**TECHNICAL UNIVERSITY OF MOLDOVA**

**FACULTY OF COMPUTERS, INFORMATICS   
AND MICROELECTRONICS**

**DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS**

**Laboratory work nr. 2  
Operating Systems - Sequential**

A elaborat: **Brinza Cristian**

st. gr. FAF-212

A verificat: **Moraru Dumitru**

lect. univ

**Chișinău 2024**

# THE TASK OF THE LABORATORY WORK:

Realization of an application for MCU that will run at least 3 tasks - Sequential. The application will run at least 3 tasks among them:

* Button Led - LED state change upon detection of a button press.
* a second Flashing LED in the phase when the LED from the first Task is off
* Increment/decrement the value of a variable when pressing two buttons that will represent the number of recurrences/time during which the led from the second task will be in a state

The Idle task will be used to display the states in the program, such as LED status display, and message display when the buttons are pressed, an implementation being that when the button is pressed, a variable is set, and when the message is displayed - reset, implementing the provider/consumer mechanism.

Guidelines for implementation:

* To implement communication between Tasks as provider consumer, that is: the task that generates data, provider, stores the results in a global variable/signal, the task that uses this data, consumer, reads this variable/signal. For example: the UI task (LCD or Serial) takes the information from some global variables-signals and reports
* To follow the principles presented in the Sequential Systems course. Reasonable setting of recurrence to reduce processor load, setting the offset, in order to activate the tasks in the proper order NOTE: maximum attendance possible only at the presentation of physical performance!!
* The reporting task for Sequential using STDIO printf() to the LCD will be run in the infinite loop/IDLE because it is based on a spin lock and could block interrupts, so classic sequential - printf & delay in main loop,

**Timing:**

* mark 5 - the simple demonstration application of sequential system and FreeRTOS
* 0.5 - for each sequential task (3 x 0.5 x 2 = 3.0)
* 1.0 - For improvising an additional Task implementing a task different from the one in the laboratory
* 1.0 - for demonstrating physical implementation evidence

**Penalties:**

* 1 - penalty for each week late from the deadline
* 1 - penalty for non-compliance with the report format
* 1 - for NOT-using STDIO for reporting messages to the LCD

# PROGRESS OF THE WORK

## 1 Main functions/methods used to execute the task

Explication about this chapterIn this chapter I will explain the functionality of diferent parts of the executed task:

1. In the **sketch.ino** file:

* **setup() Function:** Initializes the LCD, LEDs, buzzer, and buttons. This function prepares the LCD display for showing status messages and sets up the LEDs and buzzer for indicating the blinking status and sound alert, respectively.
* **loop() Function:** Handles button input and controls output. This function continuously checks the state of each button and controls the LEDs and buzzer accordingly. It updates the LCD with the current state, such as the LED status and the blink interval.

1. Following i will describe how the code works:

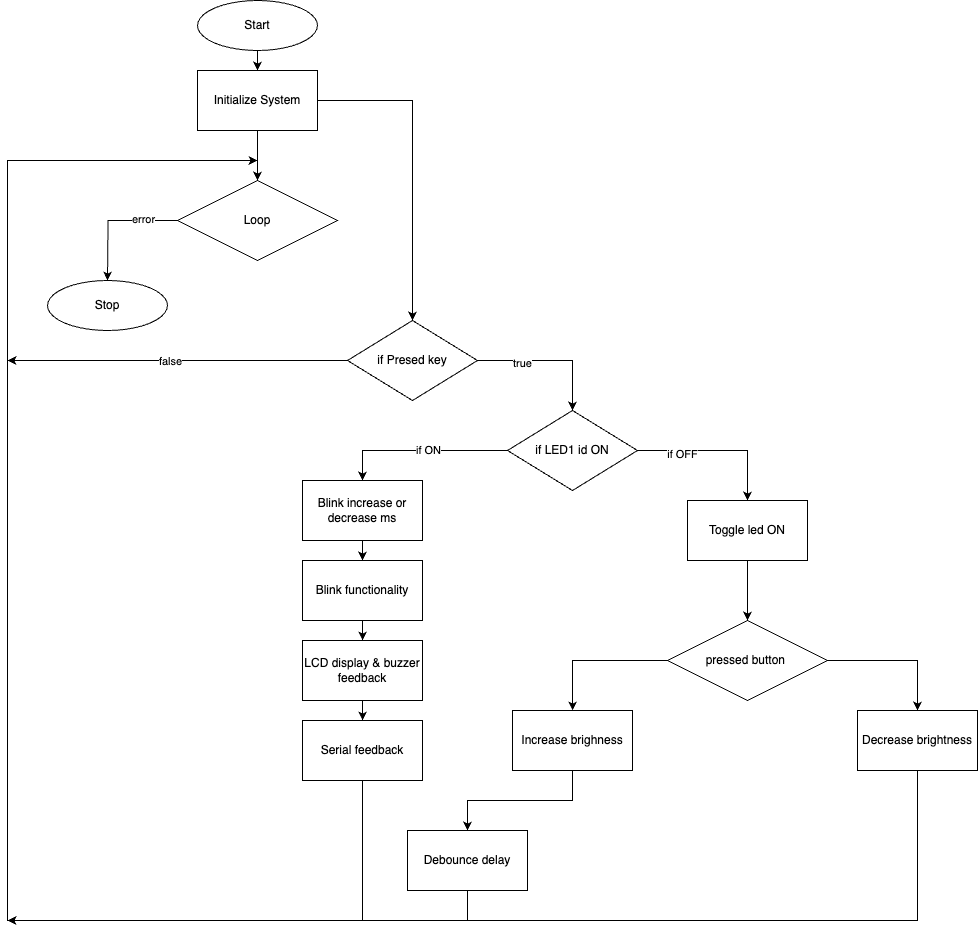
* **setup() Function**: Initializes the system's digital pins for input (buttons) and output (LEDs and buzzer). Starts serial communication at 9600 baud for debugging and user interaction via the serial monitor. Initializes the LCD display with 16 characters and 2 rows. Prints a startup message on the serial monitor and updates the LCD with the initial system state (LED off, default blink duration, and brightness).
* **loop() Function** Continuously checks for button presses to toggle the LED, adjust blink duration and brightness, and play a theme. Handles each task through dedicated functions and includes a placeholder for an idle task for future background operations.
* **handleButtonLedTask()** Toggles the main LED on or off based on a button press, updates the LED's brightness accordingly, and provides audible feedback with a beep. Updates the LCD and serial monitor with the current action and system state.
* **handleBlinkSecondLedTask()** Manages the blinking of a secondary LED at intervals defined by blinkDuration. Ensures the secondary LED is off when the main LED is on.
* **handleVariableIncrementDecrementTask()** Adjusts the blink duration for the second LED based on button inputs, with limits to prevent the duration from becoming too fast or too slow. Provides audible feedback and updates the display and serial monitor with the new blink duration.
* **handleBrightnessControlTask()** Adjusts the main LED's brightness level based on button inputs, with limits to keep the brightness within the 0-255 range (PWM). Applies the new brightness level to the LED if it is currently on, provides audible feedback, and updates the display and serial monitor.
* **handlePlayThemeTask()** Toggles the playback of a predefined melody (Harry Potter theme) based on a button press. Stops the melody if the button is pressed again or the melody finishes playing.
* **playTheme()** Plays the melody defined by arrays melody[] and noteDurations[] through the buzzer. Uses tone() to play each note for its duration and delay() to pause between notes, ensuring each note is distinct. Stops playing if the theme playback is toggled off during execution.
* **beepBuzzer()** Generates a short beep sound using the buzzer, indicating a button press or action.
* **updateLcdAndSerial()** Clears the LCD display and prints the current LED state, brightness, and blink duration. Constructs a detailed message and sends it to the serial monitor for user feedback.
* **handleIdleTask()** Placeholder for implementing background tasks or monitoring, not used in the current version of the sketch.

## Block Diagram

The diagram is consisting of the following main blocks:

* **Main Program** (sketch.ino): Acts as the core of the system, initializing the Arduino pins for the LCD, LEDs, buzzer, and buttons. It orchestrates the operational flow, handling inputs from the buttons, updating the LCD with status information, and managing the LEDs' blinking and buzzer sounds based on user interaction.
* **Setup Function:** A vital component of the Main Program, responsible for setting up the LCD, defining pin modes for LEDs, buzzer, and buttons, and displaying the initial status on the LCD. This step ensures all hardware components are correctly initialized and ready for use.
* **Loop Function**: Represents the continuous execution cycle of the Main Program, dealing with the logic of button presses, controlling the LEDs' blinking behavior, and activating the buzzer when necessary. It also updates the LCD with the current blink interval and whether the LEDs are on or off.
* **Button Input Handling**: Manages the detection and response to button presses. This part of the program is responsible for reading the state of each button, updating the blink interval based on increments or decrements, and toggling the state of the main LED.
* **LED Blinking and Sound Alert**: Controls the blinking of the second LED and the activation of the buzzer. When the second LED blinks, the buzzer emits a short sound, providing an audible alert to accompany the visual indicator.
* **LCD Feedback Display**: Handles the real-time updating of the LCD to reflect changes in the system's state, such as the current blink interval or the main LED's state. It provides feedback to the user based on their interactions and the system's current status.
* **LED and Buzzer Status Indication**: Manages the visual and auditory feedback of the system. The LEDs indicate the system's activity or status, with the main LED showing whether the system is active and the second LED blinking according to the set interval. The buzzer complements this by providing an audible signal whenever the second LED blinks.

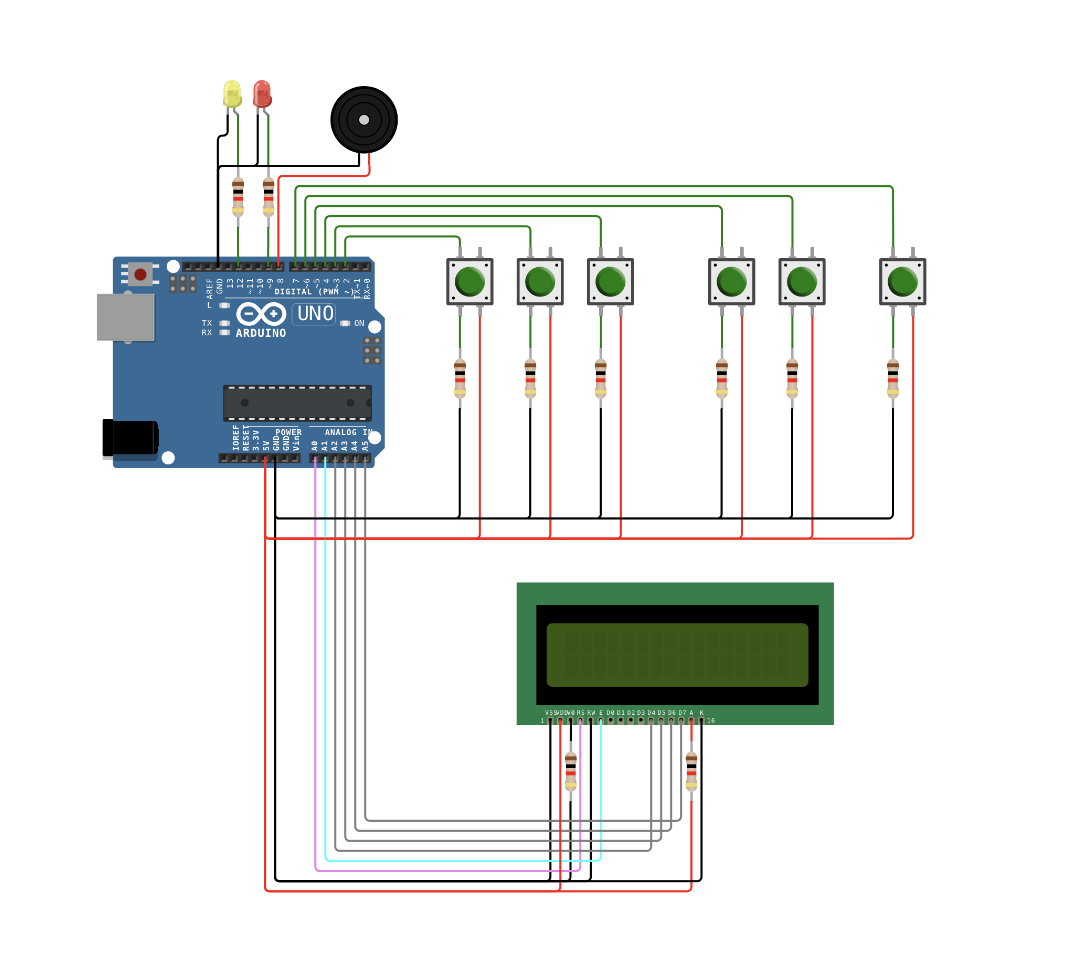
The Figure 1 depicted the UML program flow.



**Figure 1** Program schema.

## 3 Simulated or real assembled electrical schematic diagram:

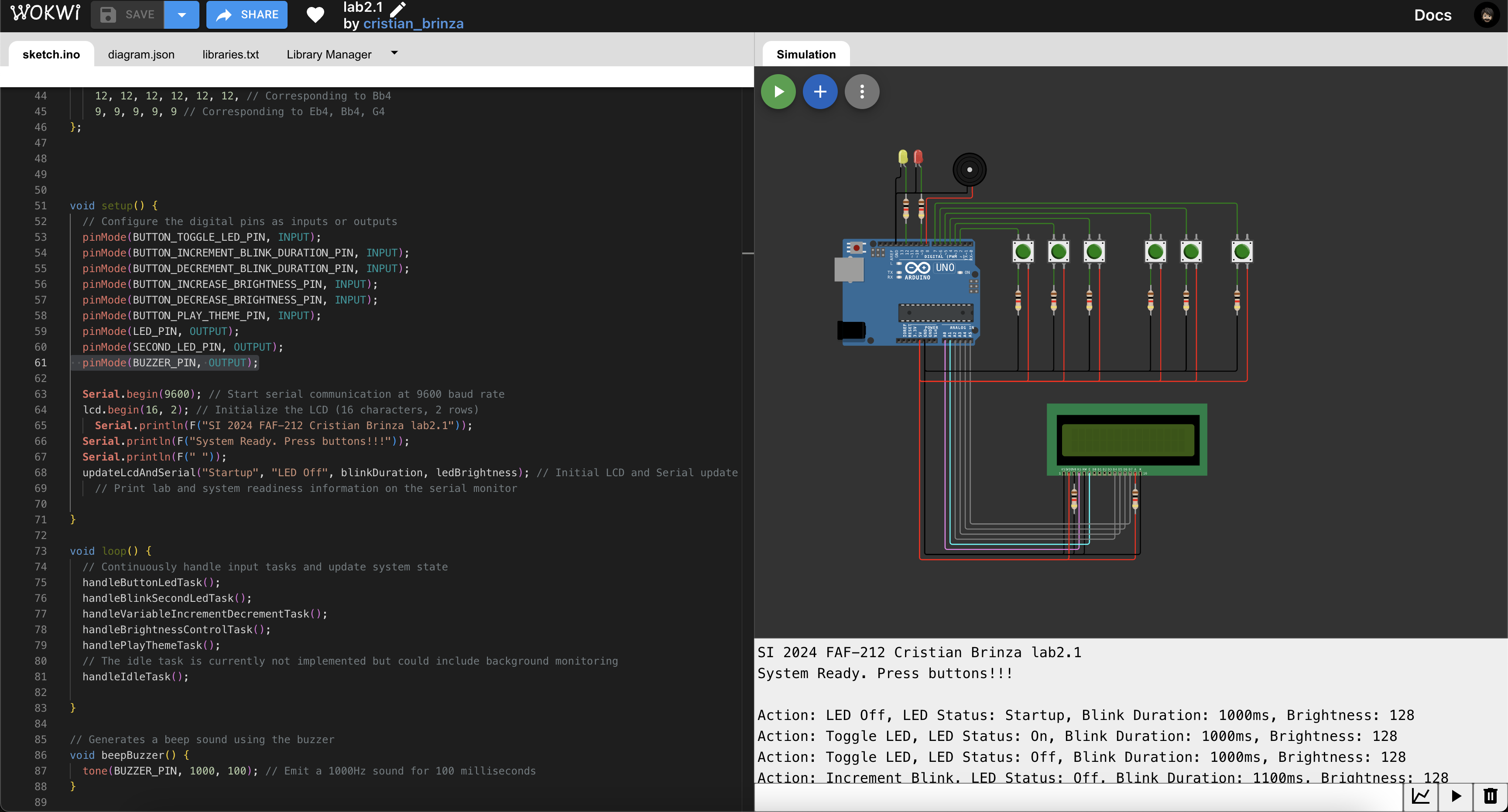
The Figure 2 depicted the phisical board schema.



**Figure 2** Phisical board schema.

## 4 Screenshots of the simulation execution:

The Figure 3 depicted a sceenshot of the WOKWI simulation :



**Figure 3** Screenshot of the simulation.

## CONCLUSION

The lab work was all about making an LED light turn on and off and blick when we click the corespppnding button, which was really cool to see in action. By utilizing the LiquidCrystal library alongside basic input and output components, we've constructed a dynamic system that interacts through user-initiated actions, reflecting outcomes on an LCD and via LED signals. 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

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 Arduino, buttons, LCD, 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 [quote 10.02.2024] Available: <https://www.educba.com/what-is-embedded-systems/>
2. ARDUINO: *Arduino UNO.* The official website of Arduino modules, ©2020 [quote 10.02.2024]. Available: <https://www.arduino.cc/>.
3. TUTORIALSPOINT: *Embedded Systems Tutorial*. Comprehensive guide for beginners and professionals to learn Embedded System basics to advanced concepts, including microcontrollers, processors, and real-time operating systems. ©2023 [quote 10.02.2024] Available: <https://www.tutorialspoint.com/embedded_systems/index.htm>
4. GURU99: *Embedded Systems Tutorial: What is, History & Characteristics*. A detailed introduction to embedded systems, covering microcontrollers, microprocessors, and the architecture of embedded systems, as well as their applications and advantages. ©2023 [quote 10.02.2024] Available: <https://www.guru99.com/embedded-systems-tutorial.html>
5. JAVATPOINT: *Embedded Systems Tutorial.* Offers basic and advanced concepts of Embedded System, designed for beginners and professionals. ©2023 [quote 10.02.2024] Available: <https://www.javatpoint.com/embedded-systems-tutorial>
6. DEEPBLUE*: Embedded Systems Tutorials Introduction* | Embedded Systems Online Course. ©2023 [quote 10.02.2024] Available: <https://deepbluembedded.com/embedded-systems-tutorials/>

**APPENDIX 1**

Code of main.ino:

// Include the standard input-output library for general I/O operations

#include <stdio.h>

#include <Arduino.h>

#include <LiquidCrystal.h>

// Define constants for pin assignments to improve readability and maintenance

#define BUTTON\_TOGGLE\_LED\_PIN 2

#define BUTTON\_INCREMENT\_BLINK\_DURATION\_PIN 3

#define BUTTON\_DECREMENT\_BLINK\_DURATION\_PIN 4

#define BUTTON\_INCREASE\_BRIGHTNESS\_PIN 5

#define BUTTON\_DECREASE\_BRIGHTNESS\_PIN 6

#define LED\_PIN 9 // PWM capable pin for LED brightness control

#define SECOND\_LED\_PIN 12

#define BUZZER\_PIN 8 // Buzzer pin for audible feedback

#define BUTTON\_PLAY\_THEME\_PIN 7

// Initialize the LCD with the analog interface pins

LiquidCrystal lcd(A0, A1, A2, A3, A4, A5);

// Global variables for system state management

volatile bool ledState = false; // Tracks the on/off state of the main LED

int blinkDuration = 1000; // Duration between blinks for the second LED in milliseconds

int ledBrightness = 128; // Brightness level of the main LED (0-255)

bool isPlayingTheme = false; // Tracks whether the theme is currently being played

// Note frequencies for the Harry Potter theme, based on the notes' musical pitch

// Notes in the melody:

int melody[] = {

392, 0, 392, 0, 392, 0, // G4

311, 0, 466, 0, 392, 0, // Eb4, Bb4, G4

784, 0, 784, 0, 784, 0, // G5

698, 0, 659, 0, 622, 0, 659, 0, // F5, E5, Eb5, E5

415, 0, 554, 0, 523, 0, 494, 0, // Ab4, C#5, C5, B4

466, 0, 466, 0, 466, 0, // Bb4

311, 0, 466, 0, 392 // Eb4, Bb4, G4

};

// Durations of the notes (in terms of 1/4 notes):

// For example, a quarter note would be 4, half note would be 2.

int noteDurations[] = {

12, 12, 12, 12, 12, 12, // Corresponding to G4

9, 9, 9, 9, 9, 9, // Corresponding to Eb4, Bb4, G4

12, 12, 12, 12, 12, 12, // Corresponding to G5

12, 12, 12, 12, 12, 12, 12, 12, // Corresponding to F5, E5, Eb5, E5

9, 9, 9, 9, 9, 9, 9, 9, // Corresponding to Ab4, C#5, C5, B4

12, 12, 12, 12, 12, 12, // Corresponding to Bb4

9, 9, 9, 9, 9 // Corresponding to Eb4, Bb4, G4

};

void setup() {

// Configure the digital pins as inputs or outputs

pinMode(BUTTON\_TOGGLE\_LED\_PIN, INPUT);

pinMode(BUTTON\_INCREMENT\_BLINK\_DURATION\_PIN, INPUT);

pinMode(BUTTON\_DECREMENT\_BLINK\_DURATION\_PIN, INPUT);

pinMode(BUTTON\_INCREASE\_BRIGHTNESS\_PIN, INPUT);

pinMode(BUTTON\_DECREASE\_BRIGHTNESS\_PIN, INPUT);

pinMode(BUTTON\_PLAY\_THEME\_PIN, INPUT);

pinMode(LED\_PIN, OUTPUT);

pinMode(SECOND\_LED\_PIN, OUTPUT);

pinMode(BUZZER\_PIN, OUTPUT);

Serial.begin(9600); // Start serial communication at 9600 baud rate

lcd.begin(16, 2); // Initialize the LCD (16 characters, 2 rows)

Serial.println(F("SI 2024 FAF-212 Cristian Brinza lab2.1"));

Serial.println(F("System Ready. Press buttons!!!"));

Serial.println(F(" "));

updateLcdAndSerial("Startup", "LED Off", blinkDuration, ledBrightness); // Initial LCD and Serial update

// Print lab and system readiness information on the serial monitor

}

void loop() {

// Continuously handle input tasks and update system state

handleButtonLedTask();

handleBlinkSecondLedTask();

handleVariableIncrementDecrementTask();

handleBrightnessControlTask();

handlePlayThemeTask();

// The idle task is currently not implemented but could include background monitoring

handleIdleTask();

}

// Generates a beep sound using the buzzer

void beepBuzzer() {

tone(BUZZER\_PIN, 1000, 100); // Emit a 1000Hz sound for 100 milliseconds

}

// Updates the LCD display and prints a corresponding message to the serial monitor

void updateLcdAndSerial(String status, String action, int duration, int brightness) {

// Clear the LCD and prepare to display the updated state

lcd.clear();

// Display the LED state on the first line with brightness

lcd.print("LED:");

lcd.print(ledState ? "On " : "Off");

lcd.print(" Br:");

lcd.print(ledBrightness);

// Move to the second line to display the blink duration

lcd.setCursor(0, 1);

lcd.print("Blink:");

lcd.print(duration);

lcd.print("ms");

// Construct and send a detailed message to the serial monitor

Serial.println("Action: " + action + ", LED Status: " + status + ", Blink Duration: " + String(duration) + "ms, Brightness: " + String(brightness));

}

// Handles the toggling of the main LED based on button input

void handleButtonLedTask() {

static bool lastButtonState = LOW; // Remember the last state to detect changes

bool currentButtonState = digitalRead(BUTTON\_TOGGLE\_LED\_PIN);

// Check for state change to HIGH (button press)

if (currentButtonState != lastButtonState && currentButtonState == HIGH) {

ledState = !ledState; // Toggle the LED state

analogWrite(LED\_PIN, ledState ? ledBrightness : 0); // Update LED brightness accordingly

beepBuzzer(); // Provide audible feedback for the action

updateLcdAndSerial(ledState ? "On" : "Off", "Toggle LED", blinkDuration, ledBrightness); // Update displays

}

lastButtonState = currentButtonState; // Update the last known state

}

// Manages the blinking of a secondary LED based on the current blink duration

void handleBlinkSecondLedTask() {

static unsigned long previousMillis = 0;

if (!ledState && (millis() - previousMillis >= blinkDuration)) {

previousMillis = millis();

digitalWrite(SECOND\_LED\_PIN, !digitalRead(SECOND\_LED\_PIN)); // Toggle the LED's state

} else if (ledState) {

digitalWrite(SECOND\_LED\_PIN, LOW); // Ensure the secondary LED is off when the main LED is on

}

}

// Adjusts the blink duration based on button inputs for incrementing/decrementing

void handleVariableIncrementDecrementTask() {

if (digitalRead(BUTTON\_INCREMENT\_BLINK\_DURATION\_PIN) == HIGH) {

blinkDuration = max(100, blinkDuration - 100); // Decrease duration for faster blinking

beepBuzzer();

updateLcdAndSerial(ledState ? "On" : "Off", "Decrement Blink", blinkDuration, ledBrightness);

delay(200); // Debounce delay

} else if (digitalRead(BUTTON\_DECREMENT\_BLINK\_DURATION\_PIN) == HIGH) {

blinkDuration = min(2000, blinkDuration + 100); // Increase duration for slower blinking

beepBuzzer();

updateLcdAndSerial(ledState ? "On" : "Off", "Increment Blink", blinkDuration, ledBrightness);

delay(200); // Debounce delay

}

}

// Adjusts the main LED's brightness based on button inputs

void handleBrightnessControlTask() {

if (digitalRead(BUTTON\_INCREASE\_BRIGHTNESS\_PIN) == HIGH) {

ledBrightness = min(255, ledBrightness + 15); // Increase brightness

if (ledState) {

analogWrite(LED\_PIN, ledBrightness); // Apply only if the LED is on

}

beepBuzzer();

updateLcdAndSerial(ledState ? "On" : "Off", "Increase Brightness", blinkDuration, ledBrightness);

delay(200); // Debounce delay

} else if (digitalRead(BUTTON\_DECREASE\_BRIGHTNESS\_PIN) == HIGH) {

ledBrightness = max(0, ledBrightness - 15); // Decrease brightness

if (ledState) {

analogWrite(LED\_PIN, ledBrightness); // Apply only if the LED is on

}

beepBuzzer();

updateLcdAndSerial(ledState ? "On" : "Off", "Decrease Brightness", blinkDuration, ledBrightness);

delay(200); // Debounce delay

}

}

void handlePlayThemeTask() {

static bool lastButtonState = LOW;

bool currentButtonState = digitalRead(BUTTON\_PLAY\_THEME\_PIN);

// Toggle the theme playback on button press

if (lastButtonState == HIGH && currentButtonState == LOW) {

isPlayingTheme = !isPlayingTheme;

if (isPlayingTheme) {

playTheme();

} else {

noTone(BUZZER\_PIN); // Stop playing

}

}

lastButtonState = currentButtonState;

}

void playTheme() {

for (int thisNote = 0; thisNote < 20; thisNote++) {

// To calculate the note duration, take one second divided by the note type.

// e.g., quarter note = 1000 / 4, eighth note = 1000/8, etc.

int noteDuration = 1000 / noteDurations[thisNote];

tone(BUZZER\_PIN, melody[thisNote], noteDuration);

// To distinguish the notes, set a minimum time between them.

// The note's duration + 30% seems to work well:

int pauseBetweenNotes = noteDuration \* 1.30;

delay(pauseBetweenNotes);

// Stop the tone playing:

noTone(BUZZER\_PIN);

// Check if the theme should stop playing

if (!isPlayingTheme) {

break;

}

}

}

// An optional idle task function, currently unused but could be implemented for background tasks

void handleIdleTask() {

// Placeholder for future implementation

}