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°03

Title:

Arduino Communications Interrupts-PWM-Sensors-Actuators

Studied by:

First Name: Yasmine

Last Name: DINARI

E_mail: ly_dinari@esi.dz

A. Theory

1. Force Sensor

1.1. Definition

A force sensor is a device designed to detect and measure physical pressure or force. It operates by converting the applied force into an electrical signal, which can be processed and analyzed for various applications such as presence detection, object detection, and pressure measurement.

Figure 1: Force Sensor

1.2. Principle

The principle of a force sensor is based on the variation of resistance. When pressure is applied to the sensor, its resistance decreases proportionally to the force exerted. This change in resistance can be measured using electronic circuits, allowing the quantification of the applied force.

1.3. Category

The main types of force sensors are:

- **Piezoelectric Sensors:** Utilize the piezoelectric effect to measure dynamic forces like vibrations.
- **Resistive Sensors (FSR):** Measure force through resistance changes; ideal for presence detection.
- **Strain Gauge Sensors:** Detect deformation in materials to calculate force.
- **Capacitive Sensors:** Measure capacitance changes caused by displacement or deformation.
- **Hydraulic and Pneumatic Sensors:** Use fluid pressure to measure heavy loads.

1.4. Connection with Microcontrollers

Force sensors like the FSR02 (Force Sensing Resistor) can be easily interfaced with microcontrollers such as Arduino. The sensor is typically connected to an analog input pin to measure resistance changes corresponding to applied force. Additionally, digital output pins can be used to control actuators like LEDs based on threshold values derived from sensor readings.

2. Interruption

2.1. Definition

An interrupt is a signal that breaks the normal execution of a program to handle a high-priority event. When an interrupt occurs, the microcontroller pauses its current operations, saves its state, executes a specific function (known as an Interrupt Service Routine or ISR), and then returns to its previous state to continue execution.

2.2. Theoretical study of Arduino pins that allow interruptions

The Arduino MKR WiFi 1010 offers 10 external interrupt pins (0, 1, 4, 5, 6, 7, 8, 9, 16/A1, 17/A2).These interrupt pins enable more flexible circuit design, support for multiple simultaneous interrupt sources so that makes the MKR WiFi 1010 ideal for IoT applications requiring response to multiple external events.

2.3. Arduino Internal Pull-up

The internal pull-up resistors are integrated into the Arduino microcontroller. On the Arduino MKR WiFi 1010, they ensure a defined state (HIGH at 3.3V) for an input pin when no external signal is applied.

To enable the internal pull-up resistor in Arduino programming:

```
1 pinMode(pin, INPUT_PULLUP);
```

2.4. Structure of program-based interrupts

Interrupts allow the system to respond instantly to external events without missing them. The function:
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode) sets up an interrupt on a specific pin, where ‘pin‘ is the pin number, ‘ISR‘ is the function executed when the interrupt occurs, and ‘mode‘ determines the triggering condition (LOW, CHANGE, RISING, FALLING). To disable an interrupt, use **detachInterrupt(digitalPinToInterrupt(pin))**.

This function is useful when an interrupt is no longer needed or to prevent temporary interference.

```
const int ledPin = 6;  // LED pin
const int interruptPin = 0;  // Interrupt pin
volatile int state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), LED, CHANGE);
}

void loop() {
  digitalWrite(ledPin, state);
  delay(1000);  // Simulate other tasks
}

void LED() {
  state = !state;
}
```

Listing 1: Interruption Example

B. ACTIVITY:

1. Force Sensor

A force sensor and an LED are connected to the Arduino's GPIOs. The LED illuminates when the applied force exceeds a predefined threshold and turns off otherwise. Additionally, the force values are displayed on the console for monitoring.

Steps:

- Connect one leg of the force sensor to the $5V$ pin (VCC) on the Arduino.
- Connect the second leg of the force sensor to one end of a $10k\Omega$ resistor and to Analog Pin $A1$ on the Arduino.
- Connect the other end of the $10k\Omega$ resistor to ground (GND).
- Connect the cathode (short leg) of the LED to ground (GND).
- Connect a 220Ω resistor in series with the anode (long leg) of the LED, and connect the other end of the resistor to Digital Pin 8 on the Arduino.

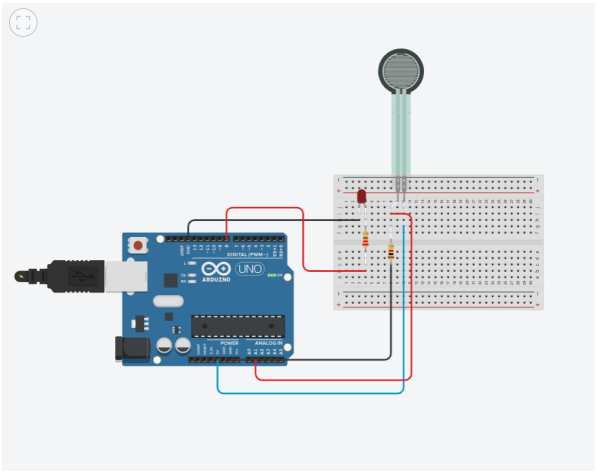

Figure 2: Simulation of the Force Sensor Circuit in Tinkercad

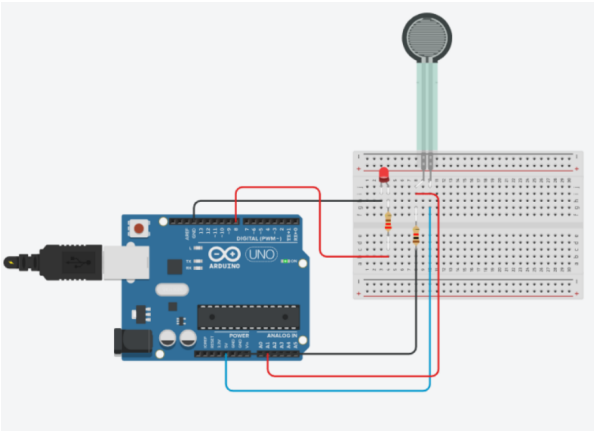

Figure 3: LED turns on when the applied force exceeds a certain threshold

```
void setup()
{
    pinMode(A1, INPUT);   // Force sensor
    pinMode(8, OUTPUT);   // LED output
    Serial.begin(9600);   // Start serial communication
}

void loop()
{
```

```
    int forceSensor = analogRead(A1); // Read sensor value

    if (forceSensor > 100) {
        digitalWrite(8, HIGH);   // Turn LED on
    } else {
        digitalWrite(8, LOW);    // Turn LED off
    }

    Serial.print("force = ");
    Serial.println(forceSensor);
    delay(100);
}
```

Listing 2: Simulation of Force Sensor Circuit Code

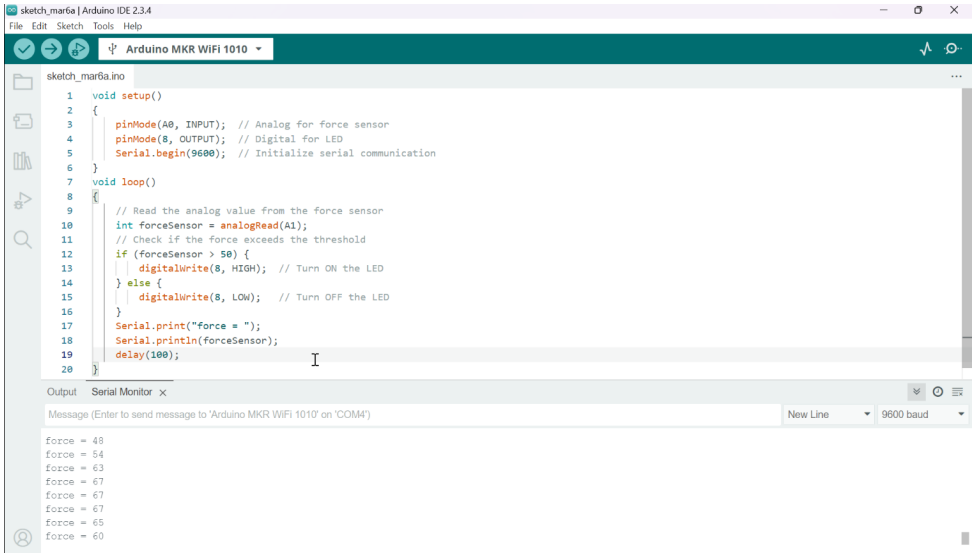

Figure 4: Simulation of the Force Sensor Circuit in Arduino IDE

2. Pushbutton with Internal Pull-Up

A pushbutton and an LED are connected to the Arduino using its built-in pull-up resistor. The LED switches on or off each time the button is released after being pressed. This setup removes the need for an external pull-up resistor.

Steps:

- Connect one side of the pushbutton to digital pin 2 on the Arduino.
- Connect the other side of the pushbutton to ground (GND).
- Connect the shorter leg (cathode) of the LED to GND.
- Connect a 220Ω resistor between the longer leg (anode) of the LED and digital pin 7 on the Arduino.

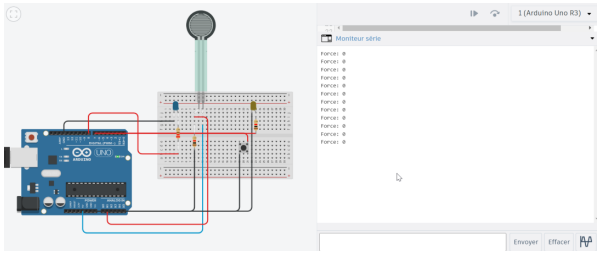

Figure 5: Pushbutton with Internal Pull-Up Circuit on Tinkercad

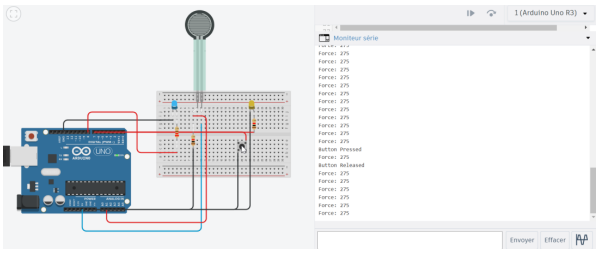

Figure 6: LED toggles when the button is released

```
const int forceSensorInput = A1;  // Analog pin for force sensor
const int pushbutton = 2;         // Digital pin for pushbutton
const int ledBlue = 8;            // LED controlled by force sensor
const int ledYellow = 7;          // LED controlled by pushbutton

bool pushButtonState = HIGH;
bool lastStatePushButton = HIGH;

void setup() {
    Serial.begin(9600);
    pinMode(ledBlue, OUTPUT);
    pinMode(ledYellow, OUTPUT);
    pinMode(pushbutton, INPUT_PULLUP); // Enable internal pull-up resistor
    for button
}

void loop() {
    int forceSensor = analogRead(forceSensorInput); // Read force sensor
    value
    pushButtonState = digitalRead(pushbutton); // Read button state

    Serial.print("Force: ");
    Serial.println(forceSensor);

    // Turn on LED1 if force sensor detects pressure
    digitalWrite(ledBlue, forceSensor > 50 ? HIGH : LOW);

    // Detect button press and release to toggle LED2
    if (pushButtonState == LOW && lastStatePushButton == HIGH) {
        Serial.println("Button Pressed");
    }
    else if (pushButtonState == HIGH && lastStatePushButton == LOW) {
        Serial.println("Button Released");
        digitalWrite(ledYellow, !digitalRead(ledYellow)); // Toggle LED2
    state
    }

    lastStatePushButton = pushButtonState;
    delay(400); // Small delay to prevent rapid toggling

}
```

Listing 3: Pushbutton with Internal Pull-Up

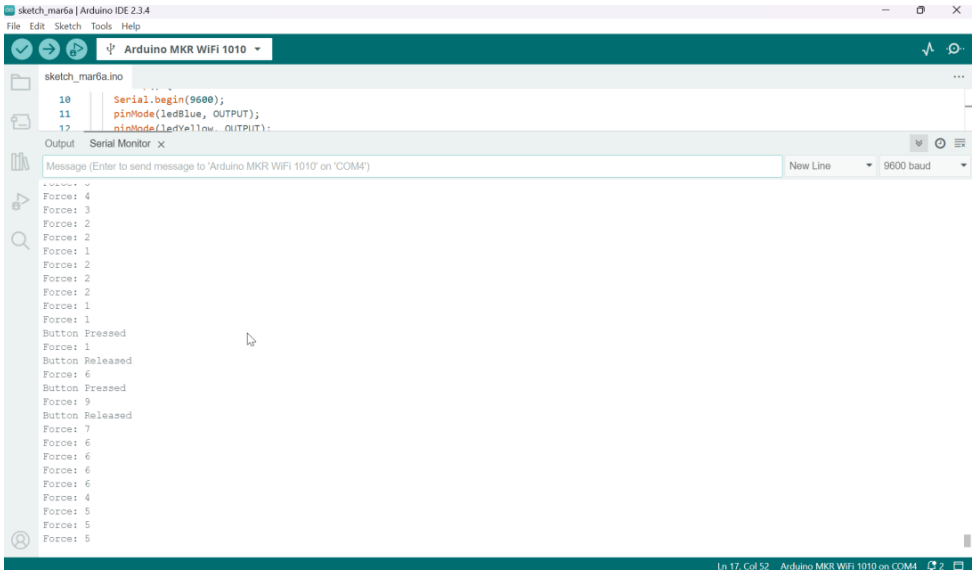

Figure 7: Pushbutton circuit in Arduino IDE

3. Force Sensor and Pushbutton

We tested how the system responds to button presses and sensor readings with two different delay settings:

1. Short Delay (100ms)

- The system quickly detects button presses and sensor changes.
- The yellow LED toggles almost immediately when the button is pressed.
- Rapid button presses are accurately registered.

2. Long Delay (1000ms)

- The system becomes slower in detecting inputs.
- Quick button presses may be missed if they happen between checks.
- The yellow LED takes longer to toggle after pressing the button.

A short delay (100ms) makes the system more responsive, while a long delay (1000ms) causes noticeable lag and missed inputs. The choice of delay depends on the need for real-time response versus system efficiency.

4. Interruptions

The pushbutton is already connected to an interrupt pin (2) on the Arduino, allowing immediate detection of button presses and releases without continuously checking its state in the main loop. Steps:

- Keep the same hardware setup.
- Connect the pushbutton to digital pin 2, which supports interrupts on the Arduino.
- Use attachInterrupt() to detect both pressing and releasing (CHANGE mode).
- Compare response time to the previous polling-based approach.

```
const int forceSensorInput = A1;  // Force sensor (analog)
const int pushbutton = 2;         // Pushbutton (digital, interrupt)
const int ledBlue = 8;            // LED1
const int ledYellow = 7;          // LED2

int lastButtonState = HIGH;  // Previous button state

void setup() {
    Serial.begin(9600);
    pinMode(ledBlue, OUTPUT);
    pinMode(ledYellow, OUTPUT);
    pinMode(pushbutton, INPUT_PULLUP);

    attachInterrupt(digitalPinToInterrupt(pushbutton), handleButtonPress,
   CHANGE);
}

void loop() {
    int buttonState = digitalRead(pushbutton);
    int forceSensor = analogRead(forceSensorInput);

    Serial.print("Force Sensor Value: ");
    Serial.print(forceSensor);
    Serial.print(" | Button State: ");
    Serial.println(buttonState == HIGH ? "HIGH" : "LOW");

    // Control the blue LED based on the force sensor value
    digitalWrite(ledBlue, forceSensor > 50 ? HIGH : LOW);

    // Toggle yellow LED on button release
    if (buttonState == HIGH && lastButtonState == LOW) {
        digitalWrite(ledYellow, !digitalRead(ledYellow));
    }

    lastButtonState = buttonState;  // Save state for next loop
    delay(100);  // Short delay for stability
}

void handleButtonPress() {
    Serial.println("Interrupt Triggered!");
    delay(100);  // Small debounce delay
}
```

Listing 4: Pushbutton with interruption

The pushbutton responds instantly, even during delays in the main loop.

5. Explanations

The CHANGE interrupt mode triggers an interrupt whenever the signal on the specified pin changes from HIGH to LOW or from LOW to HIGH. This allows immediate detection of both button presses and releases without continuously checking the pin state in the main loop.

Example:

When a pushbutton is connected to digital pin 2 with ‘attachInterrupt(digitalPinToInterrupt(2), handleButtonPress, CHANGE);‘, the ‘handleButtonPress()‘ function is executed instantly whenever the button is pressed or released. This ensures fast response times compared to traditional polling methods.

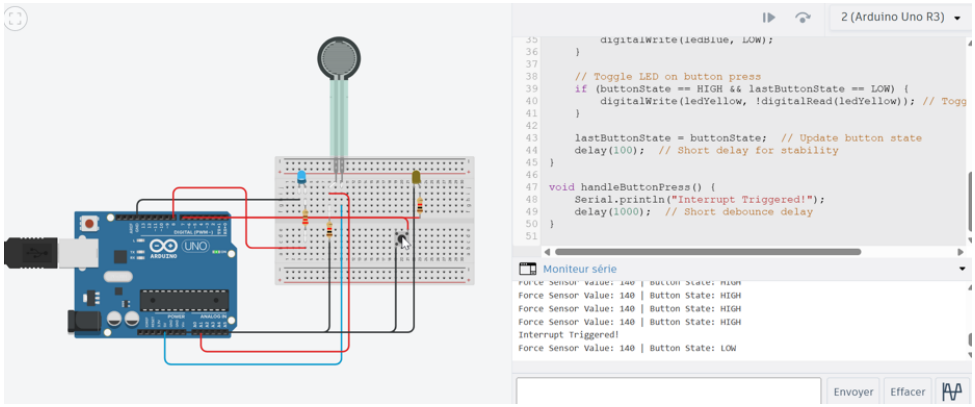

Figure 8: Force Sensor and Pushbutton using Interruption on Change

6. Conclusion

In this lab, we explored the principles and applications of force sensors and interrupts in microcontroller-based systems. We examined how force sensors detect pressure variations and how interrupts allow efficient event handling in Arduino projects. Through practical implementations, we demonstrated how these components can be integrated into circuits to enhance automation and responsiveness. Understanding these concepts is essential for developing interactive and responsive embedded systems, making them valuable tools in various real-world applications.