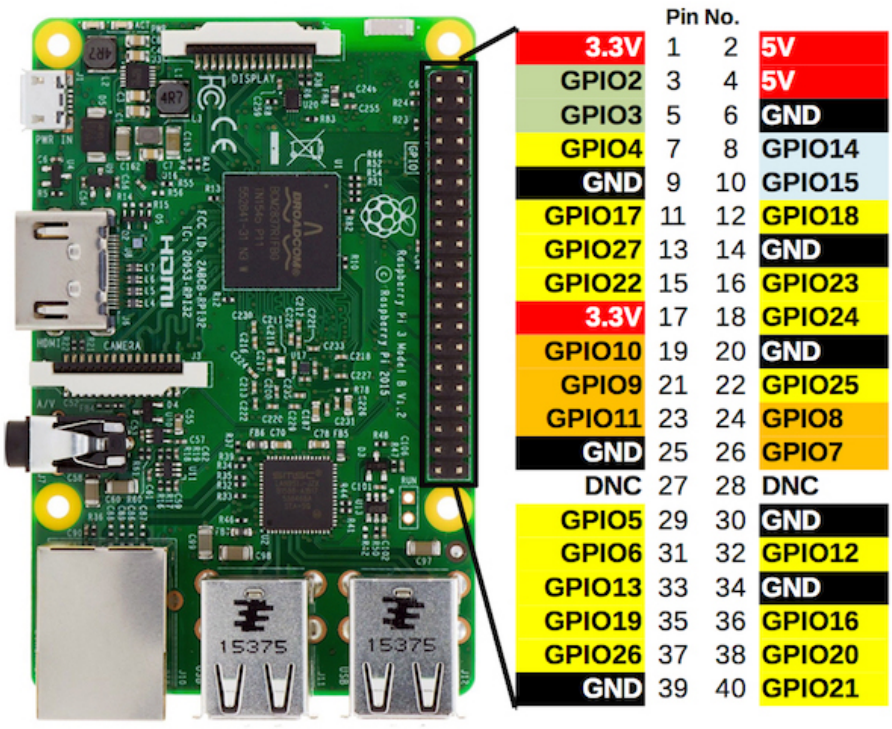


Figure 3: Enter Caption

2.2. Comparison with Arduino Mkr1010

Comparison Parameter	Arduino MKR1010	Raspberry Pi
Control unit	ATmega family	ARM family
Foundation	Microcontroller	Microprocessor
Usage	Controlling electrical components	Computing and managing electrical components
Hardware and software structure	Simple	Complex
CPU architecture	8-bit	64-bit
RAM	About 2 kB	About 1 GB
Clock speed	From 16 MHz	From 1 GHz
Cost-efficiency	Higher than Raspberry Pi	Lower than Arduino
I/O voltage	5V	Between 1.8V and 3.3V
Power consumption	About 200 MW	About 700 MW

Table 1: Comparison of Arduino MKR1010 and Raspberry Pi

2.3. Necessary steps to install Raspberry

Components

- Raspberry Pi board

- MicroSD card (8GB or larger)
- HDMI cable, keyboard, and mouse
- Internet connection (Wi-Fi or Ethernet)

Steps

1. **Download the OS:**
   - Visit the Raspberry Pi website: `https://www.raspberrypi.com/`
   - Download the latest OS.
2. **Get Etcher:**
   - Download Etcher, a tool for writing OS images to SD cards.
3. **Write OS to MicroSD:**
   - Use Etcher to write the OS onto the MicroSD card.
4. **Prepare MicroSD:**
   - Insert the MicroSD card into your computer.
   - If using Wi-Fi, create a `wpa_supplicant.conf` file.
   - Add your Wi-Fi details to the file.
5. **Setup Raspberry Pi:**
   - Insert the MicroSD card into the Raspberry Pi.
   - Connect the Pi to the monitor, keyboard, mouse, and power it up.
   - Follow the prompts on the screen.
   - Configure your preferences, including language and password.
   - Let the Pi boot up and greet you with its desktop.
6. **Update and Upgrade:**
   - Open a terminal and run the following commands:

```
sudo apt update
sudo apt upgrade
```

Your Raspberry Pi is now ready for exploration and experimentation.

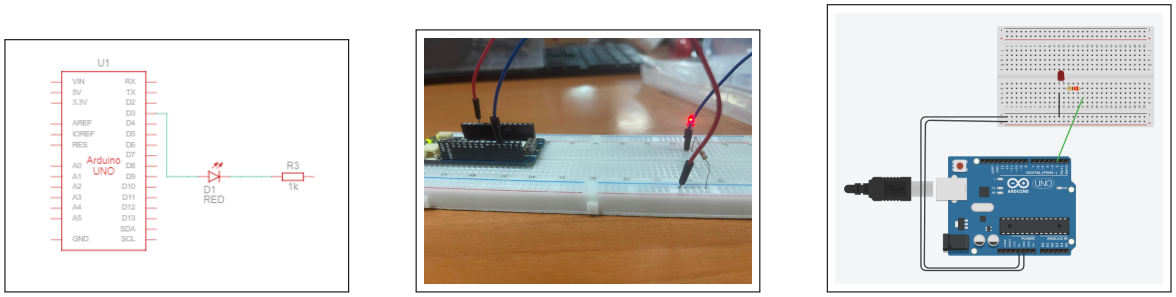
B. Activity

1. ARDUINO

1.1. PWM usage

Hardware

The setup involves a basic connection between an LED and an Arduino board.



(a) Schematic view (b) Electronics mounting (c) Circuit view

Figure 4: LDR connection electronics views

Software

This program turn on a led with different degree of brightness using the analogWrite() function.

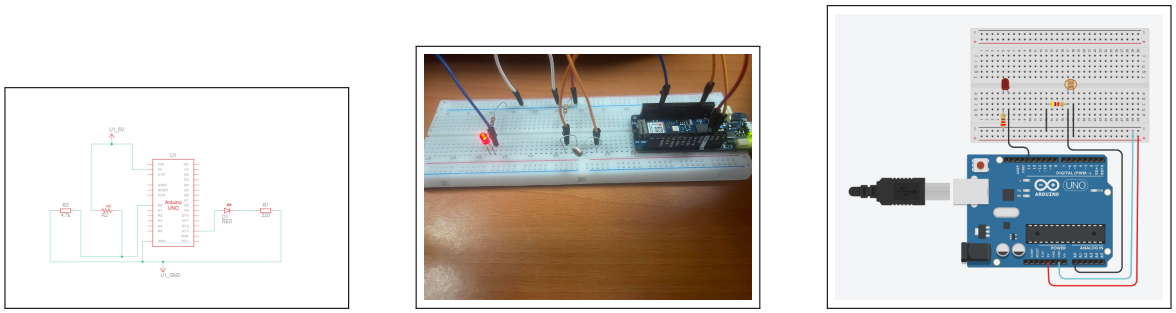
```
int ledPin = 2;
void setup() {
  pinMode(ledPin, OUTPUT);
  // Set the LED pin as an output
}
void loop() {
  for (int i = 0; i < 255; i += 20) {
    analogWrite(ledPin, i);
    // Write the analog value to the LED pin
    delay(80);
  }
  delay(1000);}
```

Listing 1: LDR connection Arduino program

2. PWM-LDR-LED usage

2.1. PWM-Analog GPIO comparaison

Hardware



(a) Schematic view (b) Electronics mounting (c) Circuit view

Figure 5: LDR connection electronics views

Software

```
int ledPin = 2;
int sensorPin = A0;
int light = 0;

void setup() {
  pinMode(ledPin, OUTPUT); // set the LED as an output
  pinMode(sensorPin, INPUT); // set the sensor pin as an input
  Serial.begin(9600);
}

void loop() {
  light = analogRead(sensorPin); // read the light value

  Serial.print("Light value: ");
  Serial.println(light);

  int mappedValue = map(light, 0, 1023, 0, 255); // map the values

  Serial.print("Mapped value: ");
  Serial.println(mappedValue); // print the mapped value

  analogWrite(ledPin, mappedValue);
  delay(200);
}
```

Listing 2: PWM-Analog GPIO comparaison

Analysis

In terms of resolution and accuracy, PWM pins offer higher precision compared to analog GPIO pins, as they can generate extremely accurate duty cycles. It's important

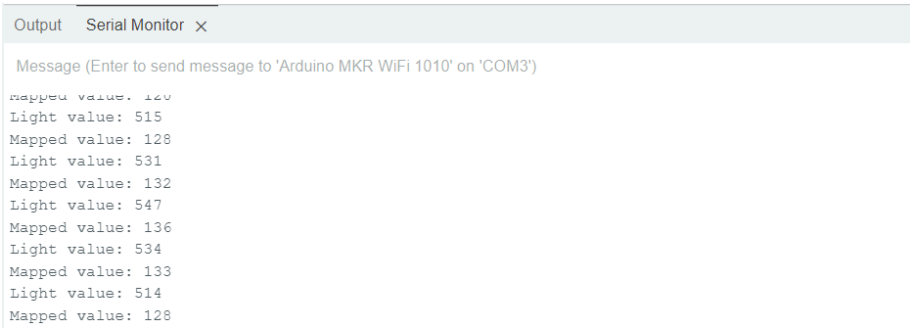
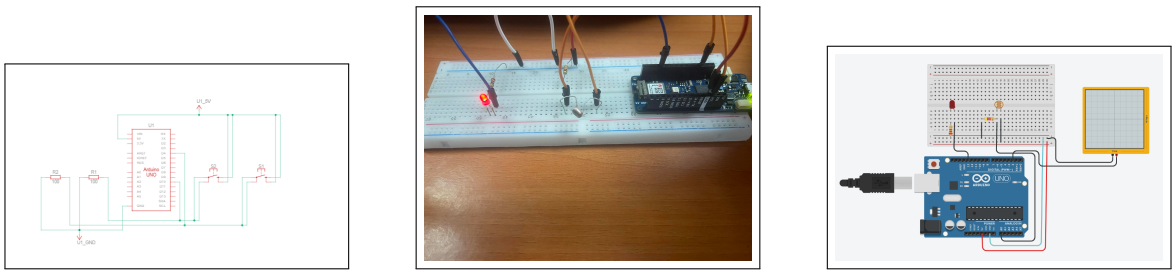


Figure 6: Console display

to note that PWM signals are inherently digital and may not be the optimal choice for applications requiring a continuous analog signal. On the contrary, analog GPIO pins can deliver a smooth and uninterrupted range of values between 0 and 5 volts, enabling precise analog control over connected components. However, it's essential to consider that the accuracy and resolution of analog GPIO pins may be susceptible to the influence of noise and interference.

2.2. Duty cycle

Hardware



(a) Schematic view (b) Electronics mounting (c) Circuit view

Figure 7: PWM duty cycle

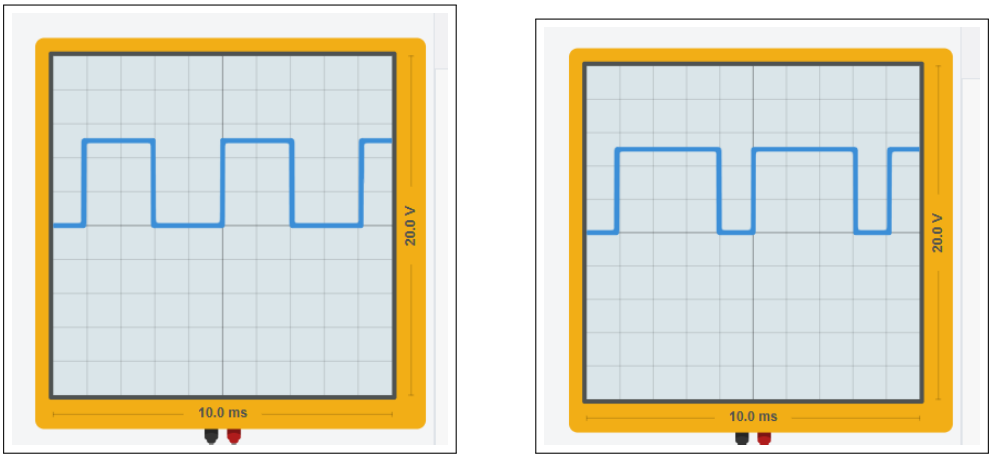
Software

```
1
2  const int sensorPin = A0;
3  const int pwmPin = 2; // Output pin for PWM signal
4  const int period = 10; // Period in milliseconds
5  const float dutyCyclePercentage = 40.0; // Desired duty cycle percentage
6
7  void setup() {
8    pinMode(sensorPin, INPUT);
9    pinMode(pwmPin, OUTPUT);
10   Serial.begin(9600);
11 }
12 void loop() {
13   // Read sensor value (0-255)
14   int sensorValue = analogRead(sensorPin);
15   // Convert sensor value to voltage (assuming 5V reference)
```

```
16    float voltage = ((sensorValue) / 255 ) * 5;
17    float onTime = (dutyCyclePercentage / 100 ) * period;
18    // Display output voltage on the oscilloscope
19    analogWrite(pwmPin, 255); // Set the PWM signal to HIGH
20    delayMicroseconds(onTime * 1000); // Delay for the on time
21    analogWrite(pwmPin, 0); // Set the PWM signal to LOW
22    delayMicroseconds((period - onTime) * 1000); // Delay for the remaining
        off time
23    // Print sensor value and voltage to Serial Monitor for verification
24    Serial.print("Sensor Value: ");
25    Serial.print(sensorValue);
26    Serial.print("\t Voltage: ");
27    Serial.println(voltage);
28
29    delay(1000); // Delay for readability, adjust as needed
30  }
```

Listing 3: pwm duty-cycle

Analysis



(a) duty cycle : 50% (b) duty cycle : 75%

Figure 8: PWM duty cycle

If you were to increase the duty cycle, the output voltage would increase as well. Similarly, if you were to decrease the duty cycle, the output voltage would decrease. You'll see a more hands-on approach a bit later in a short example we prepared.

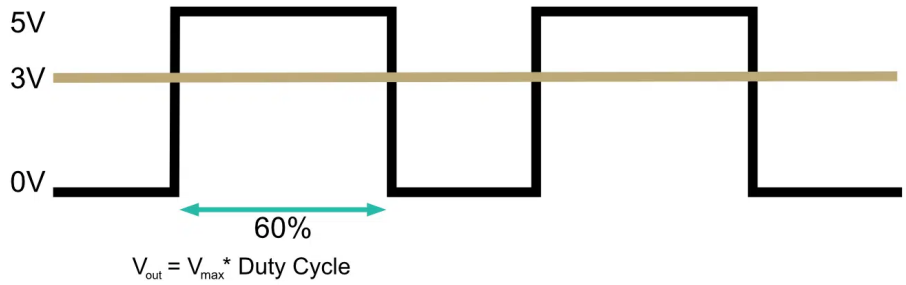
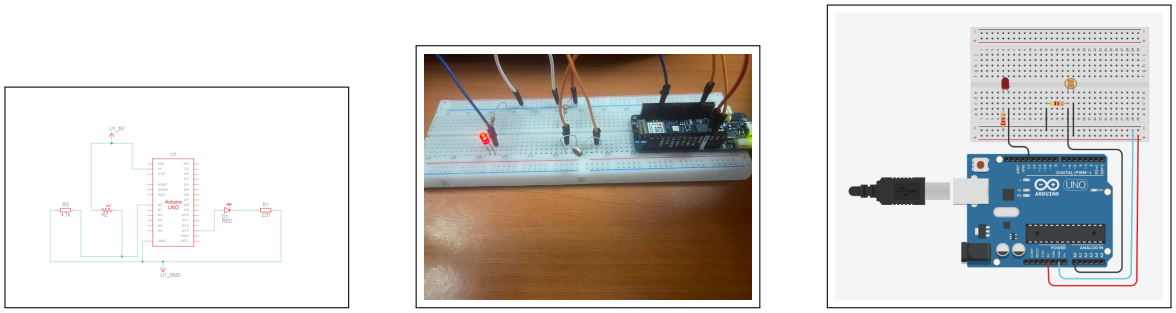


Figure 9: pwm duty-cycle calculated

2.3. Modulate the intensity of an LED using an LDR

Hardware



(a) Schematic view (b) Electronics mounting (c) Circuit view

Figure 10: LDR connection electronics views

Software

```
int ledPin = 2;
int sensorPin = A1;
int light = 0;

void setup() {
  pinMode(ledPin, OUTPUT); // Set the LED as an output
  pinMode(sensorPin, INPUT); // Set the sensor pin as an input
  Serial.begin(9600);
}

void loop() {
  light = analogRead(sensorPin); // Read the light value
  Serial.print("Light value: ");
  Serial.println(light);

  int mappedValue = map(light, 0, 1023, 0, 255); // Map the values

  Serial.print("Mapped value: ");
  Serial.println(mappedValue);

  analogWrite(ledPin, mappedValue); // Set LED brightness based on mapped
    value
  delay(200);
```

```
23 }
```

Listing 4: Modulate the intensity of an LED using an LDR

Analysis

The console displays the value read from the LDR and the mapped value.

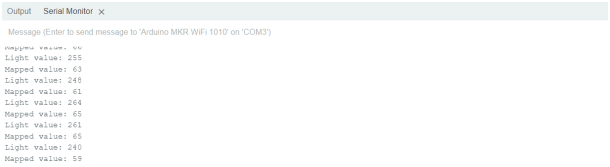


Figure 11: Console Display

3. Applications of PWM

3.1. Motor Control

PWM signals play a crucial role in the precise control of the speed and direction of DC motors and servo motors. By adjusting the duty cycle of the PWM signal, the average voltage applied to the motor can be finely tuned, thereby regulating the motor's speed.

3.2. Audio Amplification

PWM signals excel in the generation of high-fidelity audio signals. The modulation of the signal's pulse width allows for the extraction of the audio signal, achieved through the implementation of a low-pass filter.

3.3. Power Management

In the realm of power management, PWM signals find utility in controlling the power supplied to diverse electronic components. Through dynamic variations in the duty cycle of the PWM signal, there emerges precise control over the average power delivered to the component.

3.4. Communication

PWM signals can be used for communication between two devices. By modulating the pulse width of the signal, digital information can be encoded and transmitted.

C. Conclusion

Pulse Width Modulation (PWM) is a pivotal technique for analog signal generation using digital signals. PWM pins enable the output of analog-like values, utilizing converters like ADC (Analog-to-Digital Converter) and DAC (Digital-to-Analog Converter). The use of PWM facilitates precise control of components, offering a dynamic range of output values. Drawing a parallel with Raspberry Pi, a more advanced board, we recognized its extended capabilities. Notably, Raspberry Pi lacks integrated analog input, necessitating an external ADC for reading from sensors like LDR.