MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF THE REPUBLIC OF MOLDOVA

Technical University of Moldova Faculty of Computers, Informatics and Microelectronics Department of Software and Automation Engineering

Postoronca Dumitru FAF-233

Report

Laboratory work n.3

on Embedded Systems

Checked by:
A. Martiniuc, *university assistant*
DISA, FCIM, UTM

Chișinău – 2026

1 Purpose of laboratory work

Purpose of the laboratory work is to design and implement a multitasking application for a microcontroller that monitors the duration of each button press, signals the press duration visually through LEDs, and periodically reports statistics to the user via STDIO.

1.1 Objectives of the laboratory work

Laboratory work has the following objectives that should be achieved:

- Understanding the principles of non-preemptive (bare-metal) multitasking on microcontrollers
- Implementing a cooperative scheduler with context structures, recurrence, and offset parameters
- Designing an application structured into distinct tasks with clear responsibilities
- Measuring button press durations and classifying them as short (< 500 ms) or long (≥ 500 ms)
- Providing visual feedback through LEDs and periodic statistical reports through STDIO

1.2 Problem definition

The application must be structured in at least 3 distinct tasks:

1. **Task 1 – Button detection and press duration measurement:** Monitors the button state, detects pressed/released transitions, and measures the duration of each press in milliseconds. On each valid press, it saves the duration in a global variable and signals visually:
   - Green LED on for a short press (< 500 ms)
   - Red LED on for a long press (≥ 500 ms)
2. **Task 2 – Press counting and statistics:** On each detected press, increments a global press counter, updates the total duration sums for short and long presses, and tracks the count of short and long presses. Performs a rapid blink of the yellow LED: 5 blinks for a short press and 10 blinks for a long press.

3. **Task 3 – Periodic reporting:** Every 10 seconds, transmits to the user via STDIO: total number of presses, number of short presses (< 500 ms), number of long presses (≥ 500 ms), average press duration, and then resets the statistics.

2 Domain Analysis

2.1 Application Context and Utilized Technologies

The primary objective of this laboratory work is to implement a **non-preemptive cooperative multitasking system** on a bare-metal microcontroller. This approach is fundamental in embedded engineering for building responsive systems without the overhead of a full real-time operating system (RTOS).

The application uses a **tick-based cooperative scheduler** that manages multiple tasks with individual recurrence intervals and offsets. Each task is represented by a context structure containing a function pointer, recurrence period, offset, and a countdown counter. A hardware timer generates a 1 KHz system tick (1 ms resolution), and the scheduler executes at most one task per tick, ensuring deterministic and non-preemptive behavior.

Communication with the user is achieved through the standard I/O library (`stdio.h`) adapted for the microcontroller's UART interface, allowing the use of `printf()` for outputting periodic statistical reports.

2.2 Hardware and Software Components

Component	Description & Role
LAFVIN R3 (Arduino UNO R3)	Microcontroller board with ATmega328P that runs the cooperative scheduler and all application tasks.
Push Button	Input peripheral connected to pin 2 with internal pull-up resistor. Used to detect press/release transitions for duration measurement.
Green LED	Connected to pin 13 (LED_BUILTIN). Lights up to indicate a short button press (< 500 ms).
Red LED	Connected to pin 12. Lights up to indicate a long button press (≥ 500 ms).
Yellow LED	Connected to pin 11. Performs rapid blinking to acknowledge each press (5 blinks for short, 10 for long).
$220\,\Omega$ **Resistors**	Current limiting resistors for each LED to keep current within safe operating limits.
Serial-to-USB Interface	Facilitates UART communication between the MCU and the terminal for periodic report output.
Hardware Timer (Timer1)	Configured at 1 KHz to generate the system tick interrupt that drives the cooperative scheduler.
Neovim + Arduino-nvim plugin	Development environment with `arduino-cli` integration for compilation, upload, and monitoring.
Breadboard	Prototyping base for interconnecting the button, LEDs, resistors, and the Arduino without soldering.

2.3 System Architecture and Solution Justification

The system adopts a **cooperative (non-preemptive) scheduler architecture**. This design was chosen because it provides deterministic task execution without the complexity of context switching found in preemptive systems. Each task runs to completion before yielding control back to the scheduler, which simplifies shared variable access and eliminates the need for mutexes or semaphores.

The scheduler uses an array of `TaskContext` structures, each containing:

- `func` – pointer to the task function
- `recurrence` – how often the task should execute (in ticks)
- `offset` – initial delay before first execution

- `counter` – countdown to next execution

The data flow is structured as follows:

1. **Timer ISR:** Increments the global system tick and calls `srvSchedulerTick()`.
2. **Scheduler:** Iterates through the task array, decrementing counters. The first task whose counter reaches zero is executed, and only one task runs per tick.
3. **Task Communication:** Tasks communicate through global variables (e.g., `newPressDetected`, `lastPressDuration`, `reportReady`).
4. **Report Output:** The main loop polls a `reportReady` flag and prints statistics via `printf()` outside the ISR context.

2.4 Relevant Case Study

A real-world application of cooperative multitasking is found in **industrial sensor monitoring systems**, where multiple sensors must be polled at different rates without the overhead of a full RTOS. For example, in a factory conveyor system, one task may poll proximity sensors every 10 ms, another task updates a status display every 100 ms, and a third task logs aggregated data every 10 seconds. The cooperative scheduler ensures predictable timing while keeping the firmware simple and deterministic – properties critical for safety-relevant embedded systems.

3 Design

3.1 Architectural sketch

Below is the architectural sketch of the components structured in the diagram that highlights the software and hardware components.

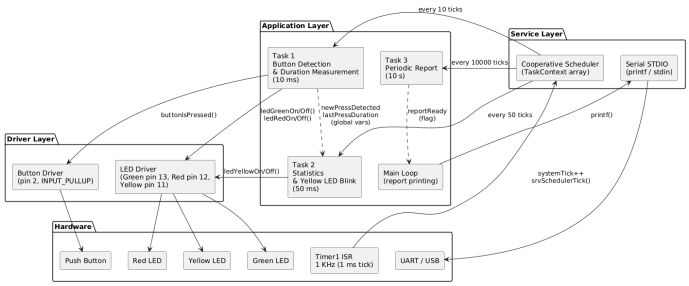

Figure 1: Architectural sketch of the project

This is the description of each of the elements of the diagram:

1. **Arduino Uno microcontroller** – central processing unit board that runs the cooperative scheduler and executes all application tasks
2. **Button (pin 2)** – input peripheral with internal pull-up resistor, used by Task 1 to detect press/release transitions
3. **Green LED (pin 13)** – output indicator for short button presses (< 500 ms)
4. **Red LED (pin 12)** – output indicator for long button presses (≥ 500 ms)
5. **Yellow LED (pin 11)** – output indicator that blinks rapidly to acknowledge each press
6. **Scheduler module** – software service that manages task execution using context structures with recurrence, offset, and countdown counter
7. **Serial STDIO module** – software driver that redirects `printf()` output to the UART serial interface for periodic report transmission
8. **LED driver module** – software driver providing abstractions for controlling each LED independently
9. **Button driver module** – software driver providing debounced button state reading

3.2 Electrical sketch

The circuit diagram outlines the setup for this project, showing the connections between the Arduino, three LEDs (green, red, yellow), a push button, and the computer for serial communication. Each LED is connected to its respective Arduino pin through a 220 Ω current-limiting resistor. The push button is connected to pin 2 with the internal pull-up resistor enabled, meaning the pin reads LOW when the button is pressed.

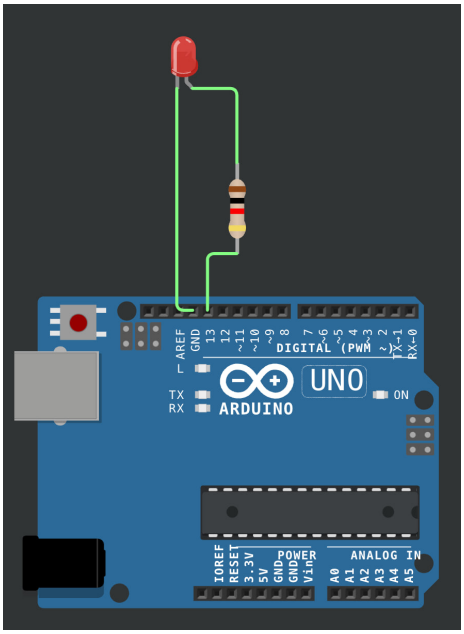

Figure 2: Circuit sketched on wokwi

Figure 2 details the connections and components that make up the system. This visual documentation is crucial for replicating the setup or diagnosing issues, as it clearly shows how each component is integrated into the overall design.

3.3 Flow chart

The Figure 3 illustrates the cooperative scheduling flow. The hardware timer generates a 1 ms tick interrupt. On each tick, the scheduler iterates through the task array and executes the first task whose counter has reached zero. Task 1 polls the button every 10 ms, Task 2 handles yellow LED blinking every 50 ms, and Task 3 triggers a

report every 10 seconds. The main loop handles printing the report outside the ISR context.

3.4 Code design

Code is developed following the principle of separating the interface from the implementation, by creating for each header file `.h` a respective `.cpp` file that implements it. The project is organized into three layers:

1. **Driver layer** – contains hardware-specific code for the button (`button.h/cpp`), LEDs (`led.h/cpp`), and serial communication (`serial.h/cpp`). All direct `Arduino.h` calls are encapsulated here.
2. **Service layer** – contains the cooperative scheduler (`scheduler.h/cpp`) which is a reusable, hardware-independent module that manages task contexts and execution timing.
3. **Application layer** – contains the task logic (`lab2-1.h/cpp`) which defines the three tasks, their recurrence/offset parameters, global shared variables, and the timer ISR that drives the system tick.

The scheduler uses a `TaskContext` structure array:

```
typedef struct {
  TaskFunc func;     // pointer to task function
  int recurrence;    // execution period in ticks
  int offset;        // initial delay
  int counter;       // countdown to next execution
} TaskContext;
```

Tasks communicate through global variables. Task 1 sets `newPressDetected` and `lastPressDuration` which Task 2 reads. Task 3 snapshots and resets the statistics, setting `reportReady` which the main loop polls to print the report via `printf()`. This design ensures that `printf()` is never called from within the ISR, avoiding potential issues with blocking I/O in interrupt context.

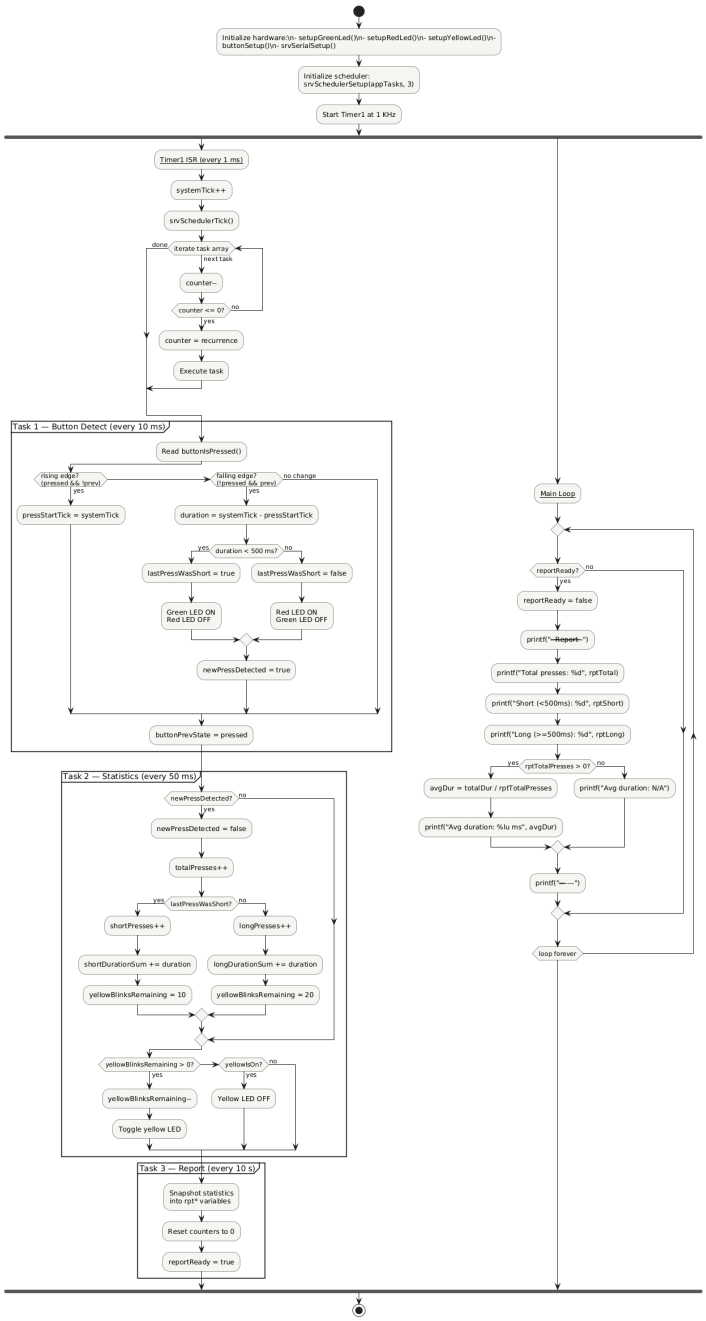

Figure 3: Flow chart of the algorithm

3.5 Results

For running the application following commands were used for compiling, uploading the project and monitoring the port to observe the periodic reports.

```
# to find to which port number the board is connected
arduino-cli board list
# compilation of the programm by indicating the target hardware
arduino-cli compile --fqbn arduino:avr:uno sketch
# uploading the program executables to the board
arduino-cli upload -p /dev/cu.usbserial-portnumber --fqbn arduino:avr:uno sketch
# monitoring the port to which the active board is connected
arduino-cli monitor -p /dev/cu.usbserial-portnumber
```

Below is the output of the commands listed above

```
$ arduino-cli compile --fqbn arduino:avr:uno sketch
Sketch uses 3918 bytes (12%) of program storage space. Maximum is 32256 bytes.
Global variables use 400 bytes (19%) of dynamic memory, leaving 1648 bytes
for local variables. Maximum is 2048 bytes.

$ arduino-cli upload -p /dev/cu.usbserial-1120 --fqbn arduino:avr:uno sketch
New upload port: /dev/cu.usbserial-1120 (serial)

$ arduino-cli monitor -p /dev/cu.usbserial-1120
Using generic monitor configuration.

Monitor port settings:
  baudrate=9600

Connecting to /dev/cu.usbserial-1120. Press CTRL-C to exit.

Lab 3.1 - Button Press Monitor
Reports every 10 seconds
--- Report ---
Total presses: 5
Short (<500ms): 3
Long  (>=500ms): 2
Avg duration: 487 ms
```

```
--------------
--- Report ---
Total presses: 0
Short (<500ms): 0
Long  (>=500ms): 0
Avg duration: N/A
--------------
```

The system behavior is as follows:

- When a short press (< 500 ms) is detected, the **green LED** turns on and the red LED turns off. The **yellow LED** blinks 5 times rapidly.
- When a long press (≥ 500 ms) is detected, the **red LED** turns on and the green LED turns off. The **yellow LED** blinks 10 times rapidly.
- Every 10 seconds, a statistics report is printed to the serial monitor showing the total number of presses, the count of short and long presses, and the average press duration. After printing, all counters are reset.
- When no presses occur during a reporting interval, the report shows zeros and "N/A" for the average duration.

Below there is the visual representation of the circuit
Figure 4 shows the circuit in its idle state with all LEDs off.

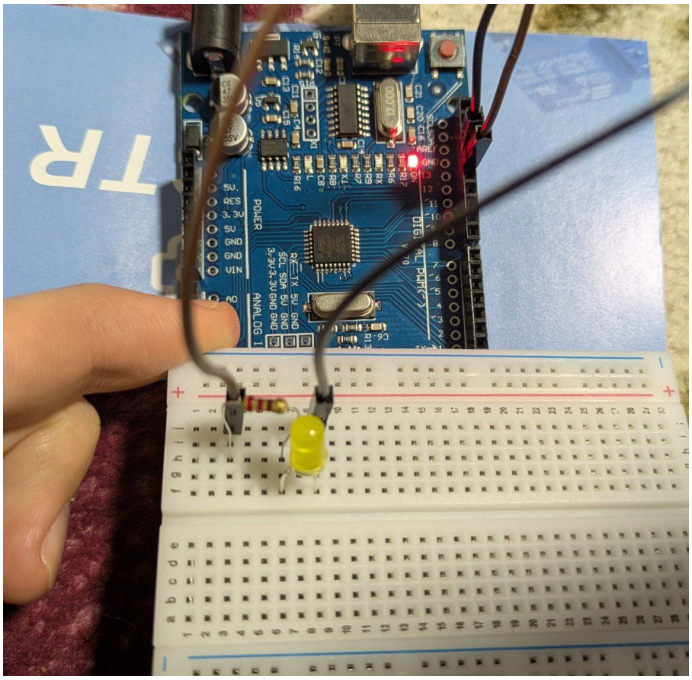

Figure 4: Circuit in idle state

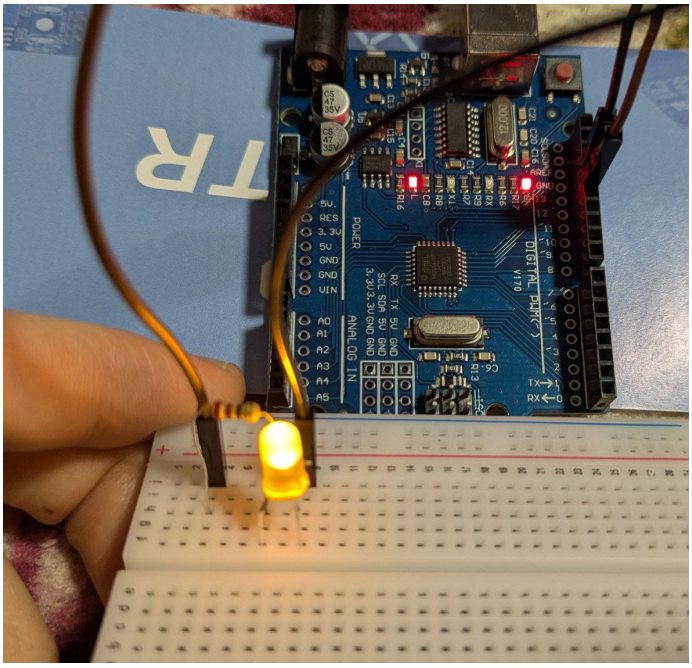

Figure 5: LED active after a button press

Figure 5 shows the circuit after a button press has been detected, with the corresponding LED (green or red) active to indicate the press duration category.

4 Conclusion

Completion of this laboratory work has provided insightful understanding of non-preemptive cooperative multitasking on bare-metal microcontroller systems and the design principles behind real-time task scheduling.

This third laboratory work covered the implementation of a cooperative scheduler with multiple tasks running at different recurrence intervals, communicating through shared global variables. The key takeaways are:

1. Designing a **cooperative scheduler** using context structures with recurrence, offset, and countdown counters – ensuring only one task executes per system tick for deterministic behavior.

2. Implementing **inter-task communication** through global variables and flags (`newPressDetected`, `reportReady`), demonstrating how tasks can exchange data without mutexes in a non-preemptive environment.

3. Structuring the application into **three distinct layers**: driver, service (scheduler), and application – maintaining portability and modularity.

4. Understanding the importance of **separating ISR-context work from main-loop work**: the timer ISR drives the scheduler, while blocking operations like `printf()` are deferred to the main loop via a flag.

5. Gaining practical experience with **button debouncing, press duration measurement**, and visual signaling through multiple LEDs at different rates.

This laboratory work serves as a foundation for understanding operating system concepts on embedded systems, bridging the gap between bare-metal sequential programming and full RTOS-based multitasking.

References

[1] Arduino. (2023). *Arduino Uno Rev3 Hardware Documentation*. Retrieved from `https://docs.arduino.cc/hardware/uno-rev3`

[2] Wunsch, J., & Weddington, E. (2022). *avr-libc: Standard IO facilities (<stdio.h>)*. Online documentation. Retrieved from `https://www.nongnu.org/avr-libc/user-manual/group__avr__stdio.html`

[3] Margolis, M. (2011). *Arduino Cookbook: Recipes to Begin, Expand, and Enhance Your Projects*. O'Reilly Media, Inc.

[4] Atmel Corporation. (2015). *ATmega328P Datasheet – 16-bit Timer/Counter1*. Retrieved from `https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf`

[5] Barr, M. (2006). *Programming Embedded Systems: With C and GNU Development Tools*. O'Reilly Media, Inc.

A Source Code

A.1 Main Sketch File

```
#include "src/lab2-1.h"

void setup() {
  lab21Setup();
}

void loop() {
  lab21Loop();
}
```

Listing 1: sketch.ino - Main Arduino Sketch

A.2 Application Module

```
#ifndef LAB21_H
#define LAB21_H

void lab21Setup();

#define SRV_OS_SYS_TICK 1  // 1 ms (1 KHr)
#define TIME_SEC 1000
#define SHORT_PRESS_THRESHOLD 500

#define TASK1_REC 10
#define TASK1_OFF 1
#define TASK2_REC 50
#define TASK2_OFF 5
#define TASK3_REC (10 * TIME_SEC)
#define TASK3_OFF (10 * TIME_SEC)

void task1ButtonDetect();
void task2Statistics();
void task3Report();
void lab21Loop();

#define NUM_TASKS 3

#endif
```

Listing 2: lab2-1.h - Application Header

```
#include "lab2-1.h"
#include "drivers/button/button.h"
#include "drivers/led/led.h"
#include "services/scheduler/scheduler.h"
#include "services/serial/serial.h"
#include "timer-api.h"
#include "timer_setup.h"

#include <stdio.h>

TaskContext appTasks[NUM_TASKS] = {{task1ButtonDetect, TASK1_REC, TASK1_OFF, 0},
                                   {task2Statistics, TASK2_REC, TASK2_OFF, 0},
                                   {task3Report, TASK3_REC, TASK3_OFF, 0}};

unsigned long systemTick = 0;

unsigned long lastPressDuration = 0;
bool newPressDetected = false;
bool lastPressWasShort = false;

bool buttonPrevState = false;
unsigned long pressStartTick = 0;

int totalPresses = 0;
int shortPresses = 0;
int longPresses = 0;
unsigned long shortDurationSum = 0;
unsigned long longDurationSum = 0;

int yellowBlinksRemaining = 0;
bool yellowIsOn = false;

bool reportReady = false;

int rptTotalPresses = 0;
int rptShortPresses = 0;
int rptLongPresses = 0;
unsigned long rptShortDurationSum = 0;
unsigned long rptLongDurationSum = 0;

void task1ButtonDetect() {
  bool pressed = buttonIsPressed();
```

```
43
44    if (pressed && !buttonPrevState) {
45      pressStartTick = systemTick;
46    } else if (!pressed && buttonPrevState) {
47      lastPressDuration = systemTick - pressStartTick;
48
49      if (lastPressDuration < SHORT_PRESS_THRESHOLD) {
50        lastPressWasShort = true;
51        ledGreenOn();
52        ledRedOff();
53      } else {
54        lastPressWasShort = false;
55        ledRedOn();
56        ledGreenOff();
57      }
58
59      newPressDetected = true;
60    }
61
62    buttonPrevState = pressed;
63  }
64
65  void task2Statistics() {
66    if (newPressDetected) {
67      newPressDetected = false;
68      totalPresses++;
69
70      if (lastPressWasShort) {
71        shortPresses++;
72        shortDurationSum += lastPressDuration;
73        yellowBlinksRemaining = 5 * 2;
74      } else {
75        longPresses++;
76        longDurationSum += lastPressDuration;
77        yellowBlinksRemaining = 10 * 2;
78      }
79
80      yellowIsOn = false;
81    }
82
83    if (yellowBlinksRemaining > 0) {
84      yellowBlinksRemaining--;
85      yellowIsOn = !yellowIsOn;
86      if (yellowIsOn) {
87        ledYellowOn();
88      } else {
```

```
89        ledYellowOff();
90      }
91    } else if (yellowIsOn) {
92      ledYellowOff();
93      yellowIsOn = false;
94    }
95  }
96
97  void task3Report() {
98    rptTotalPresses = totalPresses;
99    rptShortPresses = shortPresses;
100   rptLongPresses = longPresses;
101   rptShortDurationSum = shortDurationSum;
102   rptLongDurationSum = longDurationSum;
103
104   totalPresses = 0;
105   shortPresses = 0;
106   longPresses = 0;
107   shortDurationSum = 0;
108   longDurationSum = 0;
109
110   reportReady = true;
111 }
112
113 void timer_handle_interrupts(int timer) {
114   systemTick++;
115   srvSchedulerTick();
116 }
117
118 void lab21Setup() {
119   setupGreenLed();
120   setupRedLed();
121   setupYellowLed();
122   buttonSetup();
123   srvSerialSetup();
124   srvSchedulerSetup(appTasks, NUM_TASKS);
125   timer_init_ISR_1KHz(TIMER_DEFAULT);
126
127   printf("Lab 3.1 - Button Press Monitor\n");
128   printf("Reports every 10 seconds\n");
129 }
130
131 void lab21Loop() {
132   if (reportReady) {
133     reportReady = false;
134
```

```
135     printf("--- Report ---\n");
136     printf("Total presses: %d\n", rptTotalPresses);
137     printf("Short (<500ms): %d\n", rptShortPresses);
138     printf("Long  (>=500ms): %d\n", rptLongPresses);
139 
140     if (rptTotalPresses > 0) {
141       unsigned long totalDur = rptShortDurationSum +
    rptLongDurationSum;
142       unsigned long avgDur = totalDur / rptTotalPresses;
143       printf("Avg duration: %lu ms\n", avgDur);
144     } else {
145       printf("Avg duration: N/A\n");
146     }
147 
148     printf("--------------\n");
149   }
150 }
```

Listing 3: lab2-1.cpp - Application Implementation

A.3 Scheduler Service

```
1  #ifndef SCHEDULER_SERVICE_H
2  #define SCHEDULER_SERVICE_H
3  
4  typedef void (*TaskFunc)(void);
5  
6  typedef struct {
7    TaskFunc func;
8    int recurrence;
9    int offset;
10   int counter;
11 } TaskContext;
12 
13 void srvSchedulerSetup(TaskContext *tasks, int numTasks);
14 
15 void srvSchedulerTick();
16 
17 #endif
```

Listing 4: scheduler.h - Scheduler Header

```
1  #include "scheduler.h"
2  
3  static TaskContext *schedulerTasks = 0;
```

```
static int schedulerNumTasks = 0;

void srvSchedulerSetup(TaskContext *tasks, int numTasks) {
  schedulerTasks = tasks;
  schedulerNumTasks = numTasks;
  for (int i = 0; i < numTasks; i++) {
    tasks[i].counter = tasks[i].offset;
  }
}

void srvSchedulerTick() {
  if (!schedulerTasks)
    return;

  for (int i = 0; i < schedulerNumTasks; i++) {
    if (--schedulerTasks[i].counter <= 0) {
      schedulerTasks[i].counter = schedulerTasks[i].recurrence;
      schedulerTasks[i].func();
      break;
    }
  }
}
```

Listing 5: scheduler.cpp - Scheduler Implementation

A.4 Button Driver

```
#ifndef BUTTON_DRIVER_H
#define BUTTON_DRIVER_H

#define BUTTON_PIN 2
void buttonSetup();
bool buttonIsPressed();

#endif
```

Listing 6: button.h - Button Driver Header

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

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

bool buttonIsPressed() { return digitalRead(BUTTON_PIN) == LOW; }
```

Listing 7: button.cpp - Button Driver Implementation

A.5 LED Driver

```h
#ifndef LED_DRIVER_H
#define LED_DRIVER_H

#include <Arduino.h>

#define LED_GREEN_PIN LED_BUILTIN
#define LED_RED_PIN 12
#define LED_YELLOW_PIN 11

void setupGreenLed();
void setupRedLed();
void setupYellowLed();

void ledGreenOn();
void ledGreenOff();
bool ledGreenIsOn();

void ledRedOn();
void ledRedOff();

void ledYellowOn();
void ledYellowOff();

#endif
```

Listing 8: led.h - LED Driver Header

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

int ledPinRed;
int ledPinGreen;
int ledPinYellow;

void setupGreenLed() {
  ledPinGreen = LED_GREEN_PIN;
  pinMode(ledPinGreen, OUTPUT);
}

```

```
void setupRedLed() {
  ledPinRed = LED_RED_PIN;
  pinMode(ledPinRed, OUTPUT);
}

void setupYellowLed() {
  ledPinYellow = LED_YELLOW_PIN;
  pinMode(ledPinYellow, OUTPUT);
}

void ledGreenOn() { digitalWrite(ledPinGreen, HIGH); }
void ledGreenOff() { digitalWrite(ledPinGreen, LOW); }

bool ledGreenIsOn() { return digitalRead(ledPinGreen) == HIGH; }

void ledRedOn() { digitalWrite(ledPinRed, HIGH); }
void ledRedOff() { digitalWrite(ledPinRed, LOW); }

void ledYellowOn() { digitalWrite(ledPinYellow, HIGH); }
void ledYellowOff() { digitalWrite(ledPinYellow, LOW); }
```

Listing 9: led.cpp - LED Driver Implementation

A.6 Serial Service

```
#ifndef SERIAL_DRIVER_STDIO_H
#define SERIAL_DRIVER_STDIO_H

#include <stdio.h>

int srvSerialPutChar(char ch, FILE *file);

int srvSerialGetChar(FILE *file);

void srvSerialSetup();

#endif
```

Listing 10: serial.h - Serial Service Header

```
#include "serial.h"
#include <Arduino.h>

int srvSerialPutChar(char ch, FILE *file) { return Serial.write(ch);
    }
```

```
5
6  int srvSerialGetChar(FILE *file) {
7    while (!Serial.available())
8      ;
9
10   return Serial.read();
11 }
12
13 void srvSerialSetup() {
14   Serial.begin(9600);
15   FILE *srvSerialStream = fdevopen(srvSerialPutChar,
       srvSerialGetChar);
16   stdin = stdout = srvSerialStream;
17 }
```

Listing 11: serial.cpp - Serial Service Implementation