Laboratory Work No. 2

Sequential Systems with Task Scheduling

Gurschi Gheorghe
Group: FAF-231
Technical University of Moldova

February 2, 2026

Contents

1 Domain Analysis

1.1 Description of Technologies Used

1.1.1 Sequential Task Execution

Sequential task execution is a fundamental approach in embedded systems where multiple tasks are executed one after another in a main loop. Unlike preemptive multitasking, sequential execution does not interrupt a running task, making it simpler to implement and debug but requiring careful timing management.

Key characteristics:

- **Non-preemptive** - tasks run to completion without interruption
- **Cooperative** - tasks must voluntarily yield control
- **Deterministic** - execution order is predictable
- **Simple** - no complex scheduler or context switching required

1.1.2 Task Scheduling with Timing

Each task has a defined period (recurrence) that determines how often it should execute. The scheduler checks if enough time has passed since the last execution before running a task again. This approach reduces CPU load by preventing unnecessary executions.

Timing parameters used:

- **Task Period** - minimum time between task executions
- **Offset** - initial delay to stagger task execution
- **millis()** - Arduino function returning milliseconds since startup

1.1.3 Provider/Consumer Model

The provider/consumer model is a communication pattern where one task (provider) generates data and stores it in a global variable, while another task (consumer) reads and uses this data. This decouples tasks and allows asynchronous data sharing.

Implementation in this project:

- **Task 1 (Provider)** - provides led1State variable
- **Task 2 (Consumer/Provider)** - consumes led1State, provides blinkCount and led2State
- **Task 3 (Provider)** - provides blinkInterval variable
- **Idle Task (Consumer)** - consumes all variables for reporting

1.1.4 Input Pull-up Configuration

The Arduino's internal pull-up resistors are used for button inputs. When enabled, the pin reads HIGH when the button is not pressed and LOW when pressed. This eliminates the need for external pull-up or pull-down resistors.

1.2 Hardware Components Used

- **Arduino Mega 2560** - microcontroller based on ATmega2560
- **2x LED** - visual indicators for system state
- **2x 220Ω Resistor** - current limiting for LEDs
- **3x Tactile Button** - user input for task control
- **Breadboard** - prototyping platform
- **Jumper wires** - connections

1.3 Software Components Used

- **Arduino IDE** - development environment
- **STDIO Library** - printf for serial reporting
- **Serial Monitor** - output display for system status

1.4 System Architecture

The system implements four concurrent tasks in a sequential execution model:

1. **Task 1 - Button LED** - toggles LED1 on button press
2. **Task 2 - Blink LED** - controls LED2 blinking when LED1 is off
3. **Task 3 - Interval Control** - adjusts blink speed via buttons
4. **Idle Task - Reporting** - displays system status via STDIO

2 Design

2.1 Electrical Schematic

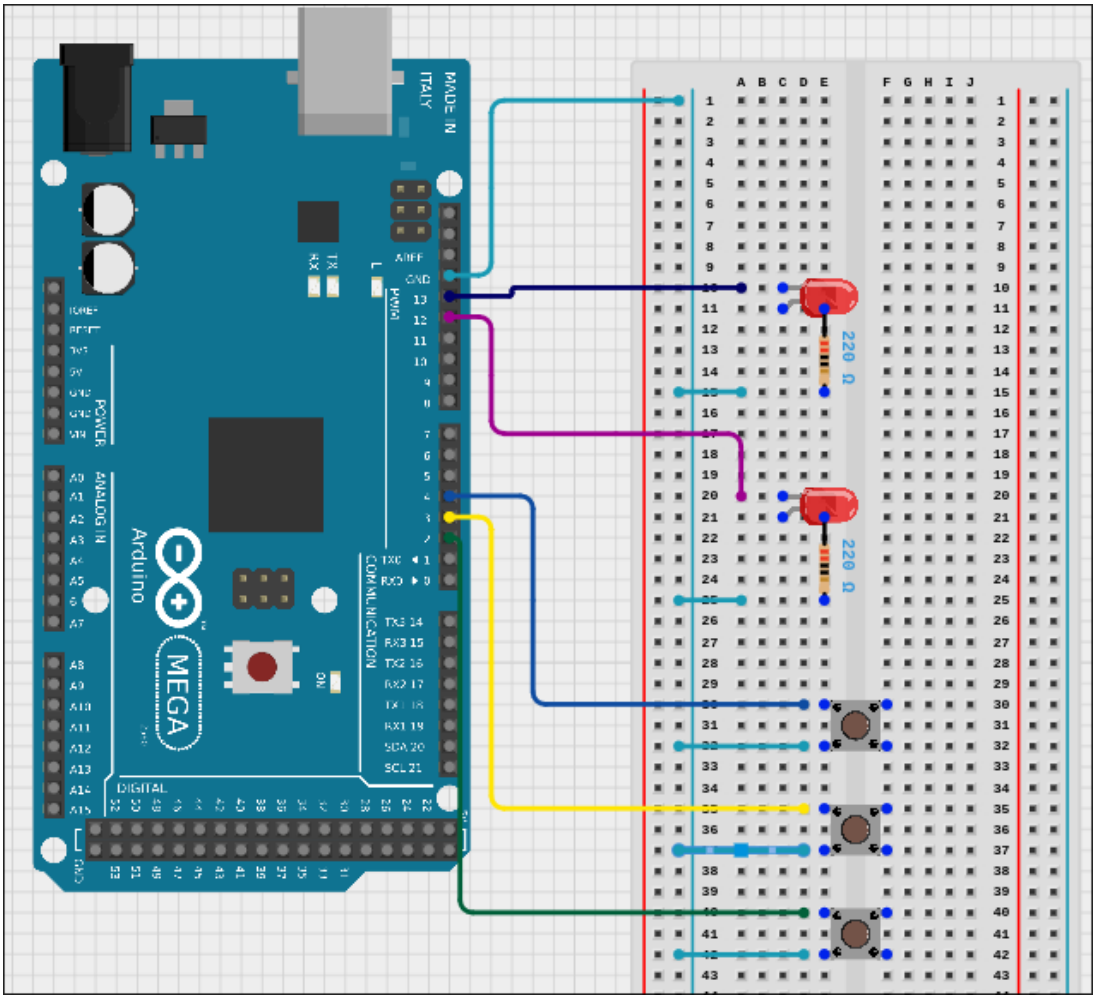

Figure 1: Circuit electrical schematic

LED Connections:

- **LED 1:** Arduino Pin 13 → LED Anode → LED Cathode → 220Ω Resistor → GND
- **LED 2:** Arduino Pin 12 → LED Anode → LED Cathode → 220Ω Resistor → GND

Button Connections (with internal pull-up):

- **Button 1:** Arduino Pin 2 → Button → GND
- **Button 2:** Arduino Pin 3 → Button → GND
- **Button 3:** Arduino Pin 4 → Button → GND

2.2 Project Structure

`Lab2_Sequential/`

```
Lab2_Sequential.ino
    Global Signals (provider/consumer variables)
    STDIO Module (serialPutchar, stdioInit)
    LED Module (ledInit, led1Toggle, led2Toggle, etc.)
    Button Module (buttonInit, buttonIsPressed functions)
    Task 1 - Button LED (task1_ButtonLED)
    Task 2 - Blink LED (task2_BlinkLED)
    Task 3 - Interval Control (task3_IntervalControl)
    Idle Task - Reporting (idleTask_Report)
```

2.3 Modular Implementation

2.3.1 Global Signals Module

Variables shared between tasks using provider/consumer model:

- `led1State` - current state of LED1 (bool)
- `led2State` - current state of LED2 (bool)
- `blinkInterval` - time between LED2 toggles (int, ms)
- `blinkCount` - number of LED2 blinks (int)

2.3.2 LED Module

Functions for LED control:

- `ledInit()` - initializes LED pins as OUTPUT
- `led1On()`, `led1Off()`, `led1Toggle()` - LED1 control
- `led2On()`, `led2Off()`, `led2Toggle()` - LED2 control

2.3.3 Button Module

Functions for button input with debouncing:

- `buttonInit()` - initializes pins with INPUT_PULLUP
- `button1IsPressed()` - returns true if button 1 pressed
- `button2IsPressed()` - returns true if button 2 pressed
- `button3IsPressed()` - returns true if button 3 pressed

2.3.4 Task Modules

Each task has its own timing and functionality:

- `task1_ButtonLED()` - period 50ms, toggles LED1
- `task2_BlinkLED()` - period 10ms, blinks LED2 when LED1 off
- `task3_IntervalControl()` - period 50ms, adjusts blink interval
- `idleTask_Report()` - period 1000ms, prints status via STDIO

3 Results Presentation

3.1 Hardware Setup

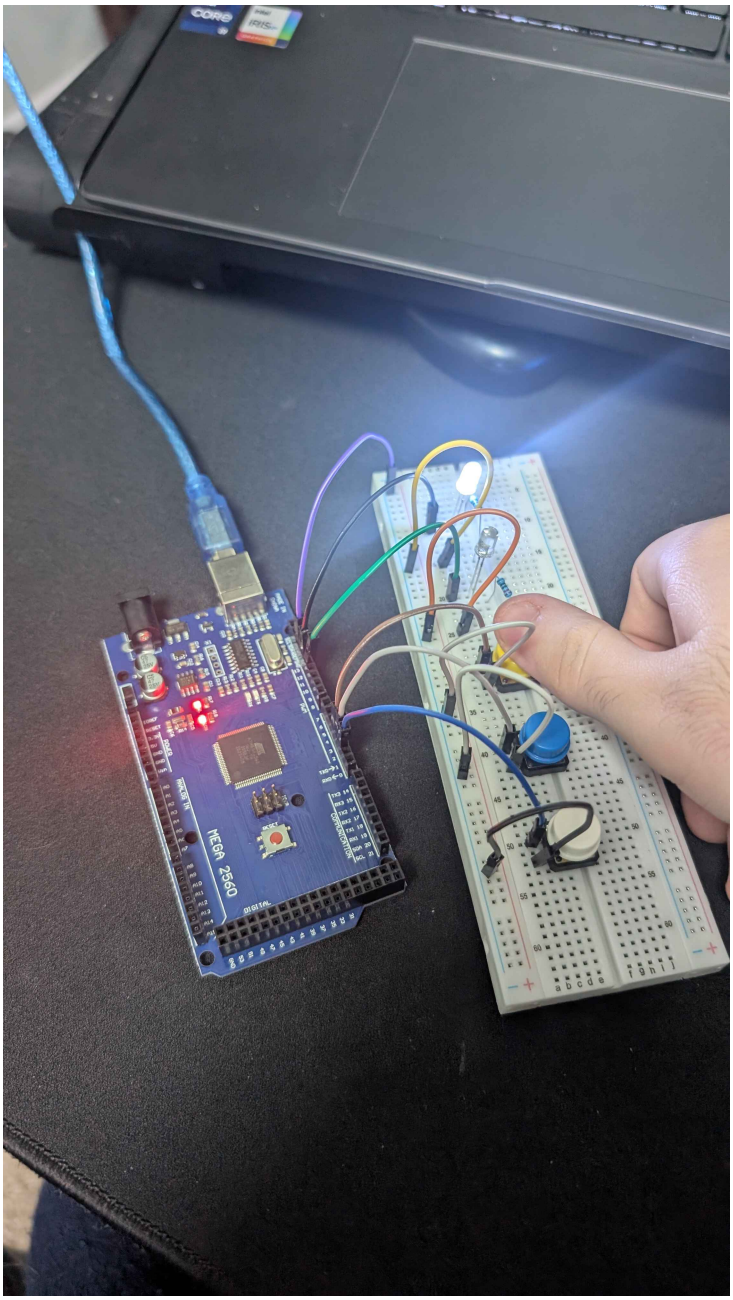

Figure 2: LED 1 ON - Initial state, LED 2 is off

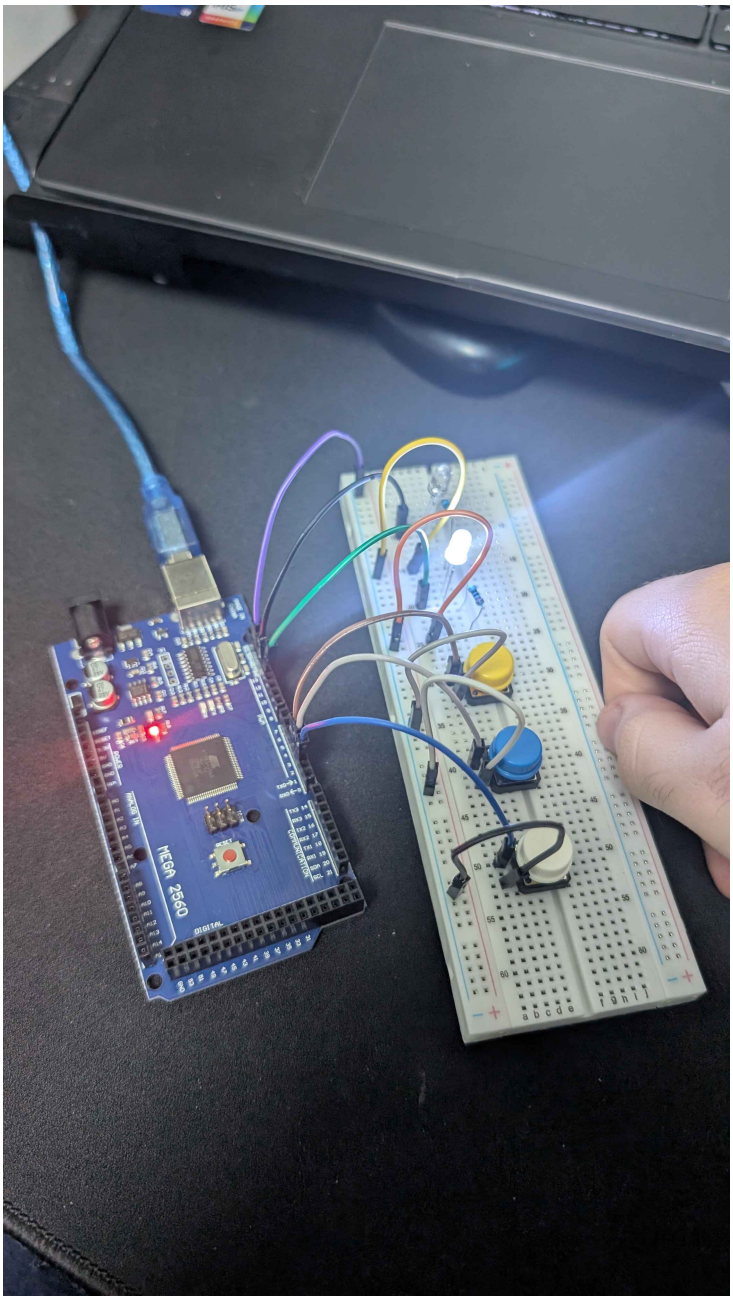

Figure 3: LED 2 ON - After pressing Button 1, LED 2 blinks

3.2 Serial Monitor Output

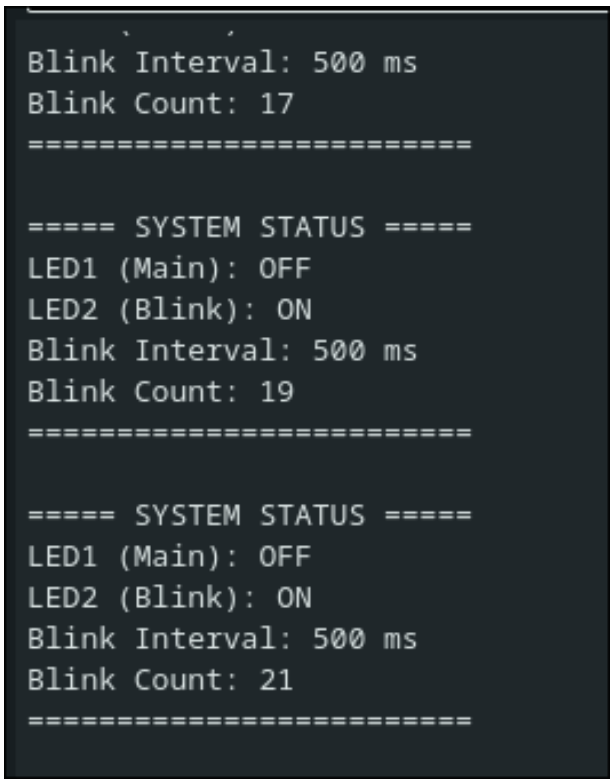

Figure 4: System status showing blink count incrementing

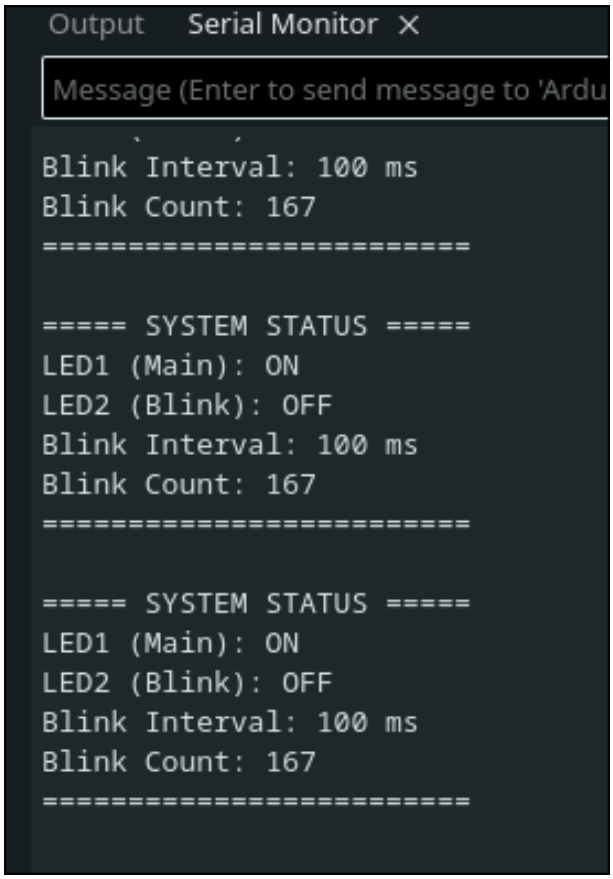

Figure 5: Blink interval increased to 100ms via Button 2

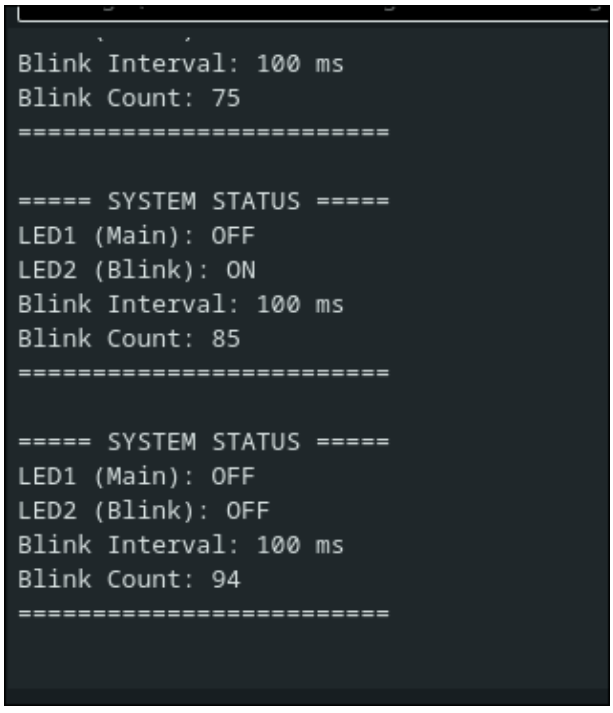

Figure 6: Blink interval at 100ms minimum after Button 3 presses

3.3 Functionality Demonstration

The system was successfully tested with all features working:

- Button 1 correctly toggles LED1 state
- LED2 blinks only when LED1 is OFF (provider/consumer working)
- Button 2 increases blink interval (max 2000ms)
- Button 3 decreases blink interval (min 100ms)
- Idle task reports system status every second via STDIO printf()
- Blink count accurately tracks LED2 state changes

4 Conclusions

The developed system successfully demonstrates sequential task execution with non-preemptive scheduling on an Arduino Mega 2560 microcontroller. The implementation achieves all laboratory objectives, including task synchronization through the provider/-consumer model, timing-based task scheduling, and system status reporting via the STDIO library.

The provider/consumer communication pattern proved effective for decoupling tasks while maintaining data consistency. Task 1 provides the LED1 state that Task 2 consumes to determine whether to blink LED2. Task 3 provides the blink interval that Task 2 uses for timing. The Idle task consumes all variables to generate comprehensive status reports.

The timing configuration with different periods for each task (50ms for button tasks, 10ms for blink control, 1000ms for reporting) optimizes CPU utilization while ensuring responsive user interaction. The use of millis() for non-blocking timing allows all tasks to share the main loop effectively.

The sequential execution model, while simpler than preemptive multitasking, requires careful design to ensure no single task blocks others for extended periods. The implemented solution demonstrates this through short, non-blocking task functions that check timing conditions before executing their main logic.

This laboratory work provides practical experience with embedded systems concepts that are fundamental to real-world applications such as industrial controllers, home automation systems, and IoT devices where multiple concurrent operations must be managed efficiently on resource-constrained hardware.

Note on AI Tools Usage

During the preparation of this report, the author used **Claude (Anthropic)** for content generation and consolidation. The resulting information was reviewed, validated, and adjusted according to laboratory requirements.

5 Bibliography

1. Arduino Reference - millis() function. `https://www.arduino.cc/reference/en/language/functions/time/millis/`

2. Arduino Reference - Digital Pins. https://www.arduino.cc/en/Tutorial/Foundations/DigitalPins

3. AVR Libc Reference Manual - Standard IO facilities. https://www.nongnu.org/avr-libc/user-manual/group__avr__stdio.html

4. Embedded Systems Design - Sequential vs Preemptive Scheduling. https://www.embedded.com/

6 Appendix - Complete Source Code

Listing 1: Lab2_Sequential.ino

```
#include <stdio.h>

//==========================================
// GLOBAL SIGNALS (Provider/Consumer)
//==========================================
volatile bool led1State = false;
volatile int blinkInterval = 500;
volatile int blinkCount = 0;
volatile bool led2State = false;

//==========================================
// PIN DEFINITIONS
//==========================================
#define LED1_PIN 13
#define LED2_PIN 12
#define BUTTON1_PIN 2
#define BUTTON2_PIN 3
#define BUTTON3_PIN 4

//==========================================
// TIMING CONFIGURATION
//==========================================
#define TASK1_PERIOD 50
#define TASK2_PERIOD 10
#define TASK3_PERIOD 50
#define IDLE_PERIOD 1000

unsigned long task1LastRun = 0;
unsigned long task2LastRun = 0;
unsigned long task3LastRun = 0;
unsigned long idleLastRun = 0;
unsigned long led2LastToggle = 0;

//==========================================
// STDIO MODULE
//==========================================
int serialPutchar(char c, FILE *stream) {
    Serial.write(c);
```

```
39      return c;
40  }
41
42  FILE serialStream;
43
44  void stdioInit(void) {
45      fdev_setup_stream(&serialStream, serialPutchar, NULL,
            _FDEV_SETUP_WRITE);
46      stdout = &serialStream;
47  }
48
49  //============================================
50  // LED MODULE
51  //============================================
52  void ledInit(void) {
53      pinMode(LED1_PIN, OUTPUT);
54      pinMode(LED2_PIN, OUTPUT);
55      digitalWrite(LED1_PIN, LOW);
56      digitalWrite(LED2_PIN, LOW);
57  }
58
59  void led1On(void) {
60      digitalWrite(LED1_PIN, HIGH);
61      led1State = true;
62  }
63
64  void led1Off(void) {
65      digitalWrite(LED1_PIN, LOW);
66      led1State = false;
67  }
68
69  void led1Toggle(void) {
70      if (led1State) {
71          led1Off();
72      } else {
73          led1On();
74      }
75  }
76
77  void led2On(void) {
78      digitalWrite(LED2_PIN, HIGH);
79      led2State = true;
80  }
81
82  void led2Off(void) {
83      digitalWrite(LED2_PIN, LOW);
84      led2State = false;
85  }
86
87  void led2Toggle(void) {
88      if (led2State) {
```

```
89          led2Off();
90      } else {
91          led2On();
92      }
93      blinkCount++;
94  }
95
96  //============================================
97  // BUTTON MODULE
98  //============================================
99  bool button1Pressed = false;
100 bool button2Pressed = false;
101 bool button3Pressed = false;
102
103 void buttonInit(void) {
104     pinMode(BUTTON1_PIN, INPUT_PULLUP);
105     pinMode(BUTTON2_PIN, INPUT_PULLUP);
106     pinMode(BUTTON3_PIN, INPUT_PULLUP);
107 }
108
109 bool button1IsPressed(void) {
110     return digitalRead(BUTTON1_PIN) == LOW;
111 }
112
113 bool button2IsPressed(void) {
114     return digitalRead(BUTTON2_PIN) == LOW;
115 }
116
117 bool button3IsPressed(void) {
118     return digitalRead(BUTTON3_PIN) == LOW;
119 }
120
121 //============================================
122 // TASK 1: Button LED Toggle
123 //============================================
124 void task1_ButtonLED(void) {
125     unsigned long currentTime = millis();
126
127     if (currentTime - task1LastRun >= TASK1_PERIOD) {
128         task1LastRun = currentTime;
129
130         bool currentState = button1IsPressed();
131
132         if (currentState && !button1Pressed) {
133             led1Toggle();
134         }
135
136         button1Pressed = currentState;
137     }
138 }
139
```

```
//==========================================
// TASK 2: Blinking LED
//==========================================
void task2_BlinkLED(void) {
    unsigned long currentTime = millis();

    if (currentTime - task2LastRun >= TASK2_PERIOD) {
        task2LastRun = currentTime;

        if (!led1State) {
            if (currentTime - led2LastToggle >= blinkInterval) {
                led2LastToggle = currentTime;
                led2Toggle();
            }
        } else {
            if (led2State) {
                led2Off();
            }
        }
    }
}

//==========================================
// TASK 3: Blink Interval Control
//==========================================
void task3_IntervalControl(void) {
    unsigned long currentTime = millis();

    if (currentTime - task3LastRun >= TASK3_PERIOD) {
        task3LastRun = currentTime;

        bool btn2State = button2IsPressed();
        bool btn3State = button3IsPressed();

        if (btn2State && !button2Pressed) {
            blinkInterval += 100;
            if (blinkInterval > 2000) {
                blinkInterval = 2000;
            }
        }

        if (btn3State && !button3Pressed) {
            blinkInterval -= 100;
            if (blinkInterval < 100) {
                blinkInterval = 100;
            }
        }

        button2Pressed = btn2State;
        button3Pressed = btn3State;
    }
```

```
}

//==========================================
// IDLE TASK: Reporting via STDIO
//==========================================
void idleTask_Report(void) {
    unsigned long currentTime = millis();

    if (currentTime - idleLastRun >= IDLE_PERIOD) {
        idleLastRun = currentTime;

        printf("=====␣SYSTEM␣STATUS␣=====\n");
        printf("LED1␣(Main):␣%s\n", led1State ? "ON" : "OFF");
        printf("LED2␣(Blink):␣%s\n", led2State ? "ON" : "OFF");
        printf("Blink␣Interval:␣%d␣ms\n", blinkInterval);
        printf("Blink␣Count:␣%d\n", blinkCount);
        printf("==========================\n\n");
    }
}

//==========================================
// SETUP
//==========================================
void setup() {
    Serial.begin(9600);
    stdioInit();
    ledInit();
    buttonInit();

    printf("Sequential␣Task␣System␣Started\n");
    printf("Button␣1:␣Toggle␣LED1\n");
    printf("Button␣2:␣Increase␣blink␣interval\n");
    printf("Button␣3:␣Decrease␣blink␣interval\n\n");
}

//==========================================
// MAIN LOOP - Sequential Execution
//==========================================
void loop() {
    task1_ButtonLED();
    task2_BlinkLED();
    task3_IntervalControl();
    idleTask_Report();
}
```