Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Software Engineering Department

Embedded Systems

Laboratory work #1.2

User Interaction: STDIO - LCD + Keypad

Author:
Sava Luchian
std. gr. FAF-233

Verified:
Alexei Martiniuc

Chisinau 2026

Purpose of the Laboratory

To familiarize students with user interaction peripherals (LCD and 4x4 keypad) using STDIO services (`printf`, `fprintf`, `fscanf`) and implement an access-code verification application with visual feedback via LEDs.

Objectives

1. Understand basic principles of user interaction peripherals in embedded systems (LCD, keypad, LEDs)
2. Use STDIO library for text exchange between user and MCU
3. Design an MCU application that interprets keypad input and prints messages on LCD
4. Develop a modular solution with separated peripheral and application modules

Problem Definition

Configure the application for STDIO-based text exchange through LCD + keypad. Design an MCU application that detects a 4-digit code from a 4x4 keypad, validates it, and displays messages on LCD:

- Valid code → Green LED ON
- Invalid code → Red LED ON
- Use STDIO for keypad scanning and LCD text printing
- Implement modular code architecture

1 Domain Analysis

1.1 Technological Analysis and Application Context

The laboratory targets embedded Human-Machine Interaction (HMI) by combining a matrix keypad (user input), character LCD (user output), and status LEDs (binary visual feedback). The system is implemented on Arduino Uno (ATmega328P), chosen for deterministic timing, simple GPIO model, and compatibility with PlatformIO and Wokwi simulation.

STDIO is used as an abstraction layer over physical peripherals:

- `printf()` for serial diagnostics
- `fprintf(lcdStream,...)` for LCD text output
- `fscanf(keypadStream,...)` for keypad character input

This approach provides a portable and structured I/O interface similar to classical C applications while preserving low-level hardware control.

1.2 Hardware Components and Justification

- **Arduino Uno (ATmega328P)**: Main MCU at 16 MHz, 32 KB flash, 2 KB SRAM. Handles keypad scan events, LCD updates, and LED control.
- **LCD 16x2 (HD44780-compatible, 4-bit mode)**: Displays prompts, entered code, and result messages.
- **4x4 Matrix Keypad**: Captures user commands and 4-digit access codes.
- **Green LED + Red LED**: Shows success/failure state.
- **2x 220 Ω resistors**: Current limiting for LEDs.
- **Breadboard/Jumper wires/USB power**: Interconnection and 5V supply.

1.3 Software Components

- **PlatformIO + VS Code**: Multi-file project support, dependency management, and reproducible builds.
- **Arduino Framework**: GPIO, timing, and serial abstractions.
- **Libraries**: `LiquidCrystal` and `Keypad`.
- **AVR-libc STDIO**: Stream setup with `fdev_setup_stream()`.
- **Wokwi Simulator**: Functional verification of the full hardware design.

1.4 System Justification

The selected architecture separates peripherals from business logic, enabling reuse in subsequent labs. The design minimizes coupling by introducing a dedicated STDIO bridge layer, so application code does not directly depend on LCD/Keypad low-level APIs.

2 Design

2.1 Architectural Sketch (HW–SW Integration)

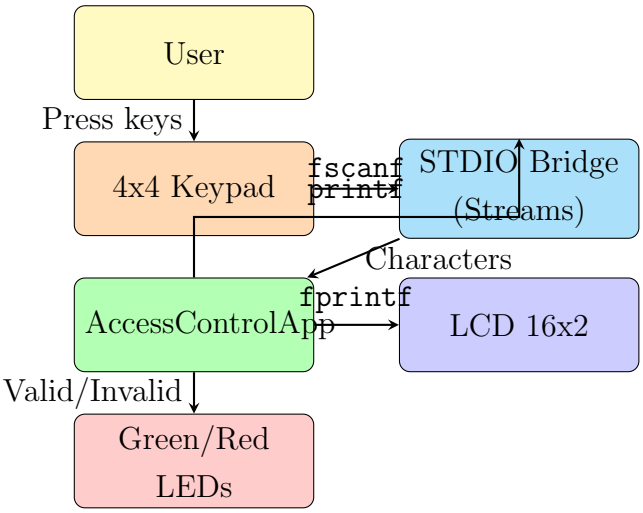

Figure 1: Block architecture and software-hardware communication

Communication model:

- Input path: Keypad → STDIO key stream → application logic
- Output path: Application logic → LCD STDIO stream and LED GPIO drivers
- Debug path: Application logic → Serial monitor

2.2 Electrical Sketch and Interconnection Explanation

Pin mapping used in implementation:

- LCD: RS=D13, E=D12, D4=D11, D5=D10, D6=D9, D7=D8, RW=GND, VO=GND, VCC=5V
- Keypad rows: D7, D6, D5, D4
- Keypad columns: D3, D2, A3, A2
- LEDs: Green=A0 (through 220 Ω), Red=A1 (through 220 Ω)

2.3 Behavioral Design (Finite-State Logic)

Main functional states:

1. **PROMPT**: Show `Enter Code:` and wait for first key

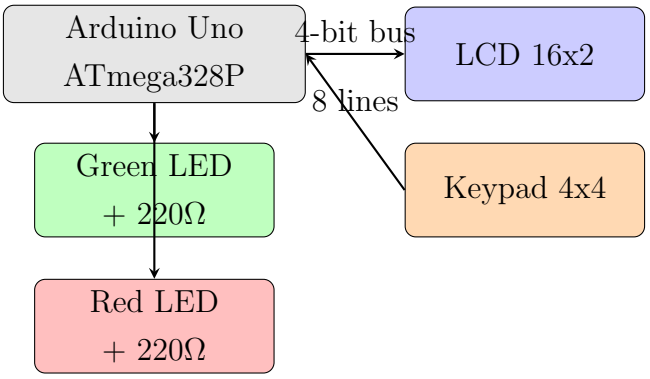

Figure 2: Electrical interconnection sketch (functional)

2. **VERIFY**: Read 4 digits and validate against active code
3. **RESULT**: Show `Access Granted` or `Access Denied`, set LEDs
4. **CHANGE MODE (Optional)**: Triggered by key `A`; read old code, then new code

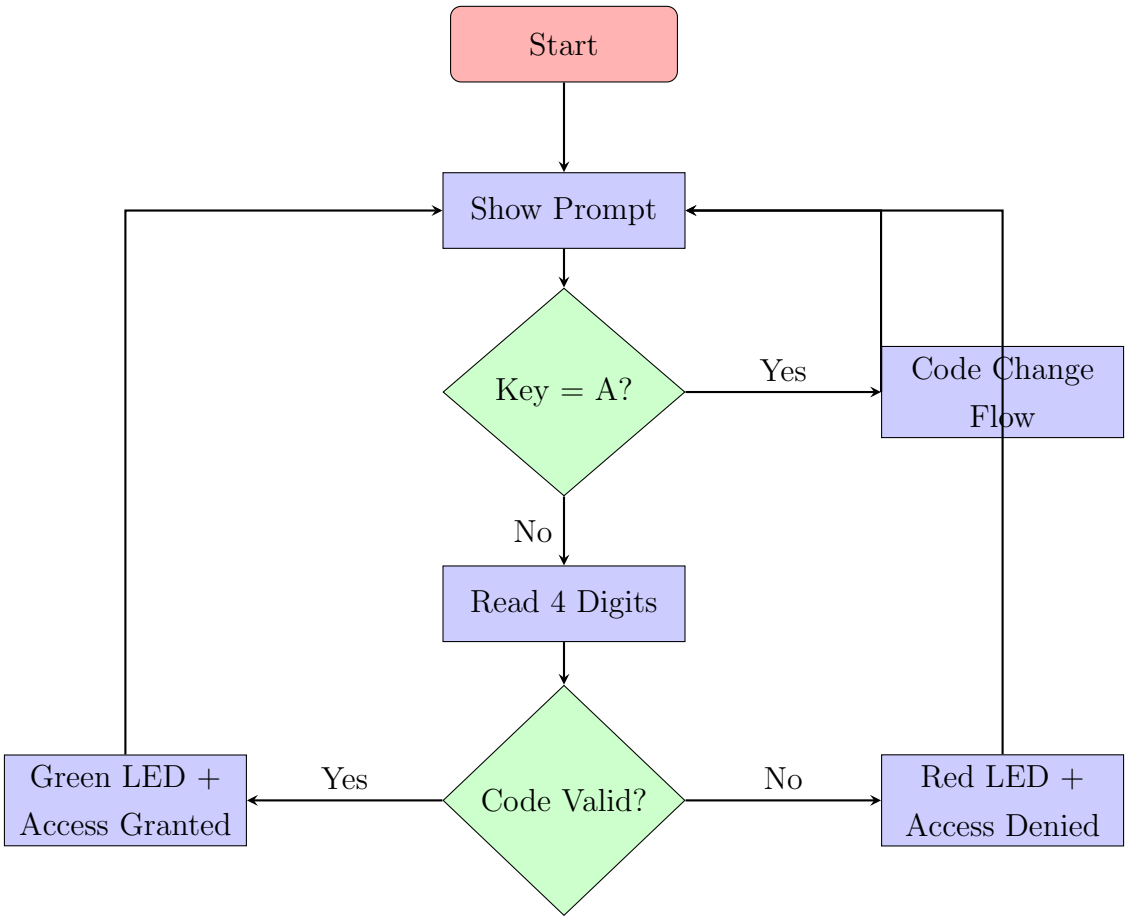

Figure 3: Application flowchart

2.4 Project Structure and Modular Implementation

```
lab2/
|-- platformio.ini
|-- diagram.json
|-- wokwi.toml
|-- src/
|   |-- main.cpp
|   |-- config/
|   |   |-- AppConfig.h
|   |   `-- AppConfig.cpp
|   |-- peripherals/
|   |   |-- LedController.h/.cpp
|   |   |-- LcdDisplay.h/.cpp
|   |   `-- KeypadInput.h/.cpp
|   |-- io/
|   |   `-- StdioBridge.h/.cpp
|   `-- app/
|       `-- AccessControlApp.h/.cpp
`-- report/main.tex
```

Coding conventions and separation principles:

- Header/source separation for each module (interface vs implementation)
- CamelCase for classes and methods
- Magic numbers moved into `AppConfig`
- Single-responsibility modules for each peripheral/service

3 Implementation

3.1 Description of Software Modules

- **AppConfig**: Constants for pins, keypad layout, code length, LCD dimensions.
- **LedController**: Controls green/red outputs (grant/deny feedback).
- **LcdDisplay**: Wraps LCD initialization and primitive display operations.
- **KeypadInput**: Wraps keypad scanning interface.
- **StdioBridge**: Implements stream redirection between STDIO and Serial/LCD/Keypad.

- **AccessControlApp**: Implements verification logic and optional code-change workflow.

3.2 STDIO Utilization

The application uses STDIO in all critical paths:

- `printf(...)` for serial diagnostics and interaction trace
- `fprintf(StdioBridge::lcdStream(), ...)` for LCD messages
- `fscanf(StdioBridge::keypadStream(), ...)` for keypad character acquisition

Implementation concept:

- `fdev_setup_stream()` binds custom put/get functions to `FILE` streams.
- LCD stream handles cursor state and line transitions.
- Keypad stream blocks until a key is pressed, then returns one character.

3.3 Functional Implementation Coverage

1. Read entered code from keypad (4 digits)
2. Validate entered code against active code
3. Display result on LCD
4. Turn ON green LED for valid code
5. Turn ON red LED for invalid code
6. Optional feature: change code from keypad by pressing `A`

4 Results

4.1 Simulation and Build Evidence

Build result (PlatformIO):

```
PLATFORM: Atmel AVR > Arduino Uno
Building .pio/build/uno/firmware.hex
[SUCCESS]
```

Wokwi runtime observations:

- Typing `1234` displays `Access Granted`; green LED turns ON
- Typing an incorrect code displays `Access Denied`; red LED turns ON
- Pressing `A` enters code-change mode (old code → new code)
- After successful update, new code is accepted and old code is rejected

4.2 Representative Interaction Logs

```
Lab 1.2 started. Enter 4 digits on keypad.
Press A to change the code.
Entered code: 1234
Result: VALID
Entered code: 1111
Result: INVALID
Code changed successfully. New code: 5678
Entered code: 1234
Result: INVALID
Entered code: 5678
Result: VALID
```

4.3 Performance and Resource Use

- Flash usage: 9.4 KB (about 29% of 32 KB)
- RAM usage: 787 B (about 38% of 2 KB)
- Response behavior: immediate user feedback suitable for educational HMI tasks

Note on AI Tool Usage

AI-assisted tools were used to support coding productivity and report drafting:

- GitHub Copilot (code suggestions, refactoring support)
- LLM assistance for technical writing structure and language polishing

All generated content was manually reviewed, adapted, compiled, and validated against simulation behavior.

Bibliography

1. AVR-libc Documentation – STDIO for AVR microcontrollers.
   `https://www.nongnu.org/avr-libc/user-manual/group__avr__stdio.html`

2. Arduino Official Reference – `Serial`, GPIO, and framework APIs.
   `https://www.arduino.cc/reference/en/`

3. PlatformIO Documentation – project configuration and toolchain.
   `https://docs.platformio.org/en/latest/`

4. Keypad Library (Chris–A) – matrix keypad handling for Arduino.
   `https://github.com/Chris--A/Keypad`

5. LiquidCrystal Library – HD44780 LCD control in 4-bit mode.
   `https://github.com/arduino-libraries/LiquidCrystal`

Annex – Source Code and Artifacts

Repository Artifacts

```
lab2/
|-- platformio.ini
|-- diagram.json
|-- wokwi.toml
|-- README.md
|-- src/...
‘-- report/main.tex
```

Key Source Files

- src/main.cpp
- src/app/AccessControlApp.h/.cpp
- src/io/StdioBridge.h/.cpp
- src/peripherals/LedController.h/.cpp
- src/peripherals/LcdDisplay.h/.cpp
- src/peripherals/KeypadInput.h/.cpp
- src/config/AppConfig.h/.cpp

GitHub Repository

Repository URL:
https://github.com/Ekkusuu/embedded-systems-repo/tree/main/lab2

5 Conclusions

The laboratory objectives were met. A complete modular HMI solution was implemented on Arduino Uno using STDIO-mediated communication between keypad, LCD, and application logic. The final system validates 4-digit codes, provides clear visual status feedback through LEDs, and includes an optional runtime code-update feature.

Key outcomes:

1. Practical integration of LCD + keypad with clean HW–SW boundaries
2. Correct use of STDIO streams in an embedded context

3. Reusable modular project structure for future laboratory work