Ministry of Education and Research of the Republic of Moldova
Technical University of Moldova
Department of Software and Automation Engineering

REPORT

Laboratory work No. 6.1

Course: Embedded Systems

Student: Ostafi Eugen, FAF-222

Verified: asist. univ., Martiniuc A.

Chișinău 2025

ANALYSIS OF THE FIELD

Overview of Technologies and Application Context

This laboratory work centers on the implementation of a finite state machine (FSM) to control the state of an LED based on input from a push-button. The system uses a modular structure running on an Arduino Uno and allows transitions between OFF, ON, and BLINK states, with each press of a debounced button triggering the next state in the cycle.

Communication with the user is achieved using printf() redirection to the serial monitor, implemented via STDIO. The FSM also includes an auto-timeout feature that resets the LED state to OFF after 10 seconds of inactivity. This ensures energy efficiency and introduces basic fail-safe behavior. The debounce logic ensures stable state transitions even in the presence of mechanical switch noise.

The main components of the application include:

- **FSM Task**: Tracks the current state and handles transitions and behavior logic for each state.
- **Display Task**: Reports the current FSM state using printf(), emulating console feedback.
- **Button Input Module**: Handles debounce logic and triggers transitions only on clean button presses.
- **LED Module**: Controls the LED pin based on current FSM state.

Hardware Components:

- **Arduino Uno**: Central microcontroller platform for implementing and running the FSM.
- **Push-Button**: Input device used to change FSM states with each press.
- **LED**: Visual output device indicating the current FSM state.
- **Breadboard & Jumper Wires**: Used to prototype the hardware connections.
- **USB Power**: Used to power the Arduino and attached components.

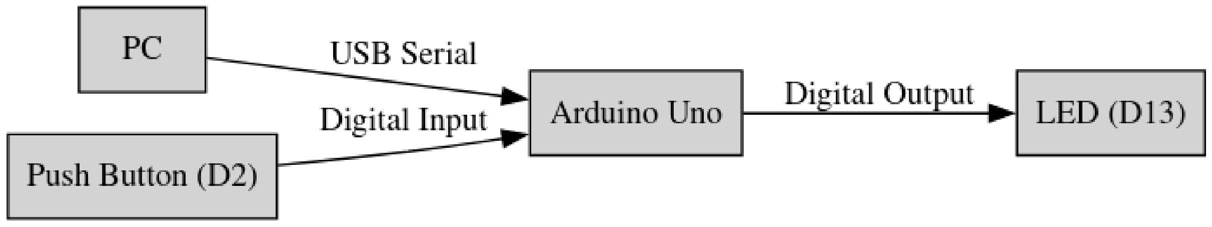


Figure 1. Hardware Interface Diagram

Software Components:

- **Arduino IDE / PlatformIO:** Development environment used to write, compile, and upload the Arduino code.
- **STDIO Library:** Utilized for formatted input and output to the serial terminal, making it possible to view real-time data on the computer.

The system architecture is divided into clear software modules:

- **fsm.cpp**: Handles FSM logic and state transitions.
- **button.cpp**: Implements button reading and debounce.
- **led.cpp**: Abstracts LED ON/OFF behavior.
- **display.cpp**: Handles STDIO output using printf().

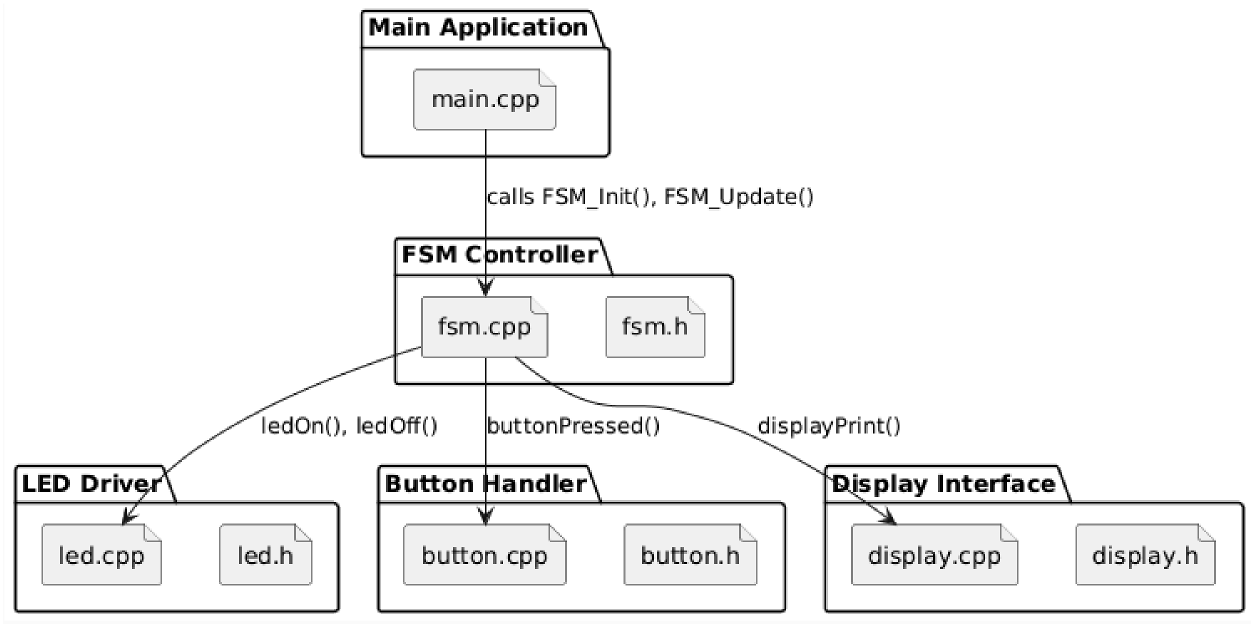


Figure 2. Software Interface Diagram

PROJECT DESIGN

Architectural Overview

The system consists of the following components:

- **Microcontroller (Arduino Uno):** Acts as the central controller, managing command parsing, relay control, and message display through modular software layers.
- **Relay Module:** Serves as the binary actuator, allowing the microcontroller to control devices such as lights or fans through a digital signal.
- **Serial Monitor:** Serves as both the command input interface and the feedback/output display for status messages, using redirected STDIO via printf().

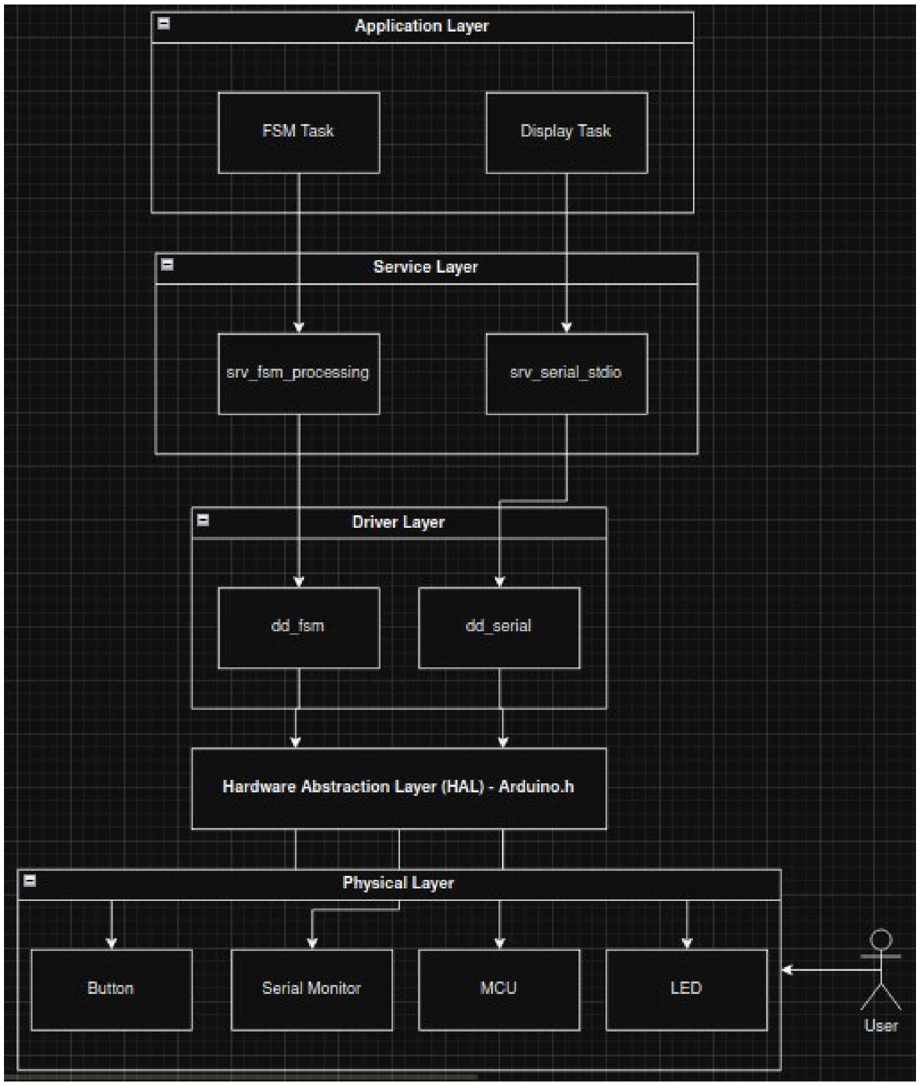


Figure 3. Architectural Overview Diagram

Operational Workflow:

- **System initialization**: Sets up LED, button (with pull-up), and STDIO (printf()).
- **Main loop**: Continuously updates the FSM and checks for button input.
- **Button press handling**: Cycles states OFF → ON → BLINK → OFF.
- **State actions**: Sets LED output (OFF, ON, or blinking).
- **Timeout check**: Returns to OFF after 10s of inactivity.
- **Loop repeats**: FSM logic runs every 10 ms.

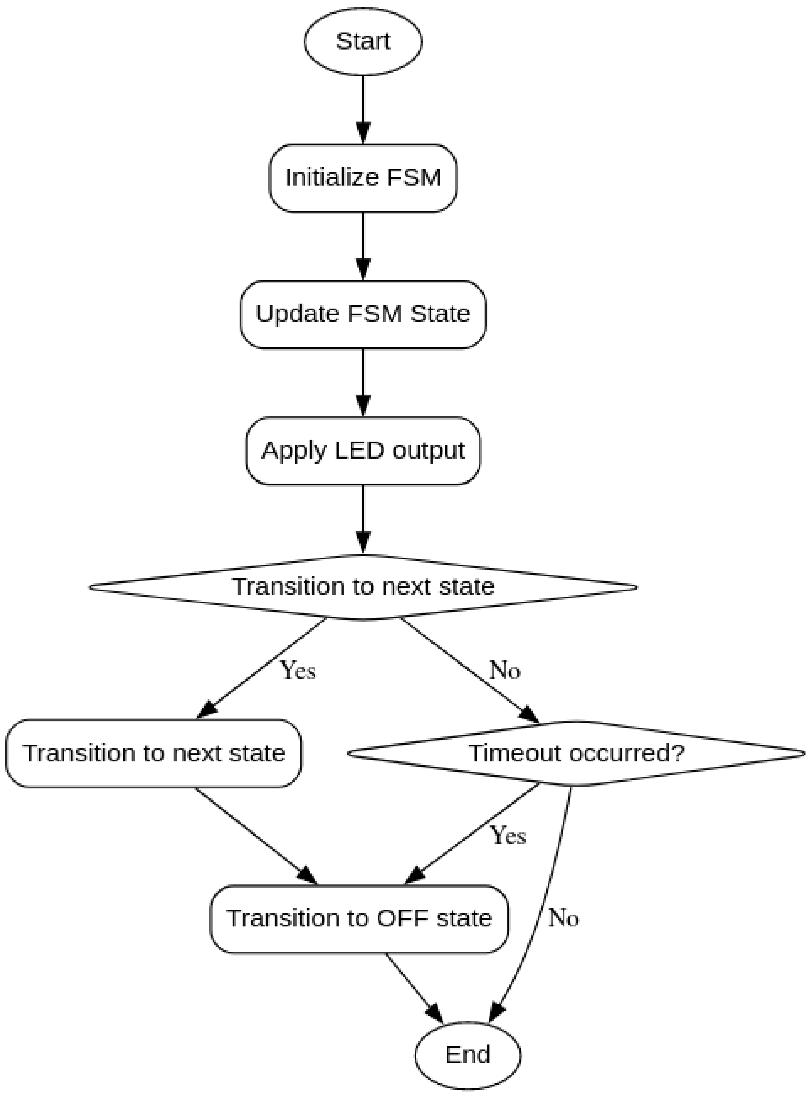


Figure 4. System Operation Diagram

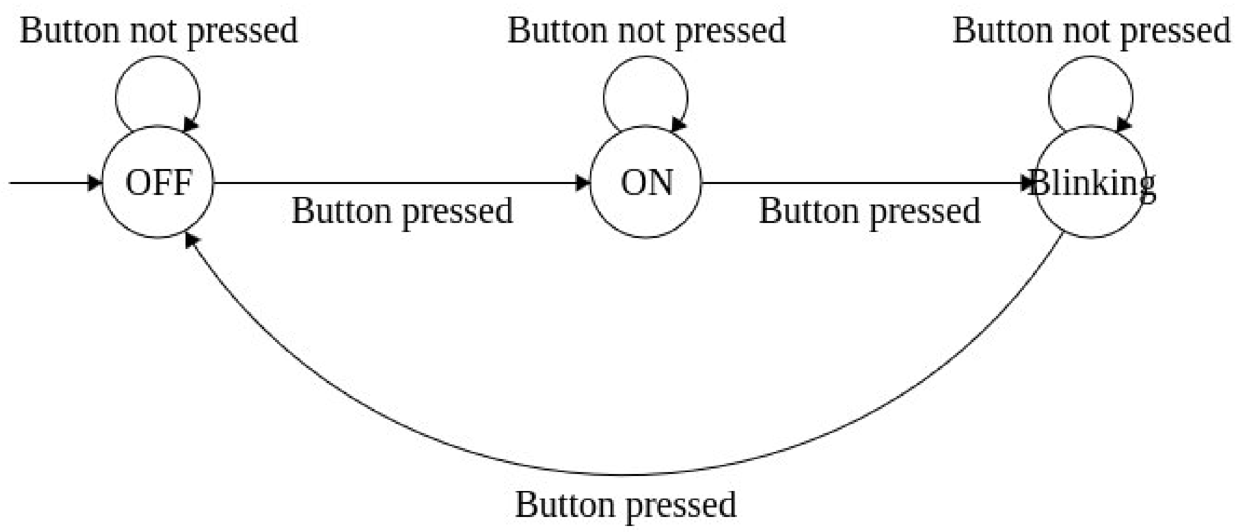
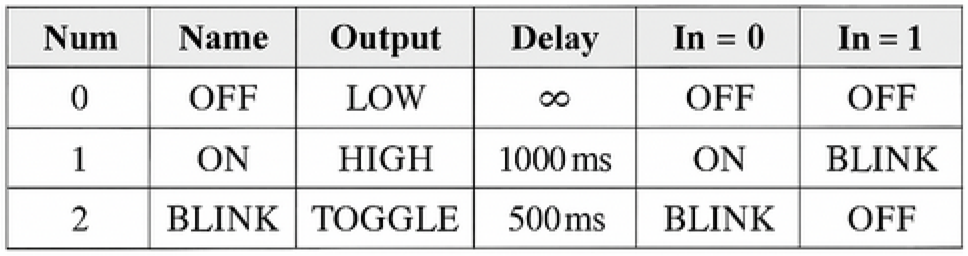


Figure 4. Finite State Machine Diagram for LED Control



Num	Name	Output	Delay	In = 0	In = 1
0	OFF	LOW	∞	OFF	OFF
1	ON	HIGH	1000 ms	ON	BLINK
2	BLINK	TOGGLE	500 ms	BLINK	OFF

Figure 5. Finite State Machine Diagram for LED Control

Electrical Schematic

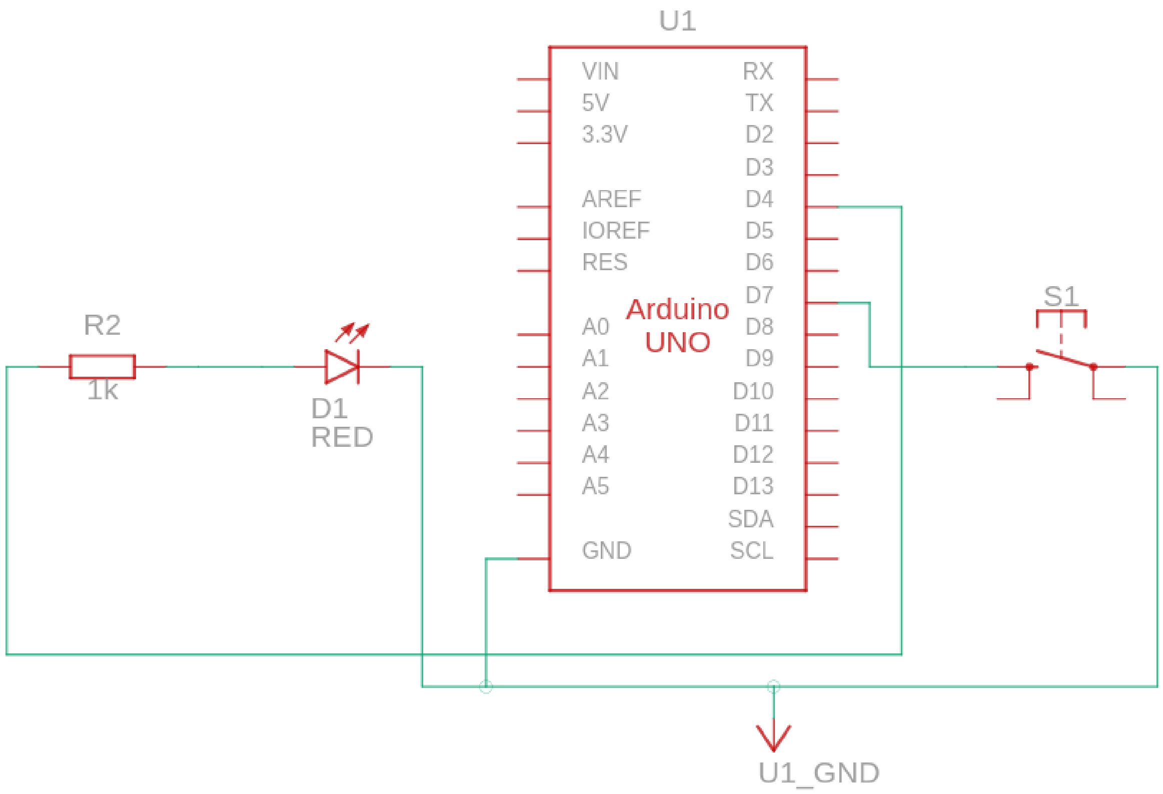


Figure 6. Electrical Schematic

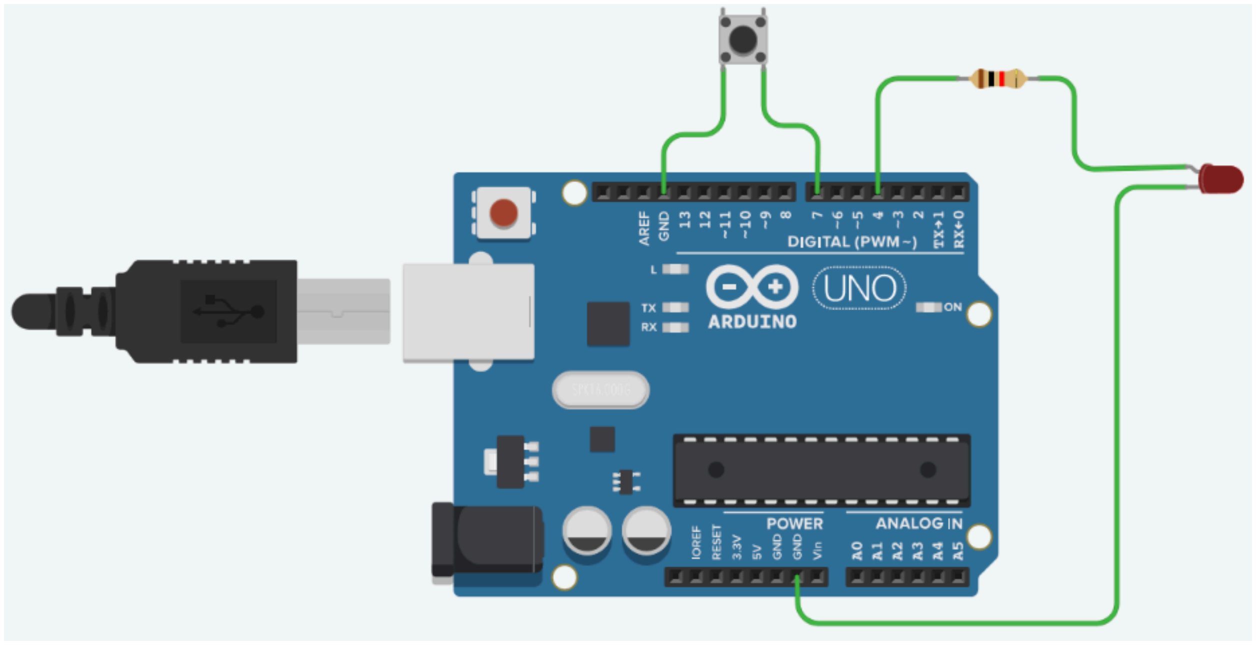


Figure 7. Circuit connections

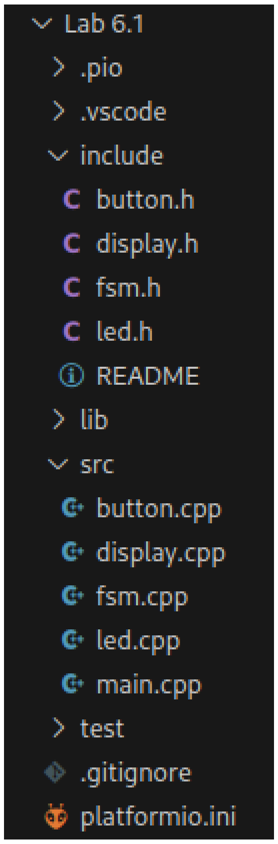
The schematics include:

- **Button** connected to **pin D7** with internal pull-up enabled.
- **LED** connected to **pin D4** with a current-limiting resistor.
- **Ground** common between button and LED components.

Modular Implementation

The project is structured into separate modules:

- **main.cpp**: Reads serial input and directs command flow.
- **relay_driver.cpp / .h**: Initializes and sets the state of the relay pin.
- **relay.cpp / .h**: Provides high-level relay ON/OFF interface.
- **display.cpp / .h**: Prints system status messages to the serial monitor.



```
Lab 6.1
  .pio
  .vscode
  include
    button.h
    display.h
    fsm.h
    led.h
    README
  lib
  src
    button.cpp
    display.cpp
    fsm.cpp
    led.cpp
    main.cpp
  test
  .gitignore
  platformio.ini
```

Figure 8. Modular Project Structure

Functional Block Diagrams

To visualize the way the program functions, I created these functional block diagram in which I illustrated the responsibilities of the main functions in the system

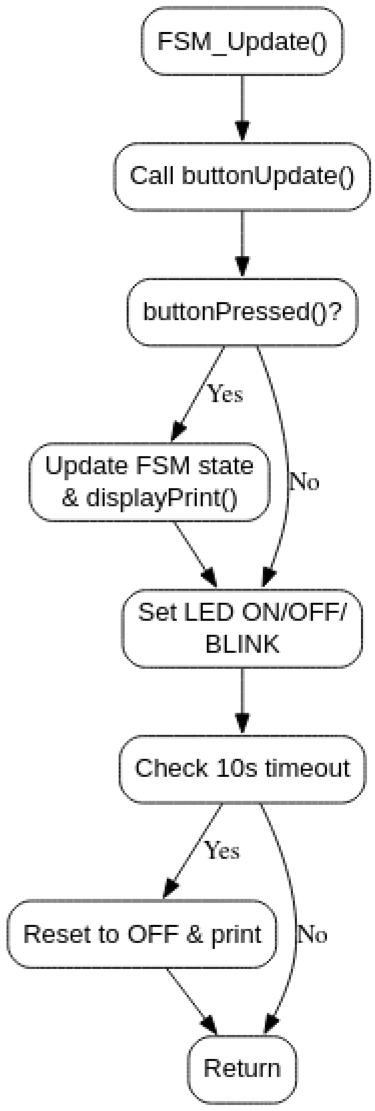


Figure 9. FSM Update Function Block Diagram

This function is called in the main loop to manage the entire FSM logic. It handles button press detection, state transitions, LED output logic and inactivity timeout

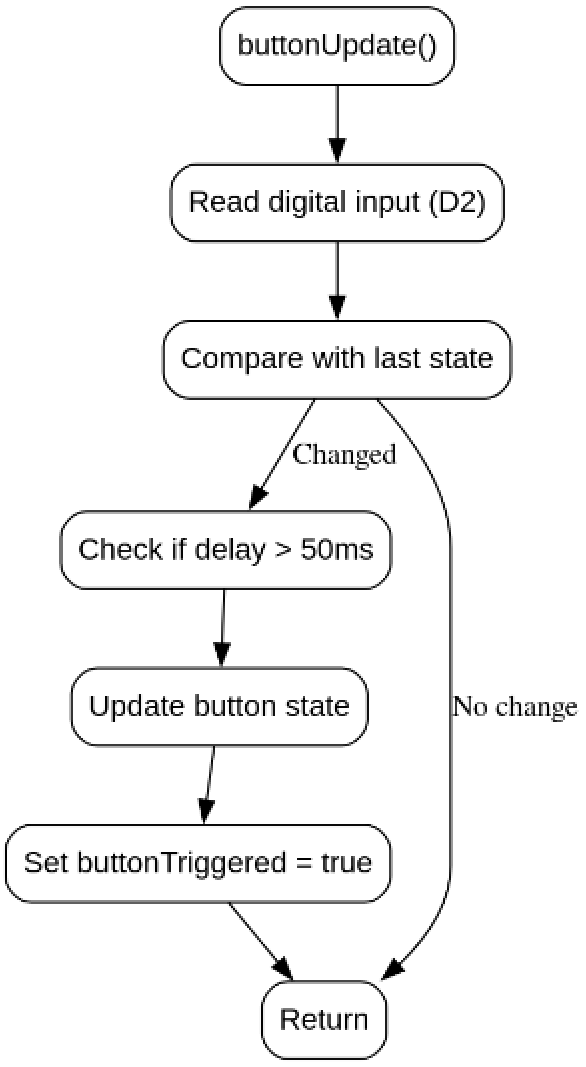
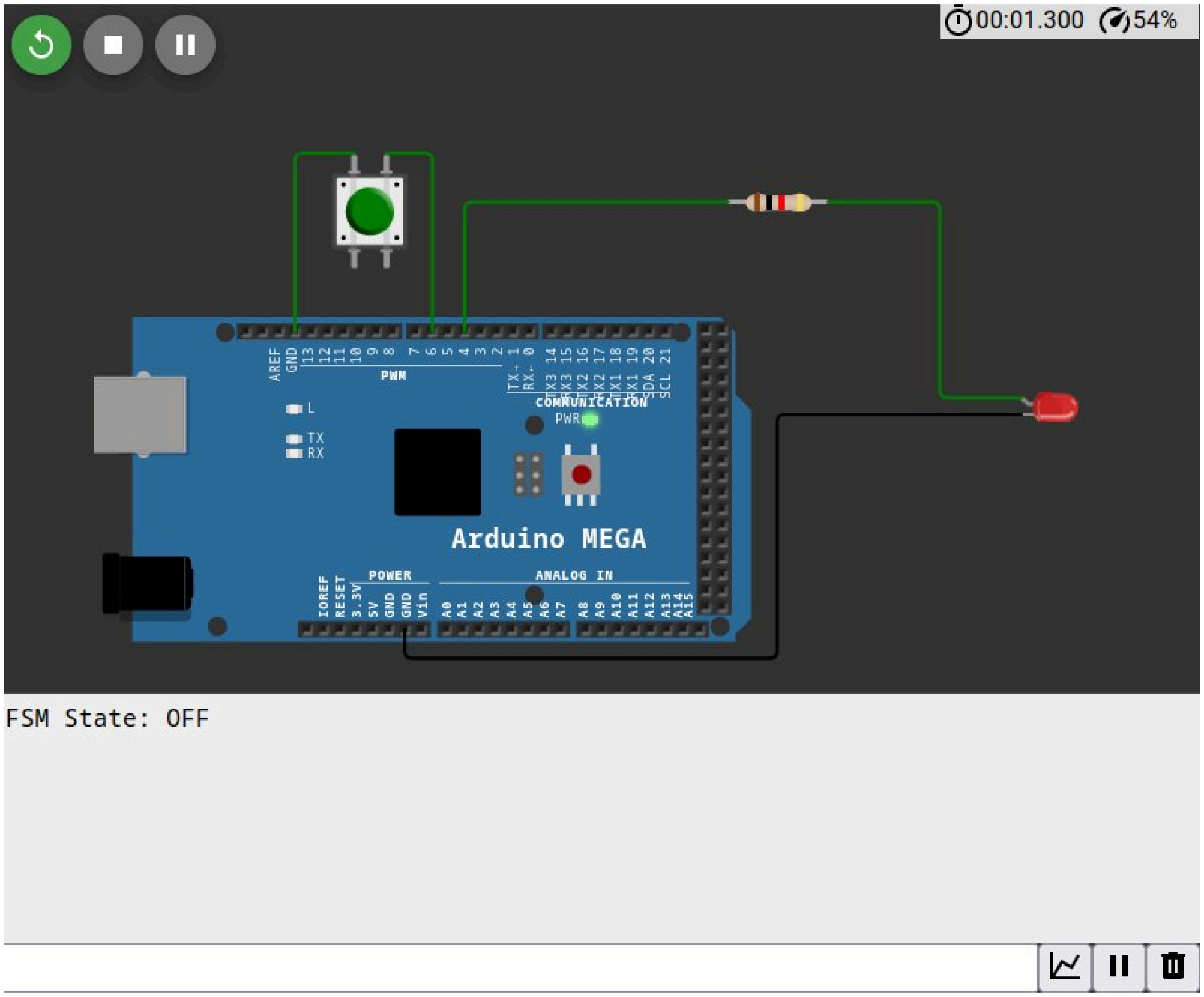


Figure 10. Button Update Function Block Diagram

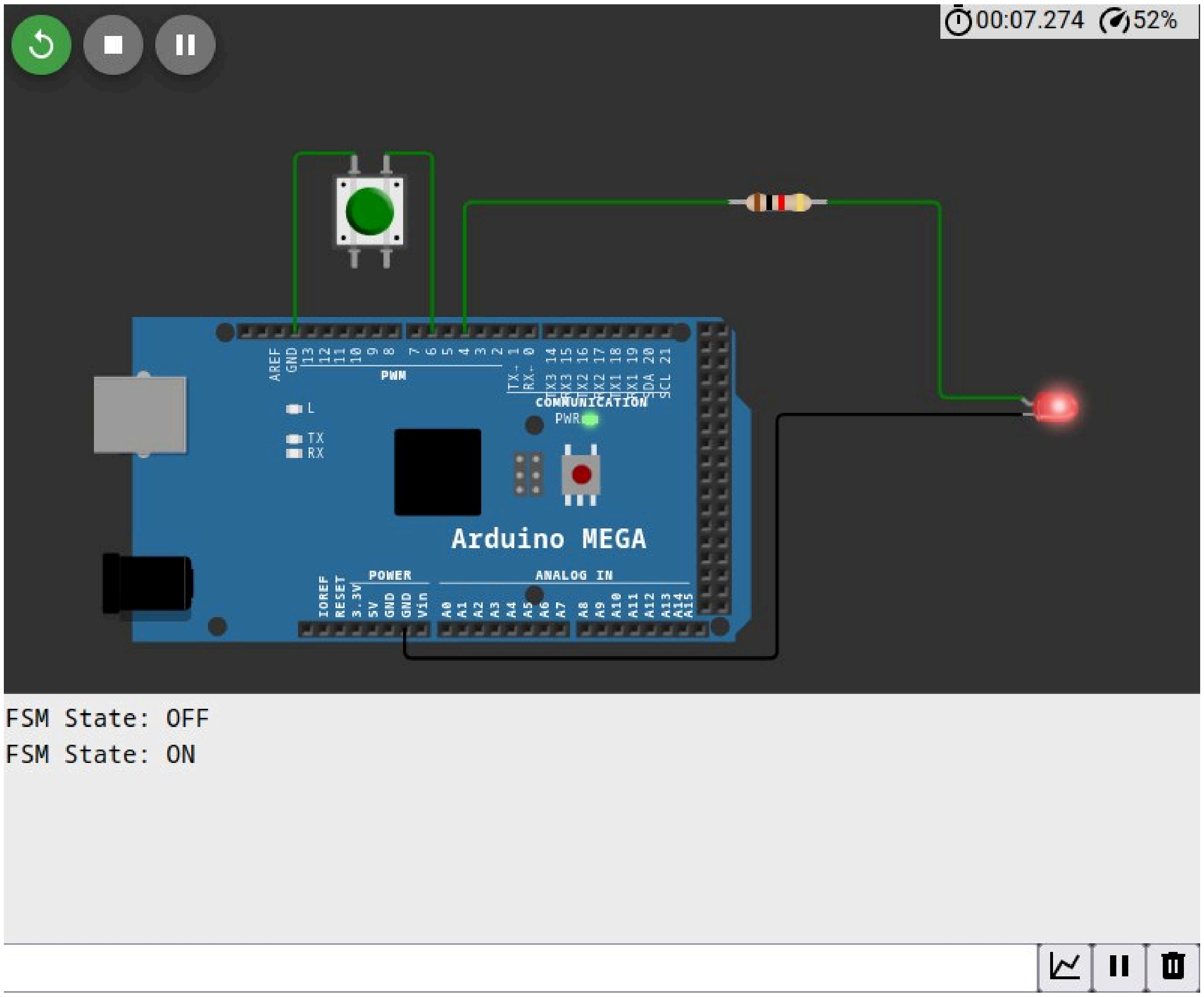
This diagram handles debouncing logic to ensure a clean press is detected without false triggers from switch noise.

RESULTS



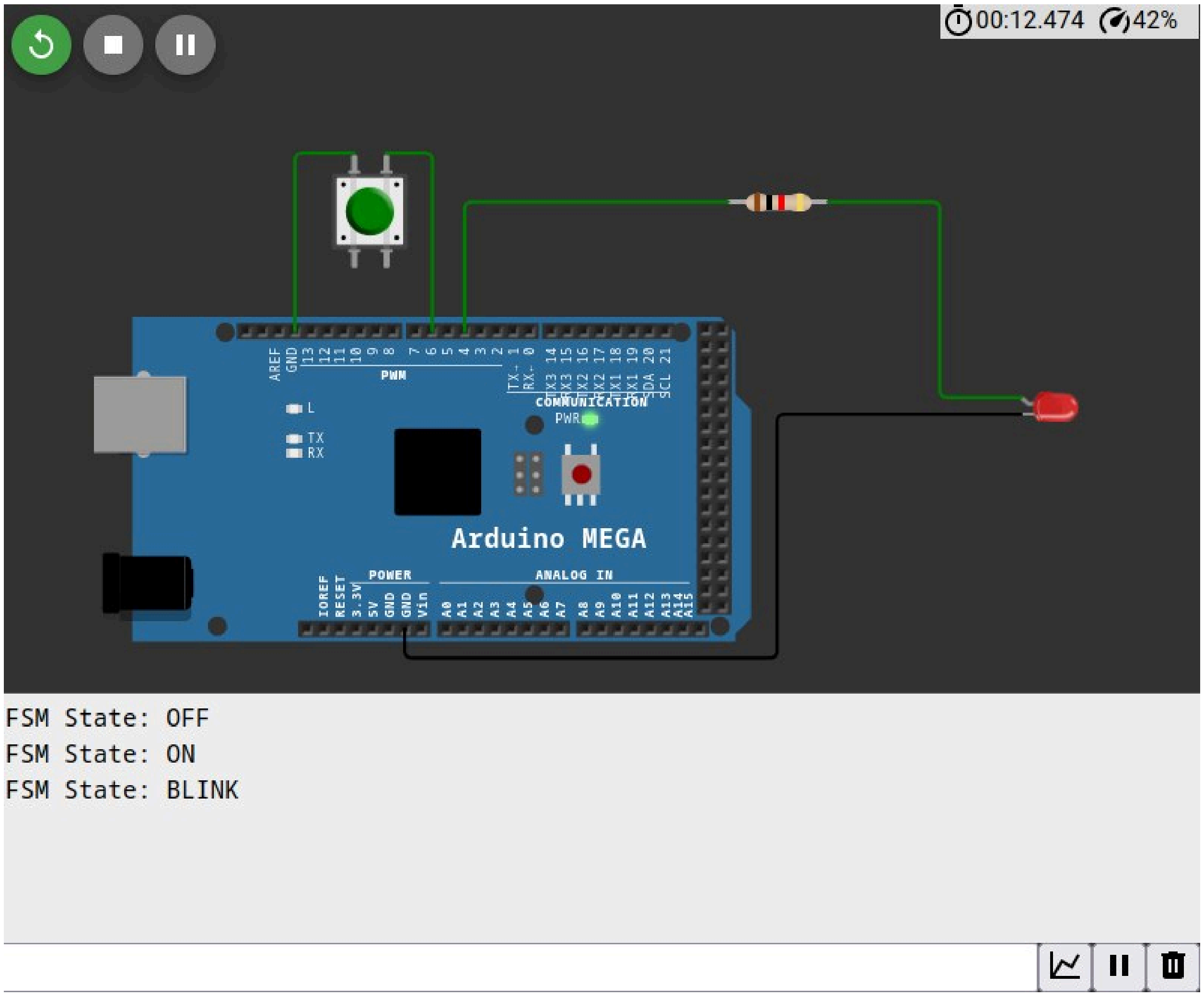
```
FSM State: OFF
```

In this image you can see the arduino mega board and the button and led connected correctly to it. The state machine begins in the OFF state, so the LED is OFF and the OFF state is printed to the Serial terminal.



```
FSM State: OFF
FSM State: ON
```

In this image the button was pressed and the state machine switched to the ON state, so the LED turned ON and the ON state is printed to the Serial terminal.

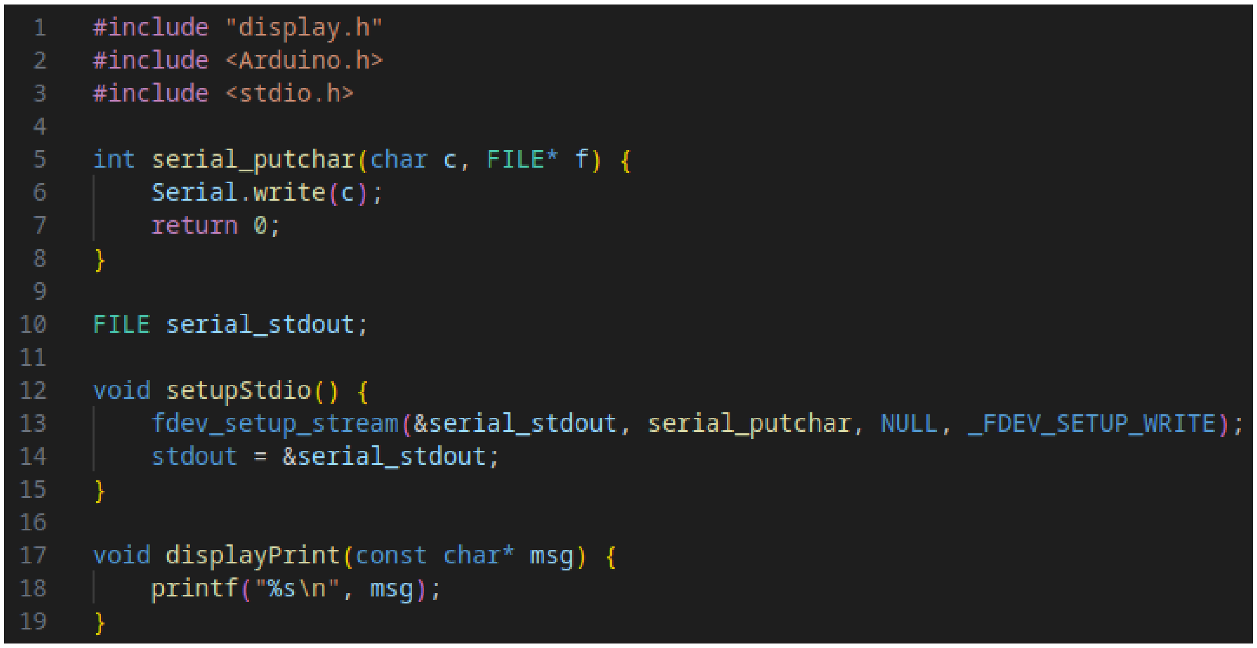


```
FSM State: OFF
FSM State: ON
FSM State: BLINK
```

In this image the button was pressed again and the state machine switched from the ON state to the blinking state, so the LED started blinking and the BLINK state is printed to the Serial terminal.

CODE IMPLEMENTATION

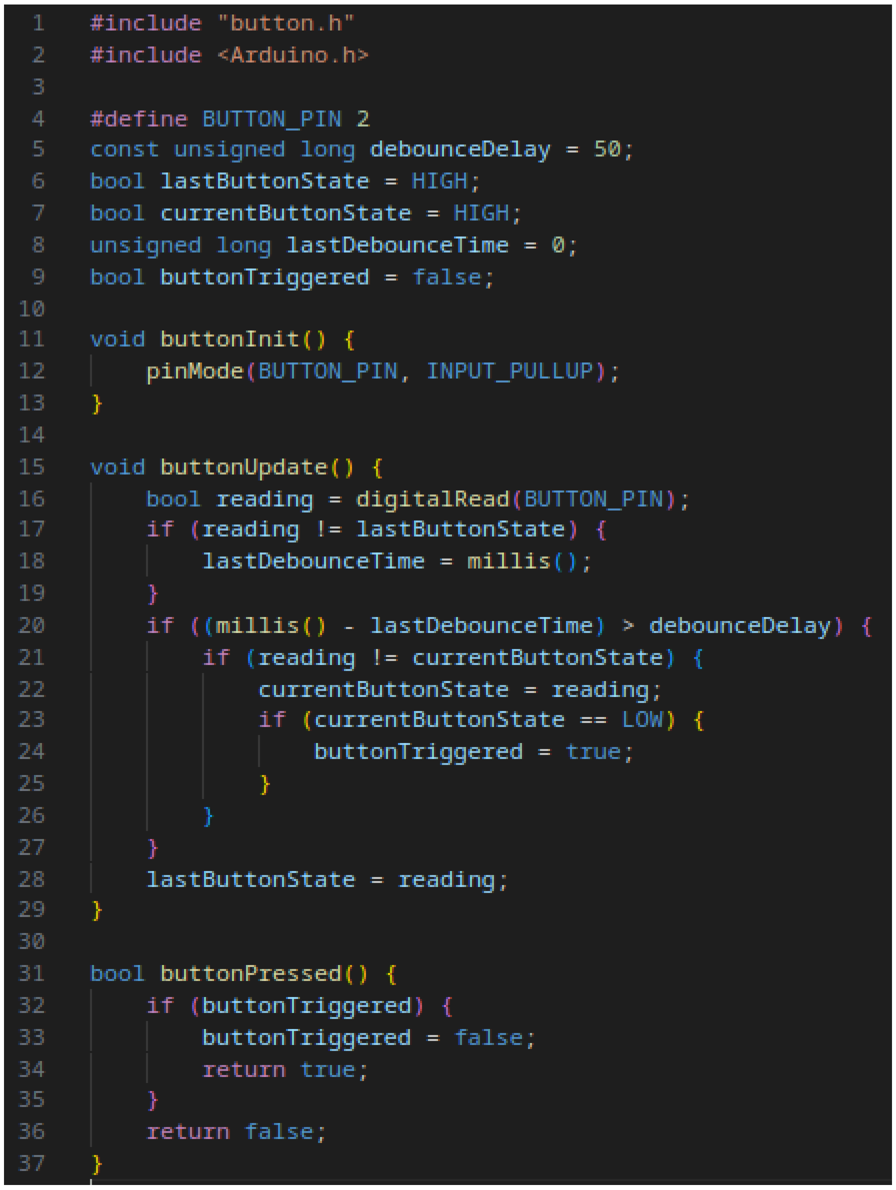
display.cpp



```
#include "display.h"
#include <Arduino.h>
#include <stdio.h>

int serial_putchar(char c, FILE* f) {
    Serial.write(c);
    return 0;
}

FILE serial_stdout;

void setupStdio() {
    fdev_setup_stream(&serial_stdout, serial_putchar, NULL, _FDEV_SETUP_WRITE);
    stdout = &serial_stdout;
}

void displayPrint(const char* msg) {
    printf("%s\n", msg);
}
```

Figure 11. display.cpp

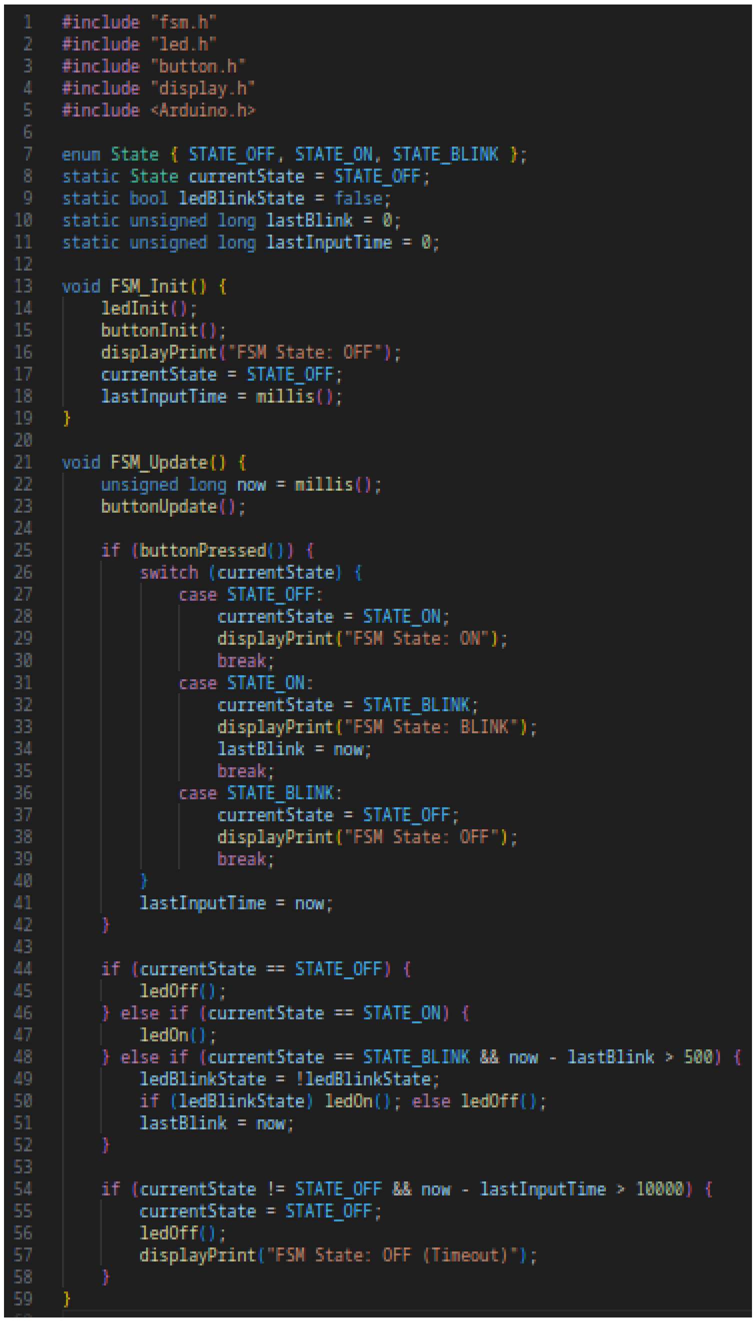
button.cpp



```cpp
#include "button.h"
#include <Arduino.h>

#define BUTTON_PIN 2
const unsigned long debounceDelay = 50;
bool lastButtonState = HIGH;
bool currentButtonState = HIGH;
unsigned long lastDebounceTime = 0;
bool buttonTriggered = false;

void buttonInit() {
    pinMode(BUTTON_PIN, INPUT_PULLUP);
}

void buttonUpdate() {
    bool reading = digitalRead(BUTTON_PIN);
    if (reading != lastButtonState) {
        lastDebounceTime = millis();
    }
    if ((millis() - lastDebounceTime) > debounceDelay) {
        if (reading != currentButtonState) {
            currentButtonState = reading;
            if (currentButtonState == LOW) {
                buttonTriggered = true;
            }
        }
    }
    lastButtonState = reading;
}

bool buttonPressed() {
    if (buttonTriggered) {
        buttonTriggered = false;
        return true;
    }
    return false;
}
```

Figure 12. *button.cpp*

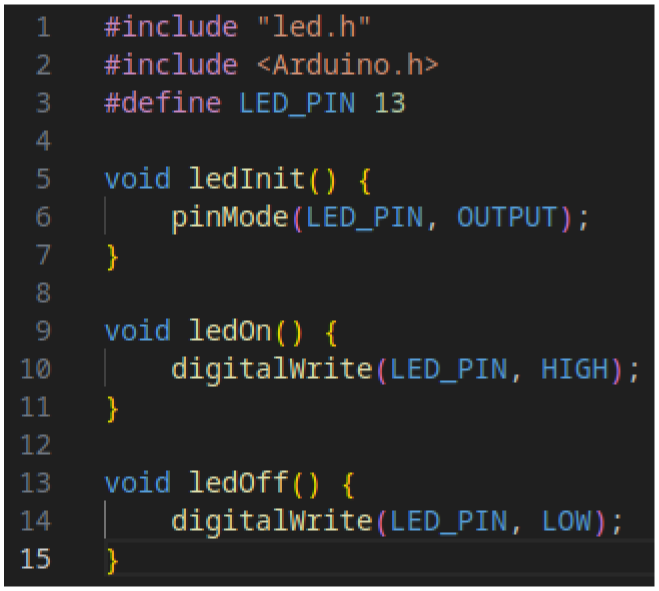
fsm.cpp



```cpp
#include "fsm.h"
#include "led.h"
#include "button.h"
#include "display.h"
#include <Arduino.h>

enum State { STATE_OFF, STATE_ON, STATE_BLINK };
static State currentState = STATE_OFF;
static bool ledBlinkState = false;
static unsigned long lastBlink = 0;
static unsigned long lastInputTime = 0;

void FSM_Init() {
    ledInit();
    buttonInit();
    displayPrint("FSM State: OFF");
    currentState = STATE_OFF;
    lastInputTime = millis();
}

void FSM_Update() {
    unsigned long now = millis();
    buttonUpdate();

    if (buttonPressed()) {
        switch (currentState) {
            case STATE_OFF:
                currentState = STATE_ON;
                displayPrint("FSM State: ON");
                break;
            case STATE_ON:
                currentState = STATE_BLINK;
                displayPrint("FSM State: BLINK");
                lastBlink = now;
                break;
            case STATE_BLINK:
                currentState = STATE_OFF;
                displayPrint("FSM State: OFF");
                break;
        }
        lastInputTime = now;
    }

    if (currentState == STATE_OFF) {
        ledOff();
    } else if (currentState == STATE_ON) {
        ledOn();
    } else if (currentState == STATE_BLINK && now - lastBlink > 500) {
        ledBlinkState = !ledBlinkState;
        if (ledBlinkState) ledOn(); else ledOff();
        lastBlink = now;
    }

    if (currentState != STATE_OFF && now - lastInputTime > 10000) {
        currentState = STATE_OFF;
        ledOff();
        displayPrint("FSM State: OFF (Timeout)");
    }
}
```

Figure 13. fsm.cpp

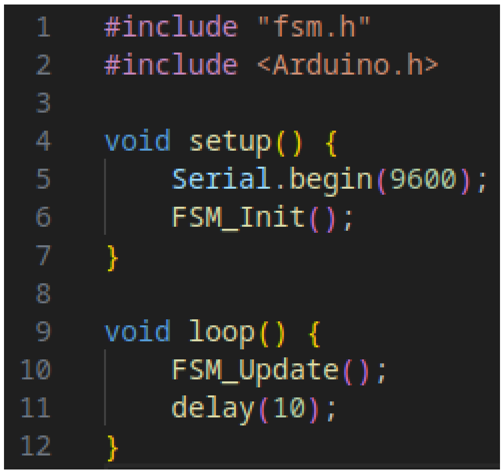
led.cpp



```cpp
#include "led.h"
#include <Arduino.h>
#define LED_PIN 13

void ledInit() {
    pinMode(LED_PIN, OUTPUT);
}

void ledOn() {
    digitalWrite(LED_PIN, HIGH);
}

void ledOff() {
    digitalWrite(LED_PIN, LOW);
}
```

Figure 14. led.cpp

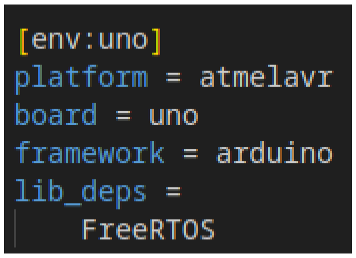
main.cpp



```cpp
#include "fsm.h"
#include <Arduino.h>

void setup() {
    Serial.begin(9600);
    FSM_Init();
}

void loop() {
    FSM_Update();
    delay(10);
}
```

Figure 15. main.cpp

PlatformIO Configuration



```
[env:uno]
platform = atmelavr
board = uno
framework = arduino
lib_deps =
    FreeRTOS
```

Figure 16. platformio.ini

CONCLUSION

This lab successfully demonstrates the use of an FSM in embedded systems to control hardware behavior through structured state transitions. The use of debouncing ensures input reliability, and the modular design simplifies testing and maintenance. The implementation of printf() output through STDIO bridges embedded hardware with familiar development practices.

BIBLIOGRAPHY

1. Arduino Uno Pinout: https://www.arduino.cc/en/Main/ArduinoBoardUno
2. Graphviz Visualization Software: https://graphviz.gitlab.io/
3. Arduino Reference - Serial Communication: https://www.arduino.cc/reference/en/#communication
4. PlatformIO Documentation: https://docs.platformio.org/en/latest/
5. Universitatea Tehnică din Moldova (UTM) - Introducere în Sistemele Embedded şi Programarea Microcontrolerelor
6. Universitatea Tehnică din Moldova (UTM) - Principiile comunicației seriale şi utilizarea interfeței UART