TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Laboratory work nr. 1:

Introduction to IoT Application Development Environment - Button-led

Brinza Cristian st. gr. FAF-212
Moraru Dumitru lect. univ

Chisinau 2024

The task of the laboratory work:

To design an application based on an MCU that would change the state of an LED upon detecting a button press.

- Use CamelCase coding conventions;
- Declare port variable like intended;
- The functionalities for each peripheral device (LED, button, LCD, keypad) should be implemented in separate files for the purpose of reuse in future projects.

Main functions/methods used to execute the task:

- In the **LED** Class:
  - **Constructor (LED(int pin)):** Initializes an **LED** object on a specified pin. It sets the pin mode to OUTPUT and initializes the internal state to represent the LED being off.
  - **on():** Sets the LED's state to **HIGH**, turning the LED on.
  - **off():** Sets the LED's state to **LOW**, turning the LED off.
  - **toggle():** Changes the LED's state from ON to OFF or from OFF to ON based on its current state. This function is crucial for toggling the LED each time the button is pressed and released.
- In the **Button** Class:
  - **Constructor (Button(int pin, unsigned long debounceDelay)):** Initializes a **Button** object on a specified pin with a debounce delay. It sets the pin mode to **INPUT** and initializes variables for managing the debounce logic.
  - **read():** Reads the button's state with debounce logic. It ensures that the button's state change is stable for a defined duration (**debounceDelay**) before considering it a valid press or release. This method returns the current debounced state of the button (either **HIGH** or **LOW**).
- In the **main.ino** Sketch:
  - **setup():** A function called once when the sketch starts. It's typically used to initialize variables, pin modes, or other libraries. In this refactored code, it might appear empty or contain initialization code for other components not shown in the example.
  - **loop():** The main program loop which runs continuously after **setup().** It reads the button state using **button.read()** and toggles the **LED** state with **led.toggle()** when a button press is detected. This function embodies the core logic of responding to user input with visual feedback.

Block Diagram:

The diagram is consisting of the following main blocks:

- **Main Program (main.ino):** The central point that initializes and controls other components (LED and Button) and manages the program flow[1].
- **LED Class:** Represents the LED control logic, including turning the LED on, off, and toggling its state.
- **Button Class:** Handles the debounce logic and state reading of a button press.
- **Arduino Board:** The physical layer where the LED and Button are connected.

Here is the diagram itself:

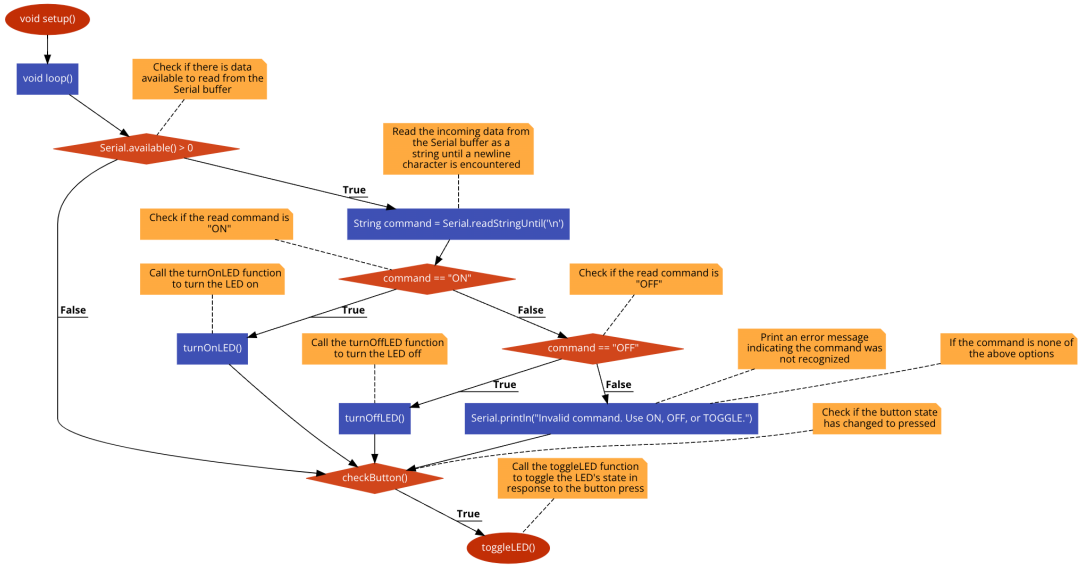

Figure 1 Program schema [1].

Simulated or real assembled electrical schematic diagram[2] :

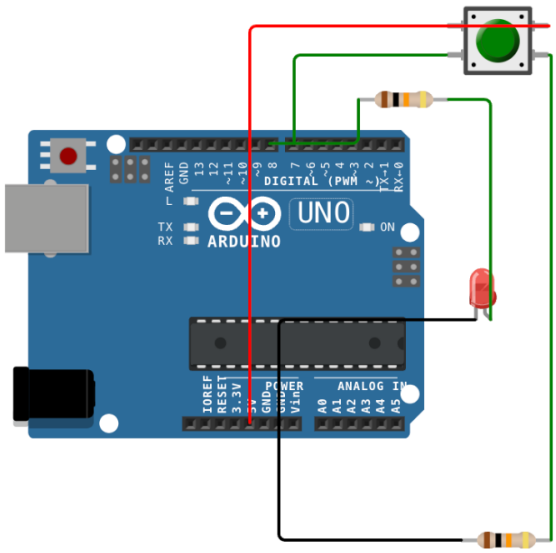

Figure 2 Phisical board schema [2].

Code:

main.ino:

```
#include "LEDControl.h" // Include the updated LED control header file for LED
operations
#include "ButtonControl.h" // Include the button control header file for button
operations

void setup() {
  Serial.begin(9600); // Initialize serial communication at 9600 baud rate for data
exchange between the Arduino and the computer
  Serial.println("SI 2024 FAF-212 Cristian Brinza lab1.1"); // Print a custom
message including session identifier to the Serial monitor
  Serial.println("Welcome to the Serial Monitor!"); // Print a welcome message to
the Serial monitor
  Serial.println("---------------------------------"); // Print a separator line to
the Serial monitor for better readability
  setupLED(); // Call the setupLED function to initialize the LED pin as an output
  setupButton(); // Call the setupButton function to initialize the button pin as
an input
}

void loop() {
  if (Serial.available() > 0) { // Check if there is data available to read from
the Serial buffer
    String command = Serial.readStringUntil('\n'); // Read the incoming data from
the Serial buffer as a string until a newline character is encountered
    if (command == "ON") { // Check if the read command is "ON"
      turnOnLED(); // Call the turnOnLED function to turn the LED on
    } else if (command == "OFF") { // Check if the read command is "OFF"
      turnOffLED(); // Call the turnOffLED function to turn the LED off
    } else if (command == "TOGGLE") { // Check if the read command is "TOGGLE"
      toggleLED(); // Call the toggleLED function to toggle the LED's current state
    } else { // If the command is none of the above options
      Serial.println("Invalid command. Use ON, OFF, or TOGGLE."); // Print an error
message indicating the command was not recognized
    }
  }

  if (checkButton()) { // Check if the button state has changed to pressed
    toggleLED(); // Call the toggleLED function to toggle the LED's state in
response to the button press
  }
}
```

ButtonControl.h:

```
// ButtonControl.h - Header file for button input handling without debouncing.
// This indicates that the file is intended to manage button inputs but does not
implement debouncing, which is a method to filter out false or repetitive triggers
due to the button's mechanical properties.

#ifndef ButtonControl_h // If ButtonControl_h hasn't been defined yet,
#define ButtonControl_h // define it now to prevent the inclusion of this header
file multiple times.

#include <Arduino.h> // Include the main Arduino library, which provides
functionality for reading and writing digital pins among other features.

#define BUTTON_PIN 7 // Define BUTTON_PIN as 7, assigning the button to digital pin
7 on the Arduino.

byte lastButtonState = LOW; // Declare a variable to store the last state of the
button, initialized to LOW.
// This assumes the use of a pull-up resistor configuration where the button is in
the LOW state when not pressed.

// Function to initialize the button pin
void setupButton() {
  pinMode(BUTTON_PIN, INPUT); // Set the BUTTON_PIN as an INPUT, configuring it to
read the state of a physical button.
}

// Function to check the button state without debouncing
bool checkButton() {
  byte currentButtonState = digitalRead(BUTTON_PIN); // Read the current state of
the button from BUTTON_PIN.
  if (currentButtonState != lastButtonState) { // Compare the current button state
to the last button state to detect changes.
    lastButtonState = currentButtonState; // Update the lastButtonState with the
current button state for future comparisons.
    if (currentButtonState == LOW) { // If the current button state is LOW,
indicating the button is pressed (assuming active low configuration).
      return true; // Return true, indicating a button press was detected.
    }
  }
  return false; // Return false if no change in button state was detected or if the
button was not pressed.
}

#endif // End of the conditional preprocessor directive, concluding the definitions
in this header file.
```

LEDControl.h:

```
// LEDControl.h - Header file description indicating it's been updated to include new functionalities for LED control

#ifndef LEDControl_h // If LEDControl_h hasn't been defined yet,
#define LEDControl_h // define it now to prevent duplicate inclusion of this header file

#include <Arduino.h> // Include the main Arduino header file for access to standard types and constants

#define LED_PIN 8 // Define the LED_PIN as 8, assigning the LED to digital pin 8 on the Arduino

byte ledState = LOW; // Declare a variable to store the current state of the LED, initialized to LOW (off)

void setupLED() {
  pinMode(LED_PIN, OUTPUT); // Set the LED_PIN as an OUTPUT, configuring it to control an LED
}

void toggleLED() {
  ledState = !ledState; // Toggle the ledState between HIGH and LOW
  digitalWrite(LED_PIN, ledState); // Apply the toggled state to the LED_PIN, turning the LED on or off
  Serial.print("LED is now "); Serial.println(ledState ? "ON" : "OFF"); // Print a confirmation message indicating the new state of the LED
}

void turnOnLED() {
  ledState = HIGH; // Set the ledState to HIGH (on)
  digitalWrite(LED_PIN, ledState); // Apply the HIGH state to the LED_PIN, turning the LED on
  Serial.println("LED turned ON"); // Print a message to the Serial monitor indicating that the LED has been turned on
}

void turnOffLED() {
  ledState = LOW; // Set the ledState to LOW (off)
  digitalWrite(LED_PIN, ledState); // Apply the LOW state to the LED_PIN, turning the LED off
  Serial.println("LED turned OFF"); // Print a message to the Serial monitor indicating that the LED has been turned off
}

bool getLEDState() {
  return ledState; // Return the current state of the ledState variable (either HIGH or LOW)
}
```

```
#endif // End of the conditional preprocessor directive, concluding the definitions
in this header file
```

Code link: https://wokwi.com/projects/388984304825841665

Screenshots of the simulation execution:

Here we have a screenshot of the WOKWI simulation[2] :

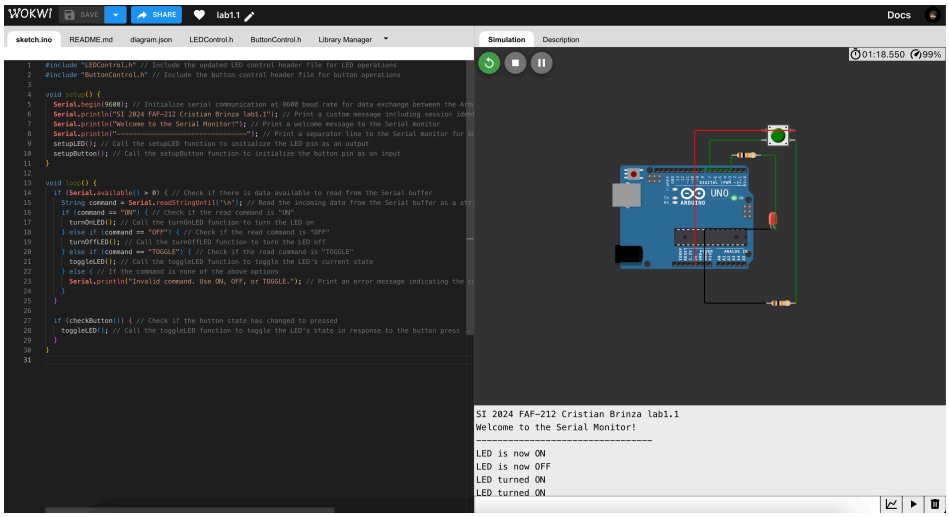

Figure 3 Screenshot of the simulation [3].

CONCLUSION

The lab work was all about making an LED light turn on and off when we press a button, or entered a console specific commend which was really cool to see in action. We learned to write our code in a neat way that makes it easy to read and fix if something goes wrong. Also, figuring out how to make sure the button press didn't mess up because of tiny electrical glitches was a big win. We used an Arduino board, which was super helpful because there's a lot of help out there if we got stuck. This project was a great lesson in how to keep everything organized, especially when working with code and electronics, setting us up nicely for more complicated projects down the road.

The task provided a practical experience in embedded system programming, emphasizing the importance of CamelCase coding conventions and modular code organization for peripheral device functionalities.

The laboratory work offered a comprehensive experience that extended beyond the mere toggling of an LED. It fostered an appreciation for the intricacies of embedded system design, from the meticulous organization of code to the thoughtful consideration of hardware-software interactions. The skills and insights gained from this project are invaluable, providing a strong foundation for future endeavors in the field of embedded systems.

BIBLIOGRAPHY

1 EDUCBA: *Introduction to Embedded Systems.* Cursuri electronice online ©2020 [citat 02.02.2021] Available: https://www.educba.com/what-is-embedded-systems/

2 SPARKFUN: *Driver motor.* Magazin online, ©2020 [citat 10.03.2020]. Available: https://www.sparkfun.com/datasheets/ Robotics/TB6612FNG.pdf.

3 ARDUINO: *Arduino UNO.* Site-ul oficial al modulelor Arduino, ©2020 [citat 30.03.2020]. Available: https://www.arduino.cc/.

4 EDUCBA: Introduction to Embedded Systems. Cursuri electronice online ¬©2020 [citat 02.02.2021] Available: https://www.educba.com/what-is-embedded-systems/

5 SPARKFUN: Driver motor. Magazin online, ¬©2020 [citat 10.03.2020]. Available: https://www.sparkfun.com/datasheets/ Robotics/TB6612FNG.pdf

6 ARDUINO: Arduino UNO. Site-ul oficial al modulelor Arduino, ¬©2020 [citat 30.03.2020]. Available: https://www.arduino.cc/