Laboratory Work No. 1.1

User Interaction: STDIO - Serial Interface

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 Arduino Platform

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It consists of a microcontroller board and a development environment (IDE) for writing and uploading code.

What is Arduino Mega 2560:

- A microcontroller board based on the ATmega2560 chip
- Operating voltage: 5V
- 54 digital I/O pins (15 can be used as PWM outputs)
- 16 analog inputs
- 256 KB flash memory for storing code
- 8 KB SRAM and 4 KB EEPROM
- Clock speed: 16 MHz

How to work with Arduino:

1. **Connect** the Arduino board to the computer via USB cable
2. **Open** Arduino IDE and select the correct board (Tools → Board → Arduino Mega 2560)
3. **Select** the correct port (Tools → Port)
4. **Write** code with two main functions:
   - `setup()` - runs once at startup, used for initialization
   - `loop()` - runs repeatedly, contains main program logic
5. **Upload** the code by clicking the Upload button
6. **Monitor** serial output using Tools → Serial Monitor

1.1.2 Serial Communication (UART)

Serial communication is a fundamental method of data transfer in embedded systems. The UART (Universal Asynchronous Receiver/Transmitter) protocol enables information exchange between the microcontroller and computer through two lines: TX (transmission) and RX (reception).

Key parameters:

- **Baud rate:** 9600 bits per second (standard speed)
- **Data bits:** 8 bits per character
- **Stop bits:** 1 bit to signal end of character
- **Parity:** None (no error checking bit)

1.1.3 STDIO Library

The STDIO (Standard Input/Output) library provides standardized functions for input/output operations, such as `printf()` and `fgets()`, allowing an abstraction of serial communication.

Key functions used:

- `printf()` - formatted output to stdout
- `fgets()` - reads a line from stdin
- `fdev_setup_stream()` - connects custom I/O functions to stdin/stdout

On AVR microcontrollers (like ATmega2560), STDIO is not directly connected to any hardware. We must manually link it to the Serial interface using `fdev_setup_stream()`, which redirects `printf()` output to Serial TX and `fgets()` input from Serial RX.

1.2 Hardware Components Used

- **Arduino Mega 2560** - microcontroller based on ATmega2560, with 54 digital I/O pins
- **LED** - light-emitting diode for visual indication
- **220Ω Resistor** - limits current through the LED
- **Breadboard** - board for rapid prototyping
- **Jumper wires** - for connections

1.3 Software Components Used

- **Arduino IDE** - integrated development environment
- **STDIO Library** - for printf/fgets functions
- **Serial Monitor** - terminal for serial communication

1.4 System Architecture

The system is organized in three levels:

1. **Application level** - processing user commands
2. **STDIO level** - input/output abstraction
3. **Hardware level** - UART and GPIO control

1.5 Case Study

The developed application simulates a simple automation system where a user can control lighting through a text interface. Such systems are used in home automation, industrial systems, and IoT.

2 Design

2.1 Electrical Schematic

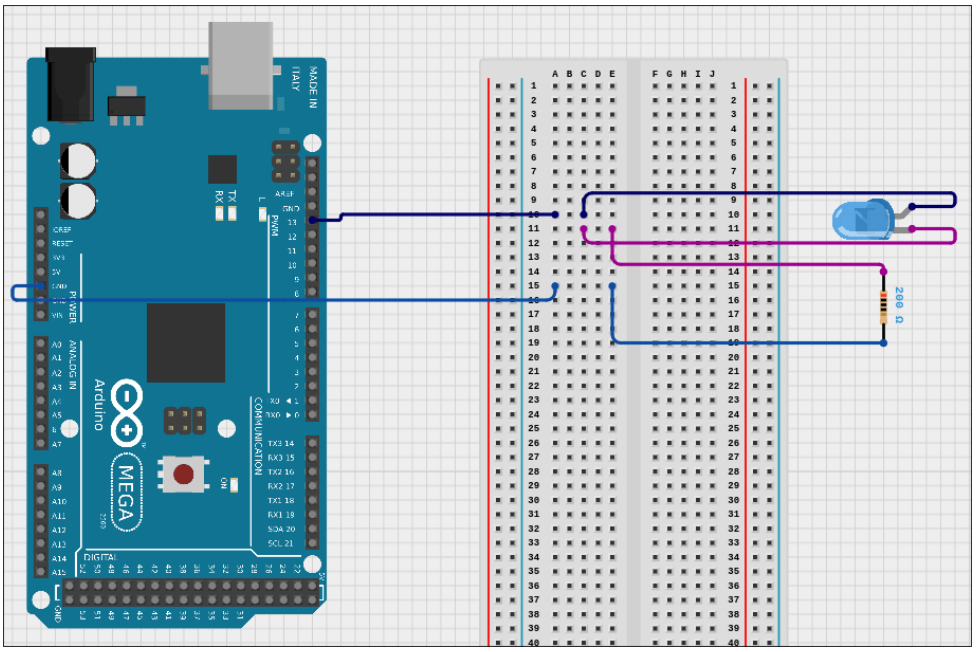

Figure 1: Circuit electrical schematic

Connections:

- **Wire 1:** Arduino **Pin 13** → Breadboard **10a** (carries the control signal to the LED)
- **LED Anode (+):** Breadboard **10c** (long leg, receives positive voltage from Pin 13)
- **LED Cathode (-):** Breadboard **11c** (short leg, connects to resistor)
- **Resistor end 1:** Breadboard **11e** (connects to LED cathode, same row)
- **Resistor end 2:** Breadboard **15e** (connects to ground wire)
- **Wire 2:** Arduino **GND** → Breadboard **15a** (completes the circuit to ground)

Circuit explanation: When Pin 13 outputs HIGH (5V), current flows from Pin 13 through the LED (causing it to light up), then through the 220Ω resistor (which limits current to protect the LED), and finally to GND, completing the circuit.

2.2 Project Structure

```
Lab1_STDIO/
 Lab1_STDIO.ino
     LED Module (ledInit, ledOn, ledOff, ledGetState)
     STDIO Module (serialPutchar, serialGetchar)
     Main Module (setup, loop, processCommand)
```

2.3 Modular Implementation

2.3.1 LED Module

Functions for LED control:

- `ledInit()` - initializes the pin as OUTPUT
- `ledOn()` - turns the LED on
- `ledOff()` - turns the LED off
- `ledGetState()` - returns the current state

2.3.2 STDIO Module

STDIO library configuration for serial communication:

- `serialPutchar()` - sends a character through Serial
- `serialGetchar()` - reads a character from Serial
- `fdev_setup_stream()` - links stdin/stdout to Serial

2.3.3 Main Module

The `processCommand()` function interprets commands:

- `led on` - turns the LED on
- `led off` - turns the LED off
- `status` - displays the LED state

3 Results Presentation

3.1 Testing LED ON Command

Figure 2: Serial Monitor - led on and status commands

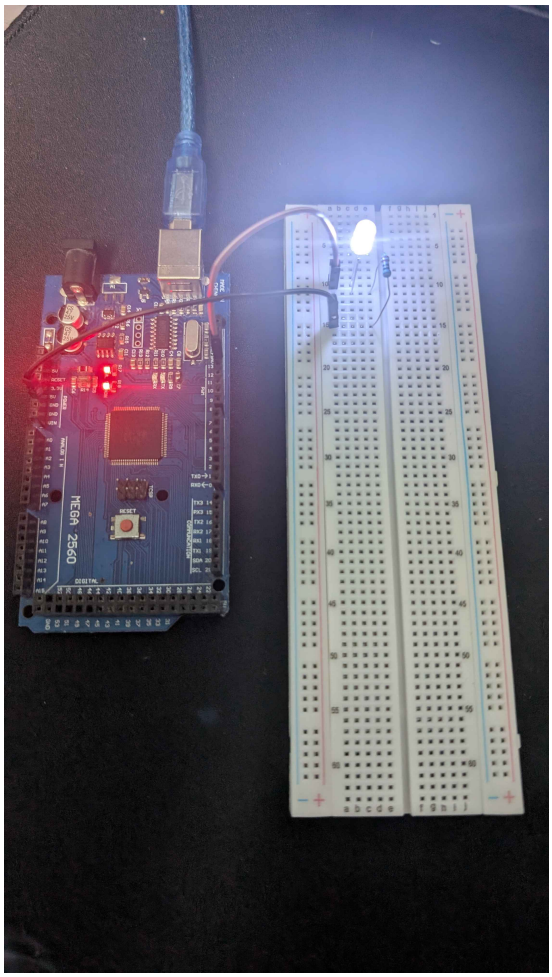

Figure 3: LED physically turned on

3.2 Testing LED OFF Command

Figure 4: Serial Monitor - led off and status commands

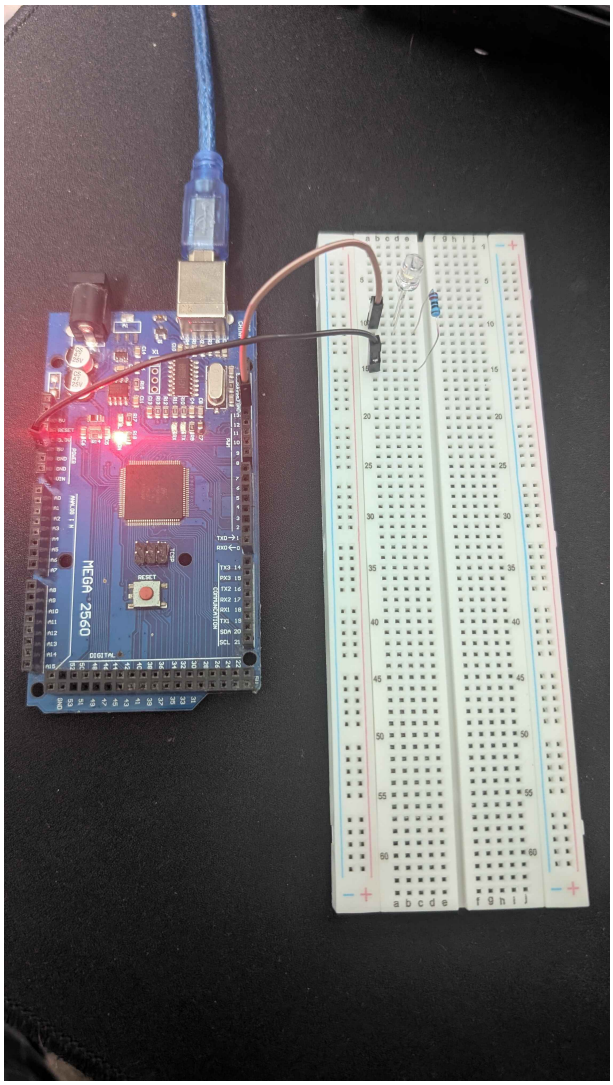

Figure 5: LED physically turned off

3.3 Functionality Demonstration

The system was successfully tested:

- The `led on` command physically turns the LED on
- The `led off` command turns the LED off
- The `status` command correctly reports the state
- Invalid commands generate an error message

4 Conclusions

4.1 Performance Analysis

The developed system successfully meets all the laboratory requirements. The serial communication operates reliably at 9600 baud, enabling smooth data exchange between the computer and the Arduino Mega 2560 microcontroller. The implementation demonstrates

proper use of the STDIO library, utilizing printf() for formatted output and fgets() for reading user input, which provides a clean abstraction layer over the hardware serial interface.

The command interpretation works correctly, recognizing "led on", "led off", and "status" commands while providing appropriate error messages for invalid input. The modular architecture separates LED control functions from the main program logic, making the code organized and maintainable.

4.2 Impact of Technology

This laboratory work demonstrates fundamental concepts that are widely applicable in embedded systems development. Serial communication and text-based interfaces remain essential in industrial automation, debugging tools, and IoT devices. Understanding how to bridge standard C libraries like STDIO with hardware peripherals is a valuable skill for developing professional embedded applications. The modular approach used in this project reflects industry best practices and prepares the foundation for more complex systems in future laboratory works.

5 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.

6 Bibliography

1. Arduino Reference - Serial Communication. `https://www.arduino.cc/reference/en/language/functions/communication/serial/`
2. AVR Libc Reference Manual - Standard IO facilities. `https://www.nongnu.org/avr-libc/user-manual/group__avr__stdio.html`
3. ATmega2560 Datasheet - Microchip Technology. `https://www.microchip.com/en-us/product/ATmega2560`

7 Appendix - Complete Source Code

Listing 1: Lab1_STDIO.ino

```
#include <stdio.h>

#define LED_PIN 13
bool ledState = false;

//=========================================
// LED MODULE
//=========================================

```

```
10 void ledInit(void) {
11     pinMode(LED_PIN, OUTPUT);
12     digitalWrite(LED_PIN, LOW);
13     ledState = false;
14 }
15
16 void ledOn(void) {
17     digitalWrite(LED_PIN, HIGH);
18     ledState = true;
19 }
20
21 void ledOff(void) {
22     digitalWrite(LED_PIN, LOW);
23     ledState = false;
24 }
25
26 bool ledGetState(void) {
27     return ledState;
28 }
29
30 //=============================================
31 // STDIO MODULE
32 //=============================================
33
34 int serialPutchar(char c, FILE *stream) {
35     Serial.write(c);
36     return c;
37 }
38
39 int serialGetchar(FILE *stream) {
40     while (!Serial.available());
41     return Serial.read();
42 }
43
44 FILE serialStream;
45
46 //=============================================
47 // MAIN MODULE
48 //=============================================
49
50 #define BUFFER_SIZE 32
51 char commandBuffer[BUFFER_SIZE];
52
53 void processCommand(const char* command) {
54     char cmd[BUFFER_SIZE];
55
56     for (uint8_t i = 0; command[i] != '\0' && i < BUFFER_SIZE -
       1; i++) {
57         cmd[i] = tolower(command[i]);
58         cmd[i + 1] = '\0';
59     }
```

```
60
61      if (strcmp(cmd, "led␣on") == 0) {
62          ledOn();
63          printf("[OK]␣LED␣on!\n");
64      }
65      else if (strcmp(cmd, "led␣off") == 0) {
66          ledOff();
67          printf("[OK]␣LED␣off!\n");
68      }
69      else if (strcmp(cmd, "status") == 0) {
70          if (ledGetState()) {
71              printf("[INFO]␣LED␣is␣ON\n");
72          } else {
73              printf("[INFO]␣LED␣is␣OFF\n");
74          }
75      }
76      else {
77          printf("[ERROR]␣Unknown␣command!\n");
78      }
79  }
80
81  void setup() {
82      Serial.begin(9600);
83      ledInit();
84
85      fdev_setup_stream(&serialStream, serialPutchar, serialGetchar
            , _FDEV_SETUP_RW);
86      stdin = &serialStream;
87      stdout = &serialStream;
88
89      printf("Commands:␣led␣on,␣led␣off,␣status\n");
90  }
91
92  void loop() {
93      if (Serial.available() > 0) {
94          fgets(commandBuffer, BUFFER_SIZE, stdin);
95
96          char *newline = strchr(commandBuffer, '\n');
97          if (newline) *newline = '\0';
98          newline = strchr(commandBuffer, '\r');
99          if (newline) *newline = '\0';
100
101         if (strlen(commandBuffer) > 0) {
102             processCommand(commandBuffer);
103         }
104     }
105 }
```