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.1

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 study and use the STDIO library for serial communication and implementing a programm that would permit controlling the led with the following commands:

1. "led on" - turn the LED on
2. "lef off" - turn the LED off

1.1 Objectives of the laboratory work

Laboratory work has the following objectives that should be achieved:

- Understanding the basic principles of the serial communication with MCU
- Using the STDIO library for input/output message exchange
- Designing an application that interprets commands sent through serial interface
- Developing the code in a modular architecture, with separate functionalities for controling the peripherics

1.2 Problem definition

1. Configure application for interaction using STDIO library through serial interface for text exchange through terminal
2. Design an application for the MCU board that will intercept commands through terminal through serial interface to set up the state of the LED
    - "led on" for turning on the LED
    - "led off" for turning off the LED
    - system should respond with text messages that confirm the state change
    - for text exchange the stdio library should be used

2 Domain Analysis

2.1 Application Context and Utilized Technologies

The primary objective of this laboratory work is to establish a communication bridge between a workstation (PC) and an embedded system (MCU) using the **UART (Universal Asynchronous Receiver-Transmitter)** protocol. This interaction is fundamental in embedded engineering for debugging, real-time sensor monitoring, and remote command execution.

The application uses the standard input/output library (`stdio.h`) adapted for microcontroller environments. This approach allows the use of classic C functions, such as `printf()` and `scanf()/fgets()`, to abstract the hardware registers of the serial port. This provides a human-readable interface for transmitting and receiving text-based commands, moving away from raw byte manipulation to a more sophisticated command-line interaction.

2.2 Hardware and Software Components

Component	Description & Role
LAFVIN R3 (Arduino UNO R3)	Microcontroller board that contains the central processing unit (ATmega328P) that parses the incoming character stream and toggles GPIO pin states.
Serial-to-USB Interface	Facilitates the translation of the microcontrollers's logic levels into USB protocol for communication with the terminal.
LED - light emiting diode	The output peripheral used for visual validation. It uses a semiconductor that emits light at $20mA$ when electrical current flows through it
$220\,\Omega$ **Resistor**	The resistor ensures current limiting to protect the component. Overwise the LED will burn out
Neovim + Arduino-nvim plugin	The chosen development environment (Text editor + plugin, arduino-cli). Neovim provides high-efficiency editing, while the plugin integrates `arduino-cli` for compilation.
arduino-cli	`arduino-cli` is a more advanced cli tool for interacting with MCU. Useful for automation and usually provides faster workflow that Arduino IDE or VSCode plugins.
Breadboard	A construction base that contains internal conductive strips to create electrical paths. Used for prototyping electronics that allows components to be interconnected without soldering.
Serial Monitor	The terminal emulator (Ghostty) with `arduino-cli monitor` command used to send the "led on" and "led off" strings and display confirmation responses. Role of the interface for sending signals to MCU

2.3 System Architecture and Solution Justification

The system adopts a **Command Parser architecture**. This solution was chosen to ensure extensibility; by utilizing the `STDIO` library, the code becomes more portable and readable compared to direct manipulation of UART circular buffers using the device-specific libraries.

The data flow is structured as follows:

1. **Input:** The user enters a string command via the Ghostty terminal or another

terminal emulator.

2. **Transmission:** Data is sent asynchronously to the MCU via the TX/RX lines.
3. **Processing:** Functions like `fgets()` capture the string, which is then passed to a logic module that compares the input against predefined instructions.
4. **Action:** The MCU interacts with the LED driver module to change the physical state of the hardware.

2.4 Relevant Case Study

A real-world application of this concept is the **Command Line Interface (CLI)** found in network infrastructure equipment. Technicians connect via a console port (serial) to input text commands for hardware configuration. Similarly, in industrial automation, this interface is used for calibration of sensors where a complex graphical user interface (GUI) is not possible to get or required. In this case specialists may need to rely on visual outputs (led blinking patterns) or outputs in terminal interface

3 Design

3.1 Architectural sketch

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

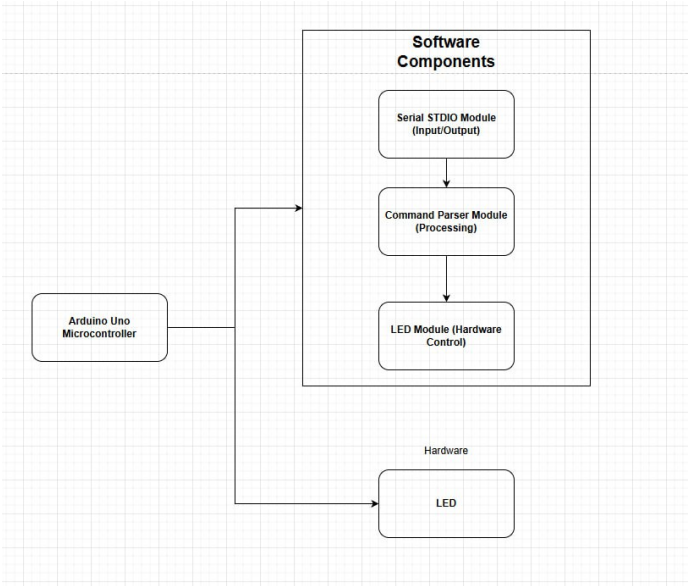

Figure 1: Architectural sketch of the project

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

1. **Arduino Uno microcontroller** - central processing unit board that serves as a executor of software
2. **LED** - hardware component which represents a semiconductor diode that serves as output for the user
3. **Serial STDIO module** - software module that serves as driver for adapting the default I/O commands like `printf();scanf()` to work with MCU's serial write and read
4. **Led module** - software module that is a driver that controls internally the state of the LED and provides the user-friendly abstractions that will not disclose internals about how it is implemented.

3.2 Electrical sketch

The circuit diagram outlines the setup for this project, showing the connections between the Arduino, an LED, and the computer for serial communication. It details the LED's connection to the Arduino's pin 13 (default LED pin), including a resistor to limit current and prevent damage to the LED by keeping it within safe electrical limits. The LED's circuit is completed by connecting its ground to the Arduino's ground.

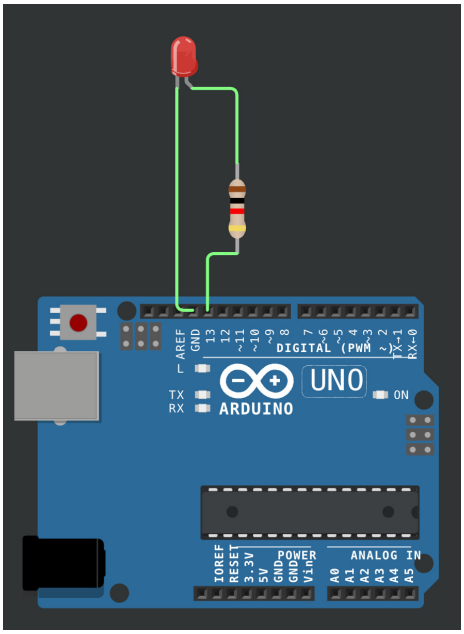

Figure 2: Circuit sketched on wokwi

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. Understanding the electrical connections is essential for both the construction and troubleshooting phases of the project.

3.3 Flow chart

The Figure 3 highlights how the Arduino's USB interface is used for serial communication with a computer. This enables the use of the Arduino IDE or terminal

software to issue commands like ”led on” and ”led off” to the Arduino, which then switches the LED on or off accordingly.

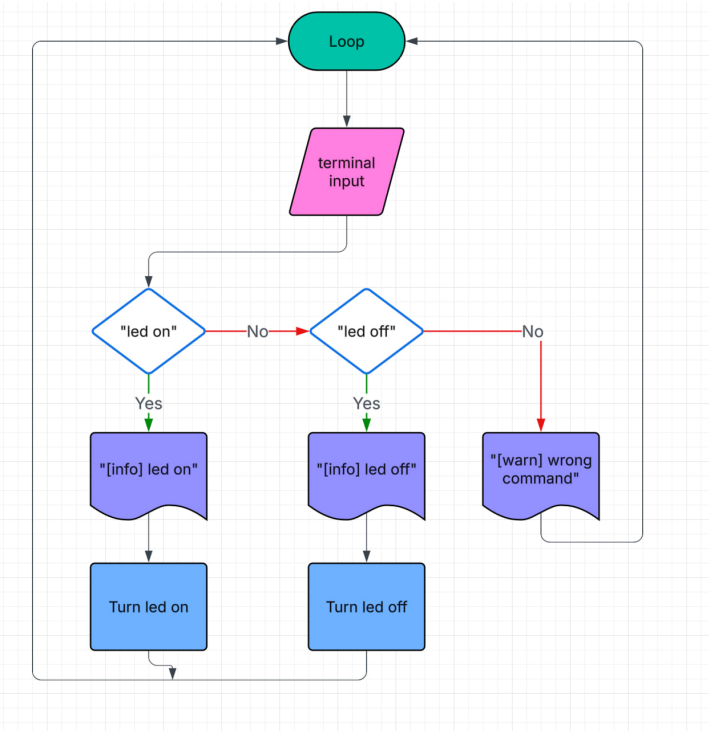

Figure 3: Flow chart of the algorithm

3.4 Code design

Code is developed to followed the principle of the interface and the implementation, by creating for each header file `.h` a respective `.cpp` file that implements it. At the same time the most important part of the implementation is separation of the **driver** and **abstraction** layer. Abstraction layer is the main portable program that calls low level modules. This makes the program portable to any device, ensuring that only the driver level is adapted properly. Driver level contains the low level logic related to specific hardware. In my case it contains all the calls to Serial port and LED power control using the `<Arduino.h>` library.

3.5 Results

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

```
# 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

$ 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.
WARNING: Your board may require different settings to work!

Monitor port settings:
  baudrate=9600
  bits=8
  dtr=on
  parity=none
  rts=on
  stop_bits=1

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

[DEBUG] System Ready. type led
[DEBUG] System Ready. type led on/led off to control the led
```

```
led on
[INFO] Led was turned on
led off
[INFO] Led was turned off
led sos
[SOS] SOS SOS
```

Below there is the visual representation of the circuit

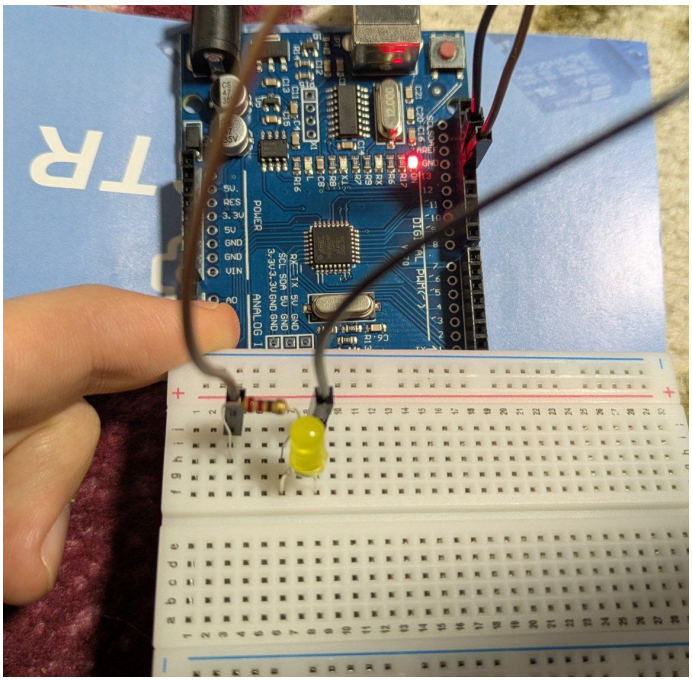

Figure 4: Led state after typing "led off"

After typing `led off` the visual state of the LED does not change (Figure 4). This is expected behavior since the electrical in not emitter from the LED pin.

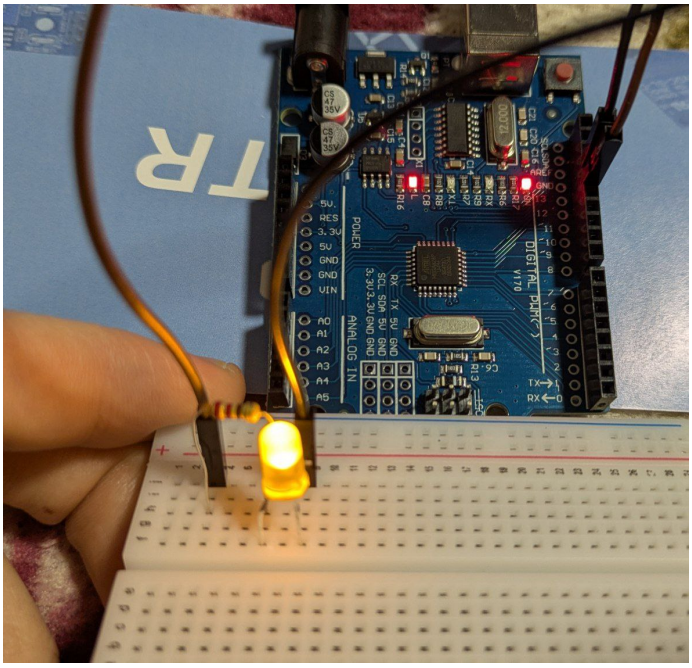

Figure 5: Led state after typing "led on"

After typing `led on` the visual state of the led changes to active, since now the electrical current is emited on the pin 13 (Figure 5).

Additionally the `led sos` command was implemented by using the `delay()` function that will stops the execution by specified time which allows to send the signals at the desired intervals of time. In this case it is used to send the sos signal in Morse code ...___... (3 long signals, 3 short signals, 3 long signals). The result of this command and other commands can be seen in the video posted on YouTube video hosting platform. `https://youtu.be/vp0BN4FRQHY?si=EbB3ROkQ5iNmlyLn`

4 Conclusion

Completions of this laboratory work have provided me insightful information about the basics of microelectronics, the embedded systems and details about how the hardware and software interacts to provide the desired results.

First laboratory work covered trivial use case of a microcontrolled board, specifically building a valid circuit that contains a led and a resistor and controlling the state of the led using the serial interface. This work was a good introduction that layed the foundation of the future process of work. Specifically:

1. Designing the software by dividing the abstraction layer and the driver layer. Program now is portable to any device.

2. Report template that will be used accross all the ES laboratory works which is designed according to TUM guildelines

3. Hands on experience on working with Arduino UNO other microcontroller boards

4. Understanding the basics of the interaction with peripherics (sensors, leds, resistors, etc).

This laboratory work is a good foundation in the further works, where the focus will be on developing further the engineering mindset.

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.

A Source Code

A.1 Main Sketch File

```
#include "src/lab1-1.h"

void setup() {
  lab11Setup();
}

void loop() {
  lab11Loop();
}
```

Listing 1: sketch.ino - Main Arduino Sketch

A.2 Lab1-1 Module

```
#ifndef LAB11_H
#define LAB11_H

void lab11Setup();

void lab11Loop();

#endif
```

Listing 2: lab1-1.h - Lab1-1 Header

```
#include "lab1-1.h"
#include "drivers/led/led.h"
#include "services/serial/serial.h"

#include <stdio.h>
#include <string.h>

void lab11Setup() {
  srvSerialSetup();
  setupLed();

  printf("\n[DEBUG] System Ready. type led on/led off to control the
    led\n");
}

```

```
char command[20];
void lab11Loop() {
  fgets(command, sizeof(command), stdin);
  command[strcspn(command, "\n")] = '\0';

  if (strcmp(command, "led on") == 0) {
    ledOn();
    printf("[INFO] Led was turned on\n");
  } else if (strcmp(command, "led off") == 0) {
    ledOff();
    printf("[INFO] Led was turned off\n");
  } else if (strcmp(command, "led sos") == 0) {
    ledSOS();
    printf("[SOS] SOS SOS\n");
  } else {
    printf("[WARNING] Unknown command\n");
  }
}
```

Listing 3: lab1-1.cpp - Lab1-1 Implementation

A.3 LED Driver

```
#ifndef LED_DRIVER_H
#define LED_DRIVER_H

#include <Arduino.h>

#define LED_PIN LED_BUILTIN

void setupLed();

void ledOn();
void ledOff();
void ledSOS();

#endif
```

Listing 4: led.h - LED Driver Header

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

int ledPin;

```

```
6  void setupLed() {
7    ledPin = LED_PIN;
8    pinMode(ledPin, OUTPUT);
9  };
10
11 void ledOn() { digitalWrite(ledPin, HIGH); }
12 void ledOff() { digitalWrite(ledPin, LOW); }
13 void ledSOS() {
14   // s
15   digitalWrite(ledPin, HIGH);
16   delay(150);
17   digitalWrite(ledPin, LOW);
18   delay(100);
19   digitalWrite(ledPin, HIGH);
20   delay(150);
21   digitalWrite(ledPin, LOW);
22   delay(100);
23   digitalWrite(ledPin, HIGH);
24   delay(150);
25   digitalWrite(ledPin, LOW);
26   delay(200);
27
28   // o
29   digitalWrite(ledPin, HIGH);
30   delay(300);
31   digitalWrite(ledPin, LOW);
32   delay(100);
33   digitalWrite(ledPin, HIGH);
34   delay(300);
35   digitalWrite(ledPin, LOW);
36   delay(100);
37   digitalWrite(ledPin, HIGH);
38   delay(300);
39   digitalWrite(ledPin, LOW);
40   delay(200);
41
42   // s
43   digitalWrite(ledPin, HIGH);
44   delay(150);
45   digitalWrite(ledPin, LOW);
46   delay(100);
47   digitalWrite(ledPin, HIGH);
48   delay(150);
49   digitalWrite(ledPin, LOW);
50   delay(100);
51   digitalWrite(ledPin, HIGH);
```

```
52   delay(150);
53   digitalWrite(ledPin, LOW);
54   delay(200);
55 }
```

Listing 5: led.cpp - LED Driver Implementation

A.4 Serial Service

```
1  #ifndef SERIAL_DRIVER_STDIO_H
2  #define SERIAL_DRIVER_STDIO_H
3
4  #include <stdio.h>
5
6  int srvSerialPutChar(char ch, FILE *file);
7
8  int srvSerialGetChar(FILE *file);
9
10 void srvSerialSetup();
11
12 #endif
```

Listing 6: serial.h - Serial Service Header

```
1  #include "serial.h"
2  #include <Arduino.h>
3
4  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 7: serial.cpp - Serial Service Implementation