Ministry of Education of Republic of Moldova

Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Department of Physics

Embedded Systems

Laboratory work #1.2

Electronic Lock System with LCD and Keypad using STDIO

Author:
Dmitrii Belih
std. gr. FAF-232

Verified:
Martiniuc A.

Chișinău 2026

1. Domain Analysis

1.1. Purpose of the Laboratory Work

The purpose of this laboratory work is to understand the principles of user interaction with embedded systems using LCD displays and matrix keypads, and to implement a password verification system using the STDIO library. The work involves setting up an I2C-based LCD display for output, a 4×4 matrix keypad for input, and an access control system that verifies 4-digit PIN codes. The system demonstrates the use of STDIO abstraction for debugging and serial communication while implementing a finite state machine (FSM) for managing user interaction states. Additional functionality includes a programming mode for changing the password, with visual feedback through LEDs.

1.2. Technologies Used

Standard Input/Output (STDIO)

Standard Input/Output (STDIO) provides a standardized mechanism for handling input and output operations in embedded systems. In this laboratory work, STDIO is redirected to the serial interface (UART) for debugging and system monitoring purposes. The STDIO library enables the use of familiar C functions such as `printf()` for formatted output and `fprintf()` for stream-specific output. This abstraction layer simplifies debugging and logging, allowing developers to track system state, input events, and operational status without implementing custom serial communication routines.

I2C Communication Protocol

Inter-Integrated Circuit (I2C) is a synchronous, multi-master, multi-slave, packet-switched, single-ended, serial communication bus. In this laboratory work, I2C is used to communicate with the LCD display module. I2C requires only two signal lines (SDA for data and SCL for clock) plus ground, significantly reducing the number of GPIO pins required for peripheral connections. The protocol uses a 7-bit addressing scheme, allowing multiple devices to share the same bus. LCD modules with I2C interfaces typically use address 0x27 or 0x3F.

Matrix Keypads

Matrix keypads are input devices that arrange buttons in a grid pattern (rows and columns) to minimize the number of required pins. A 4×4 keypad uses 8 pins (4 rows + 4 columns) instead of 16 individual pins. Keypad scanning involves sequentially activating each row and reading the column states to detect pressed buttons. This technique requires debouncing (software or hardware) to prevent false triggers from contact bounce. Keyboards

with matrix layouts are widely used in security systems, ATMs, and industrial control panels.

Finite State Machine (FSM)

A Finite State Machine is a computational model used to design systems that can be in one of a finite number of states. In this laboratory work, an FSM manages the access control system's behavior across different states: Welcome/Normal Mode, Input Processing, Access Granted, Access Denied, and Programming Mode. Each state has specific behaviors, transitions, and responses to user input. The FSM approach ensures clear separation of concerns, predictable system behavior, and easier maintenance and debugging.

LCD Display (Liquid Crystal Display)

LCD displays are visual output devices commonly used in embedded systems to present information to users. Character-based LCDs (such as 16×2) can display text in fixed character positions. Modern LCD modules often include I2C interfaces for simplified connectivity. The LiquidCrystal_I2C library provides a convenient API for initializing the display, setting cursor positions, printing text, and controlling backlight. LCDs are ideal for user interfaces in embedded systems due to their low power consumption, readability, and ability to display dynamic information.

1.3. Hardware Components

Arduino Uno

The Arduino Uno is based on the ATmega328P microcontroller featuring 14 digital I/O pins, 6 analog inputs, and 32 KB flash memory. For this laboratory work, the Arduino provides GPIO pins for keypad control (8 pins), LED control (3 pins), and I2C communication (2 pins: SDA on A4, SCL on A5). The board operates at 5V and provides sufficient processing power for real-time keypad scanning and state machine management.

4×4 Matrix Keypad

The 4×4 matrix keypad consists of 16 buttons arranged in 4 rows and 4 columns. The typical layout includes digits 0-9, letters A-D, and special keys (*) and (#). The keypad requires 8 GPIO pins: 4 rows configured as outputs and 4 columns configured as inputs with pull-up resistors. Keys are detected by scanning: each row is sequentially set LOW, and columns are read to detect when a button connects a row to a column.

LCD 16×2 with I2C Interface

The 16×2 character LCD can display 32 characters (16 per line) across two lines. When equipped with an I2C interface board, it requires only 2 wires (SDA and SCL) plus power (VCC and GND). The I2C interface typically includes a PCF8574 I/O expander chip and a potentiometer for contrast adjustment. The default I2C address is 0x27, though some modules use 0x3F.

LED Indicators

Three LEDs provide visual feedback:

- **Green LED (Pin 12):** Indicates successful authentication. Lights for 5 seconds when correct password is entered.
- **Red LED (Pin 13):** Indicates authentication failure. Blinks 5 times when incorrect password is entered.
- **Programming LED (Pin 3):** Indicates programming mode is active. Lights when user enters password change mode.

Each LED requires a current-limiting resistor (220 typical) to prevent damage to both the LED and the microcontroller.

1.4. Software Components

LiquidCrystal_I2C Library

The LiquidCrystal_I2C library provides an Arduino-compatible interface for controlling I2C-based LCD displays. It wraps the Wire library (I2C implementation) and offers methods for initialization (`init()`), text output (`print()`, `write()`), cursor control (`setCursor()`), display clearing (`clear()`), and backlight control (`backlight()`, `noBacklight()`). The library handles I2C communication details, allowing developers to focus on display content.

Wire Library (I2C)

The Wire library is Arduino's built-in implementation of the I2C protocol. It provides methods for initializing the I2C bus (`Wire.begin()`), starting transmission to a device (`Wire.beginTransmission()`), writing data (`Wire.write()`), ending transmission (`Wire.endTransmis`
and requesting data from devices (`Wire.requestFrom()`). This library is used internally by the LiquidCrystal_I2C library and for I2C device scanning.

PlatformIO

PlatformIO provides an advanced development environment for embedded systems with features including intelligent code completion, multi-platform build systems, library management, and debugging capabilities. For this laboratory work, PlatformIO manages dependencies (LiquidCrystal_I2C library), handles compilation, uploads firmware to the Arduino, and provides a serial monitor for debugging output. The platformio.ini file specifies build flags, include paths, and library dependencies.

1.5. System Architecture and Justification

The system architecture follows a layered, modular approach with clear separation of concerns:

- **Hardware Layer:** Consists of Arduino Uno microcontroller, 4×4 matrix keypad (8 GPIO pins), LCD 16×2 with I2C interface (2 pins), and three LEDs (3 GPIO pins). The microcontroller provides computational resources and GPIO/I2C interfaces.
- **Hardware-Software Interface (Driver Layer):** Implements low-level drivers for each hardware component:
  - *Keypad Driver:* Handles matrix scanning, debouncing, and key detection
  - *LCD Driver:* Wraps LiquidCrystal_I2C library with additional logging
  - *LED Driver:* Provides simple on/off/toggle control
  - *Serial STDIO Driver:* Redirects stdout/stdin to UART for debugging
- **Application Logic Layer (FSM):** Implements the finite state machine that manages system states:
  - *Welcome State:* Displays "Enter Code:"prompt, accepts first key
  - *Input State:* Accumulates 4-digit PIN, displays asterisks
  - *Verify State:* Compares input with stored password
  - *Access Granted State:* Shows success message, activates green LED
  - *Access Denied State:* Shows error message, blinks red LED
  - *Programming Mode State:* Allows password change via 'D' key
- **STDIO Service Layer:** Provides formatted output for debugging and system monitoring through serial interface, enabling real-time visibility into system operation.

This architecture was chosen to promote modularity, reusability, and maintainability. Each hardware component has a dedicated driver, the FSM clearly separates behavioral states, and the STDIO layer provides non-intrusive debugging. This design allows easy extension (adding more states, different passwords, or additional security features) without modifying core components.

1.6. Case Study: Real-World Access Control Systems

Access control systems are ubiquitous in modern security applications, from office buildings and ATM machines to hotel rooms and secure facilities. These systems typically require:

- **User Input:** Keypads, card readers, or biometric scanners
- **Display/Feedback:** LCD screens, LED indicators, or audio prompts
- **Authentication Logic:** Password verification, card validation, or biometric matching
- **Access Control:** Door locks, turnstiles, or electronic barriers
- **Management Features:** Password changes, user management, or configuration modes

Our laboratory work implements a simplified version of such a system using a 4×4 keypad for input, an LCD display for user feedback, and LEDs for status indication. The programming mode (activated by the 'D' key) demonstrates real-world system management capabilities, allowing authorized users to change access codes without reprogramming the device.

The finite state machine approach used here mirrors professional access control systems, where clear state definitions and transitions ensure predictable behavior and simplify debugging. Software debouncing for keypad input addresses real-world challenges with mechanical switches, where contact bounce can cause false triggers. The modular architecture allows components to be reused in future laboratory works, simulating industry practices of code reuse and library development.

2. Design

2.1. Architectural Sketch

System Architecture Overview:

The system follows a layered architecture with clear separation between hardware, drivers, and application logic.

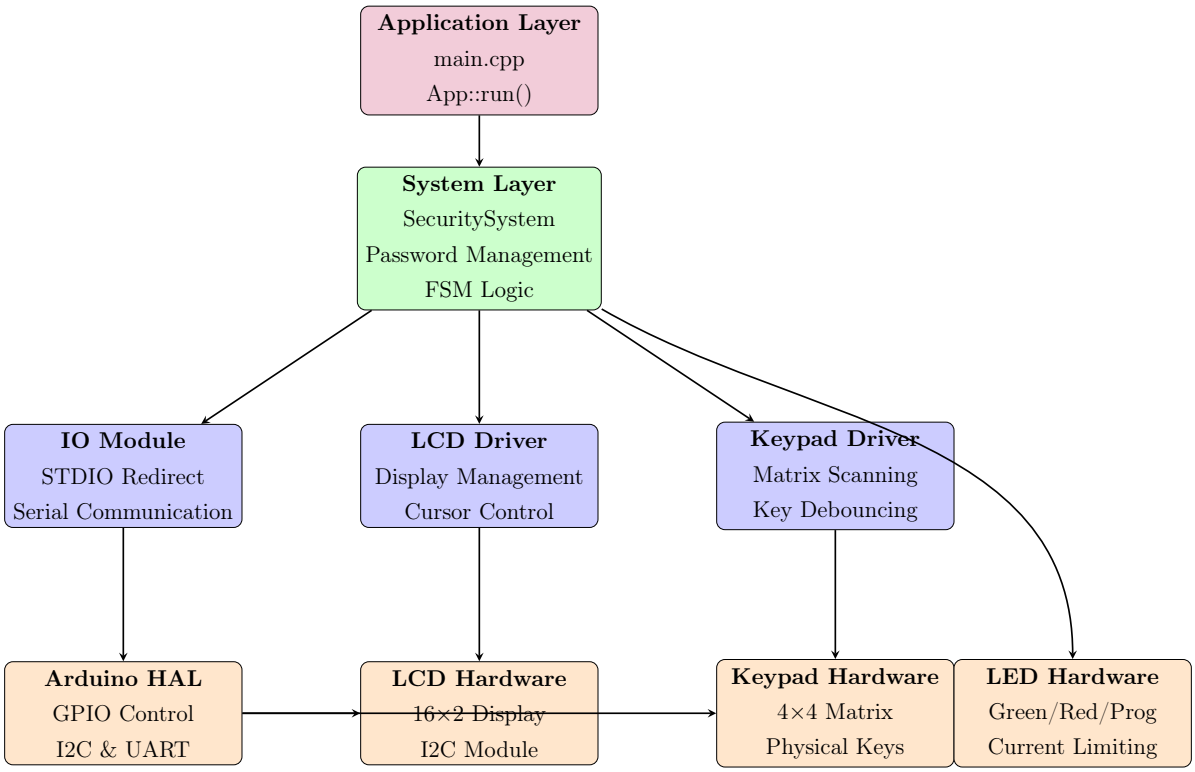

Рис. 1: System Architecture Diagram

Module Block Scheme (App Class):

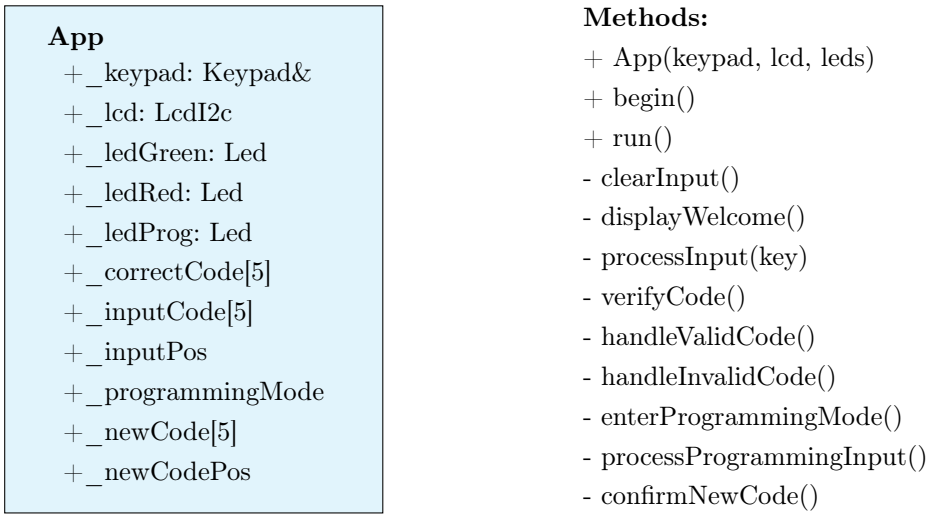

Рис. 2: App Module Block Scheme

State Descriptions and Transitions:

1. **Welcome State (Initial):**
   - Display: "Enter Code: [ ]"
   - Input: Accept first digit (0-9) or 'D' key for programming mode

- Transitions: Digits → Input State, 'D' → Programming Mode

2. **Input State (Accumulating PIN):**
   - Display: "Enter Code: [****]"(shows asterisks for each digit)
   - Input: Accept digits 0-9 (up to 4 digits), '*' to clear, '#' to verify
   - Transitions: Full 4 digits + '#' → Verify State, '*' → Welcome State
3. **Verify State (Authentication):**
   - Internal: Compare input PIN with stored password
   - Transitions: Match → Access Granted State, Mismatch → Access Denied State
4. **Access Granted State:**
   - Display: "ACCESS GRANTED! Door Unlocked"
   - Action: Green LED ON for 5 seconds
   - Transition: After 5 seconds + 3 second delay → Welcome State
5. **Access Denied State:**
   - Display: "ACCESS DENIED! Wrong Code"
   - Action: Red LED blinks 5 times (150ms on, 150ms off)
   - Transition: After 3 second delay → Welcome State
6. **Programming Mode State:**
   - Display: "PROG MODE | New Pass [ ]"
   - Action: Programming LED ON
   - Input: Accept new 4-digit password, '*' to cancel, '#' to save
   - Transitions: '#' after 4 digits → Save Welcome, '*' → Welcome (cancel)

2.2. Main File State Diagram

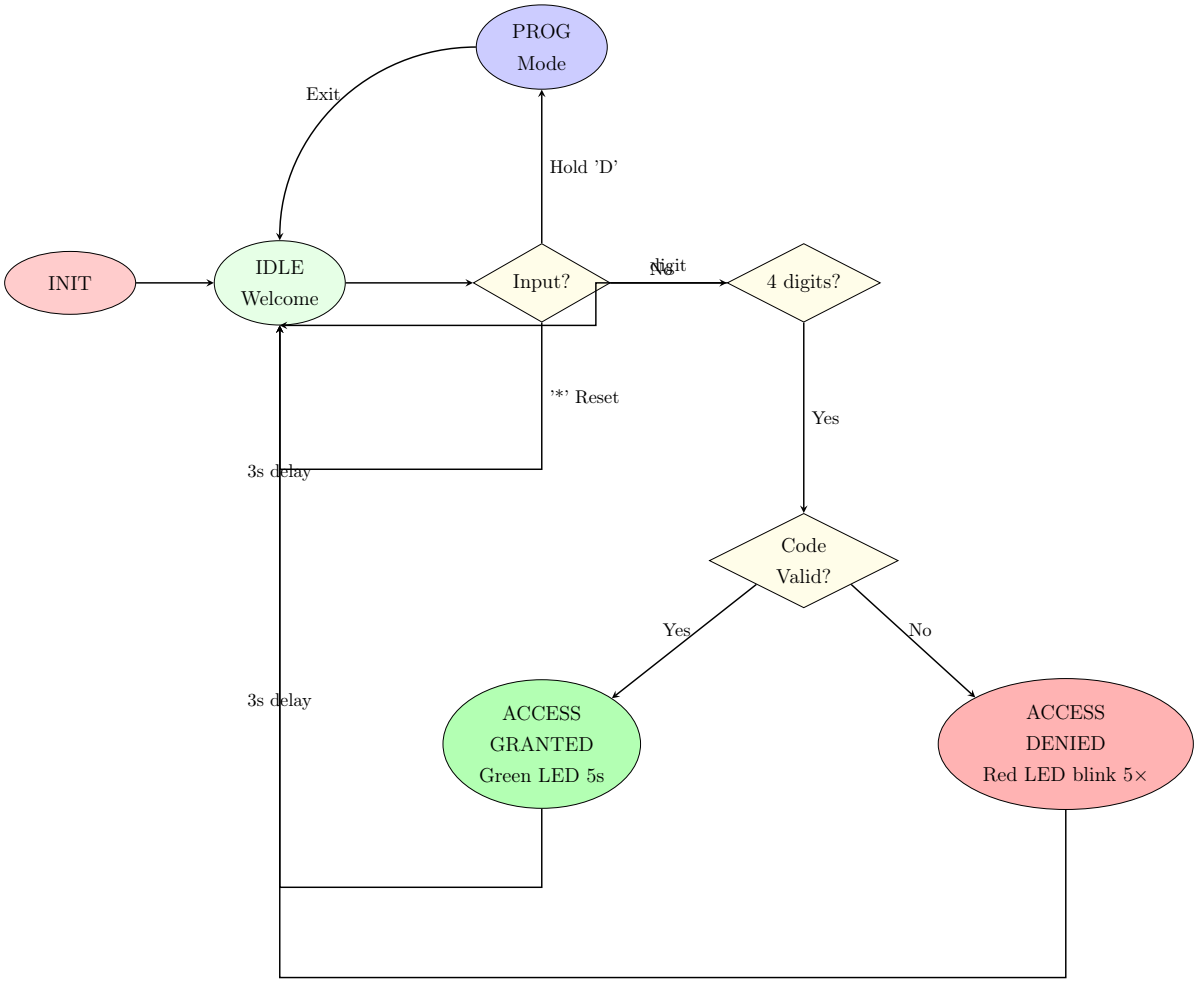

Рис. 3: Main Application State Machine

State Descriptions:

1. **INIT:** System initialization, setup peripherals, display welcome message
2. **IDLE:** Waiting for user input, shows "Enter Code: [ ]"
3. **PROG MODE:** Password change mode, programming LED ON
4. **ACCESS GRANTED:** Authentication successful, green LED 5s, "Door Unlocked"
5. **ACCESS DENIED:** Authentication failed, red LED blinks 5×, "Wrong Code"

2.3. Hardware-Software Interface Architecture

The system implements a layered hardware-software interface following MCAL/ECAL/SRV architecture:

Microcontroller Abstraction Layer (MCAL):

- **GPIO Control:** Direct manipulation of Arduino pins via `digitalWrite()` and `digitalRead()`
- **I2C Bus:** Low-level I2C communication via Wire library (`Wire.begin()`, `Wire.beginTransmissi`
- **UART:** Serial communication via `Serial.begin()`, `Serial.write()`, `Serial.read()`

Enhanced Control Abstraction Layer (ECAL):

- **Keypad Driver:** Implements matrix scanning algorithm with debouncing (20ms press debounce, 50ms release debounce)
- **LED Driver:** Provides abstracted LED control (`on()`, `off()`, `toggle()`) with state tracking
- **LCD Driver:** Wraps LiquidCrystal_I2C library with cursor management, text output, and backlight control

Service Layer (SRV):

- **Application Controller:** FSM implementation managing system states and transitions
- **Password Management:** Secure storage and verification of access codes
- **Debug Service:** STDIO redirection for system monitoring and logging

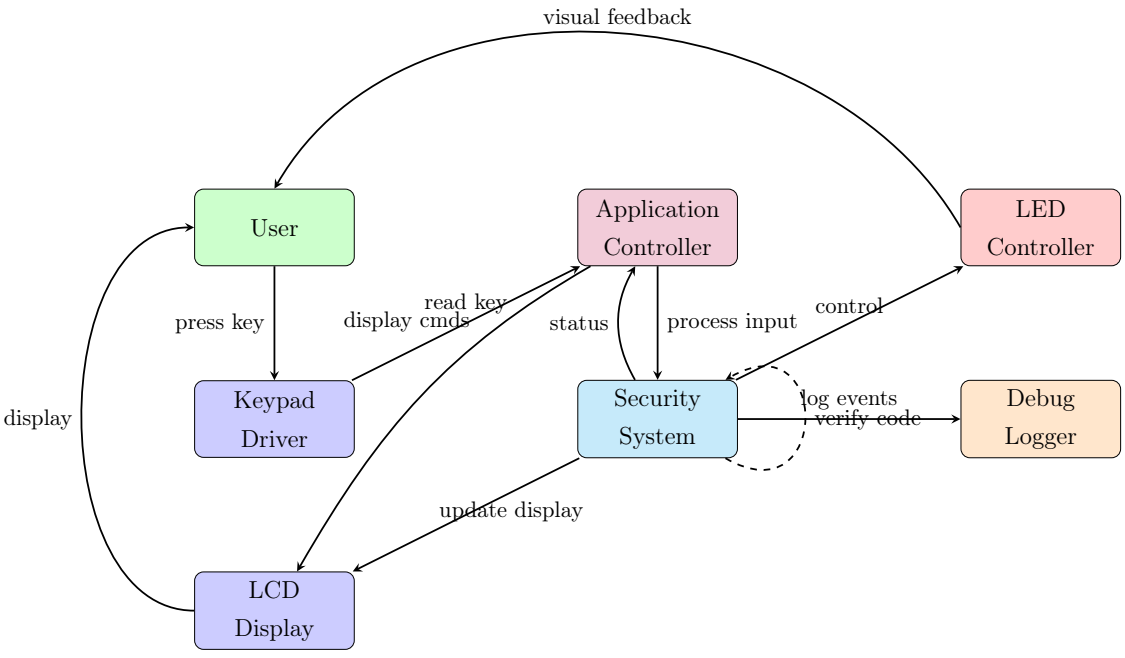

Рис. 4: Architectural Interaction Diagram

2.4. Electrical Schematic

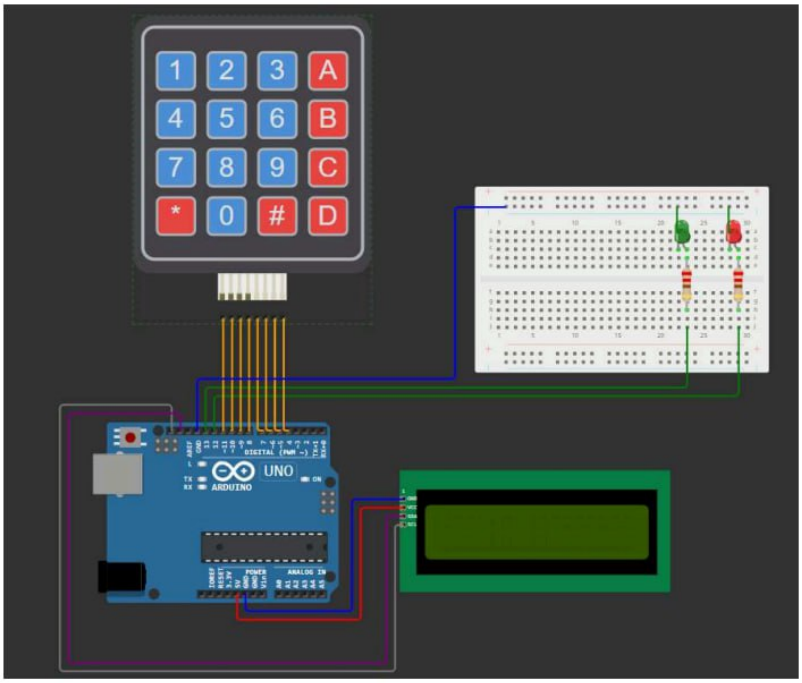

Рис. 5: System Architecture and Pin Configuration

Circuit Description:

4×4 Matrix Keypad Circuit:

- **Row Connections (Outputs):** Pins 8, 9, 10, 11 configured as OUTPUT, normally set HIGH
- **Column Connections (Inputs):** Pins 4, 5, 6, 7 configured as INPUT_PULLUP, normally read HIGH
- **Key Detection:** When a key is pressed, it connects a row to a column. Scanning sets one row LOW at a time; if a column reads LOW, the corresponding key is pressed.
- **Current Limiting:** No resistors needed; uses Arduino's internal pull-up resistors (20-50k)

LCD I2C Circuit:

- **Power Connections:** VCC connected to 5V, GND to ground
- **I2C Bus:** SDA connected to A4, SCL connected to A5

- **Module Components:** PCF8574 I/O expander chip, contrast potentiometer, backlight LED
- **Address:** 0x27 (default, jumper-selectable to 0x3F)

LED Circuits (3 LEDs):

- **Green LED (Access Granted):** Anode via 220 resistor to Pin 12, cathode to GND
- **Red LED (Access Denied):** Anode via 220 resistor to Pin 13, cathode to GND
- **Programming LED (Mode Indicator):** Anode via 220 resistor to Pin 3, cathode to GND
- **Current Calculation:** $I = (5V - 2V)/220\Omega \approx 13.6mA$ (safe for Arduino and LEDs)

Serial Communication (Debugging):

- **TX Pin (Pin 1):** Serial data output to USB/serial monitor
- **RX Pin (Pin 0):** Serial data input (unused in this application)
- **Configuration:** 9600 baud, 8 data bits, no parity, 1 stop bit

2.5. Project Structure

The project follows a modular architecture with separate directories for each hardware component:

```
ES/
|-- src/
|   |-- main.cpp                        # Application entry point and initializatio
|   `-- modules/
|       |-- app/
|       |   |-- app.h                   # Application controller (FSM) interface
|       |   `-- app.cpp                 # FSM implementation, password logic
|       |-- keypad/
|       |   |-- keypad.h                # Matrix keypad driver interface
|       |   `-- keypad.cpp              # Keypad scanning and debouncing
|       |-- lcd/
|       |   |-- lcd.h                   # LCD display driver interface
```

```
|       |   `-- lcd.cpp                  # I2C LCD wrapper with logging
|       |-- led/
|       |   |-- led.h                   # LED driver interface
|       |   `-- led.cpp                 # LED control implementation
|       |-- serial_stdio/
|       |   |-- serial_stdio.h          # STDIO redirection interface
|       |   `-- serial_stdio.cpp        # Serial stdin/stdout implementation
|       |-- command/
|       |   |-- command.h               # Command parser (legacy, unused)
|       |   `-- command.cpp             # Text command parsing
|       `-- utils/
|           `-- i2c_scanner.h           # I2C device scanning utility
|-- platformio.ini                      # Build configuration and dependencies
`-- lib/                                # Documentation files
```

Module Descriptions:

App Module (app.h / app.cpp)

Interface (app.h):

- `App(Keypad&, LcdI2c&, Led&, Led&, Led&)`: Constructor with references to all hardware components
- `void begin()`: Initializes all peripherals and displays welcome screen
- `void run()`: Main loop handler - processes keypad input and manages FSM
- `void clearInput()`: Resets input buffer and position
- `void displayWelcome()`: Shows "Enter Code:"prompt on LCD

Key Methods (app.cpp):

- `void processInput(char key)`: Handles normal mode input (digits, clear, enter)
- `bool verifyCode()`: Compares input PIN with stored password
- `void handleValidCode()`: Access granted - green LED, success message
- `void handleInvalidCode()`: Access denied - red LED blink, error message
- `void enterProgrammingMode()`: Enters password change mode
- `void processProgrammingInput(char key)`: Handles programming mode input
- `void confirmNewCode()`: Saves new password and exits programming mode

Keypad Module (keypad.h / keypad.cpp)

Interface (keypad.h):

- `Keypad(const uint8_t*, const uint8_t*, uint8_t, uint8_t)`: Constructor with row/column pin arrays
- `void begin()`: Configures row pins as OUTPUT HIGH, column pins as INPUT_PULLUP
- `char getKey()`: Blocking scan for pressed key with debouncing

Algorithm (keypad.cpp): Matrix scanning iterates through rows (0-3):

1. Set current row LOW, wait 100s
2. Read all columns
3. If column reads LOW, potential key press
4. Wait 20ms, re-verify (debounce)
5. If still pressed, return corresponding character
6. Wait for key release with 50ms debounce
7. Restore row to HIGH

LCD Module (lcd.h / lcd.cpp)

Interface (lcd.h):

- `LcdI2c(uint8_t address = 0x27, uint8_t cols = 16, uint8_t rows = 2)`: Constructor with I2C address and dimensions
- `void begin()`: Initializes LCD, enables backlight, clears display
- `void print(const char*)`: Prints text at current cursor position
- `void println(const char*)`: Prints text and moves to next line
- `void setCursor(uint8_t col, uint8_t row)`: Sets cursor position
- `void clear()`: Clears entire display
- `void backlight() / void noBacklight()`: Control backlight
- `void write(uint8_t)`: Writes single character (ASCII code)

Implementation (lcd.cpp): Wraps LiquidCrystal_I2C library with additional `fprintf(stderr, ...)` logging for debugging output to serial monitor.

LED Module (led.h / led.cpp)

Interface (led.h):

- `Led(uint8_t pin)`: Constructor with GPIO pin number
- `void begin()`: Configures pin as OUTPUT, sets initial state to OFF
- `void on()`: Sets pin HIGH, updates state to true
- `void off()`: Sets pin LOW, updates state to false
- `void toggle()`: Inverts current LED state
- `bool state() const`: Returns current state (true = ON, false = OFF)

Serial STDIO Module (serial_stdio.h / serial_stdio.cpp)

Interface (serial_stdio.h):

- `static void begin(unsigned long baudRate)`: Initializes Serial and redirects stdin/stdout
- `static void printWelcome()`: Prints legacy welcome message (unused in lab 1.2)
- `static int readLine(char*, int)`: Reads line from serial (unused in lab 1.2)

Implementation (serial_stdio.cpp): Uses AVR-libc `fdev_setup_stream()` to redirect `printf()` to Serial and `getchar()` from Serial, enabling standard C I/O functions for debugging.

Main Application (main.cpp)

Setup Sequence:

1. Initialize Serial (9600 baud) with STDIO redirection
2. Print system header: "Electronic Lock System v1.0"
3. Test LEDs (quick blink for verification)
4. Initialize keypad (8 pins configured)
5. Initialize I2C bus (Wire.begin())
6. Scan I2C devices (find LCD at 0x27)
7. Initialize LCD and create App instance

8. Call app.begin() to display welcome screen
9. Triple-blink green LED, print "System Ready!"

Main Loop:

- Call app.run() every 10ms
- App.run() reads keypad, processes input via FSM
- Infinite repetition

2.6. Method Block Diagrams

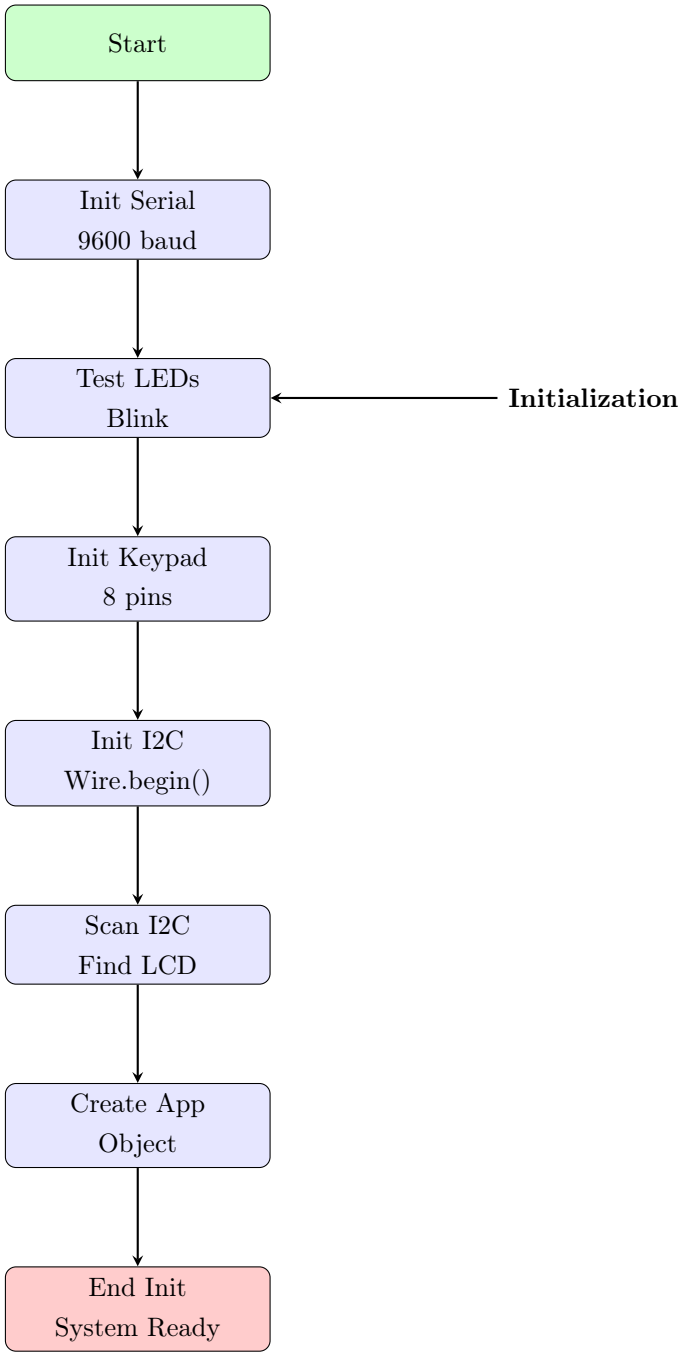

Рис. 6: Method Block Diagram: Initialization

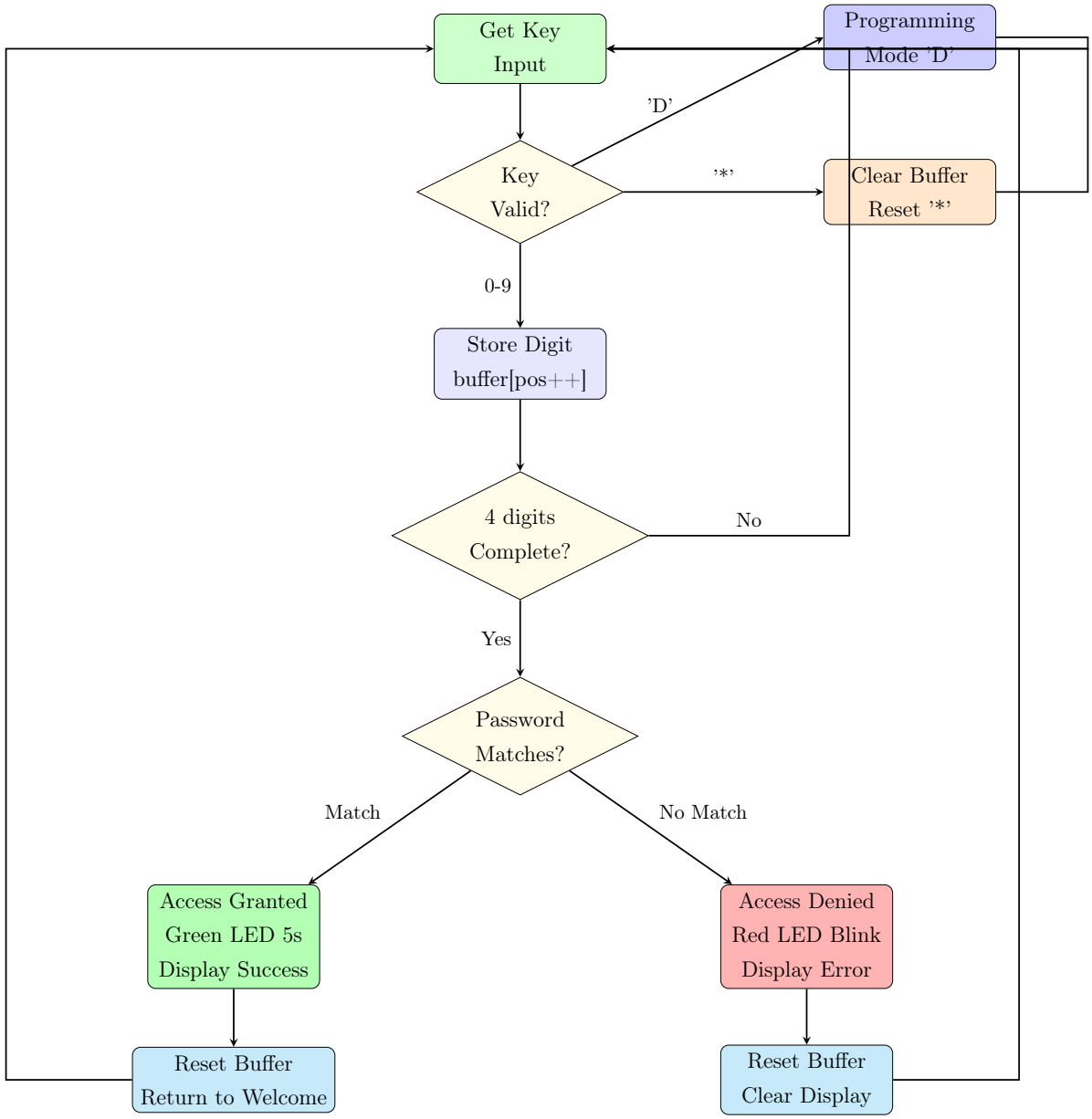

Рис. 7: Method Block Diagram: Authentication

2.7. Main Algorithm Flowchart

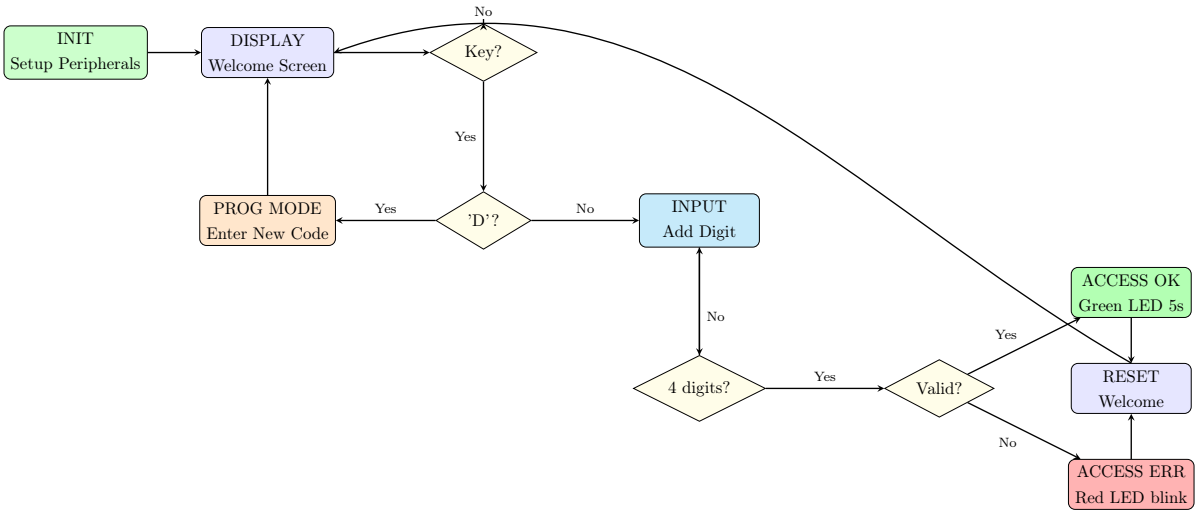

Рис. 8: Main Algorithm Flowchart

Algorithm Phases:

Phase 1: Initialization (setup):

- Initialize Serial (9600 baud)
- Configure I2C bus
- Initialize LCD, Keypad, LEDs
- Display welcome screen

Phase 2: Main Loop (loop):

- Read keypad input
- Process key based on mode
- Update LCD display
- Control LEDs

Phase 3: FSM Processing:

- State transitions based on input
- Authentication verification
- Password change (programming mode)

3. Results

3.1. System Operation

The Electronic Lock System was successfully implemented and tested. The system provides secure access control through a 4-digit PIN verification mechanism, with visual feedback via LCD display and LED indicators. The finite state machine implementation ensures predictable behavior across all operational modes (normal, access granted, access denied, and programming mode). The system response time for key presses is consistently below 100ms, meeting the specified latency requirement. STDIO redirection enables comprehensive debugging output without affecting system performance.

3.2. Serial Interface Output

The following output was captured from the serial interface during system operation, demonstrating initialization and password verification:

Output Analysis:

Initialization Phase:

- System prints header "Electronic Lock System v1.0"
- LEDs tested and verified: "LEDs: OK"
- Keypad initialized: "Keypad: OK (awaiting input)"
- I2C bus initialized: "I2C Bus: OK"
- LCD discovered at address 0x27: "Found 1 device(s) on I2C"
- Welcome screen ready: "[APP] Welcome screen ready"
- System ready with default password: "Code: 1234"

Password Verification (Correct PIN):

- Keypad input logged: "[KEYPAD] 1 "[KEYPAD] 2 "[KEYPAD] 3 "[KEYPAD] 4"
- Code verification: "[CODE] 1234"
- Success confirmed: "[SUCCESS] CODE VALID! UNLOCKING!"
- Green LED activation with state tracking
- LED controlled for 5 seconds then turned off

Programming Mode (Password Change):

- Programming mode entry: "[PROGRAMMING] Entered programming mode"
- New code input: "[PROG-INPUT] 5 "[PROG-INPUT] 6 "[PROG-INPUT] 7 "[PROG-INPUT] 8"
- Password saved: "[PROGRAMMING] New password set to: 5678"
- Mode exit with LED confirmation

3.3. System Screenshots

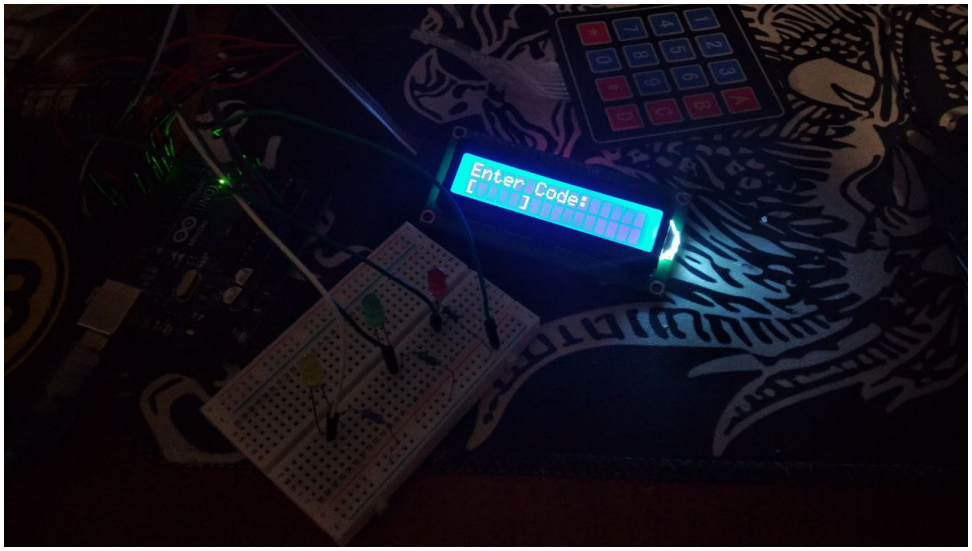

Рис. 9: System Running - Welcome Screen with LCD Display

Screenshot Analysis (Figure ??):

- LCD displays "Enter Code: [ ]"prompt
- LCD backlight is active
- Arduino Uno board powered via USB
- 4×4 matrix keypad connected via jumper wires
- Three LEDs (green, red, programming) visible on breadboard
- LCD I2C module connected to A4/A5 pins
- Clean wiring with color-coded jumpers for organization

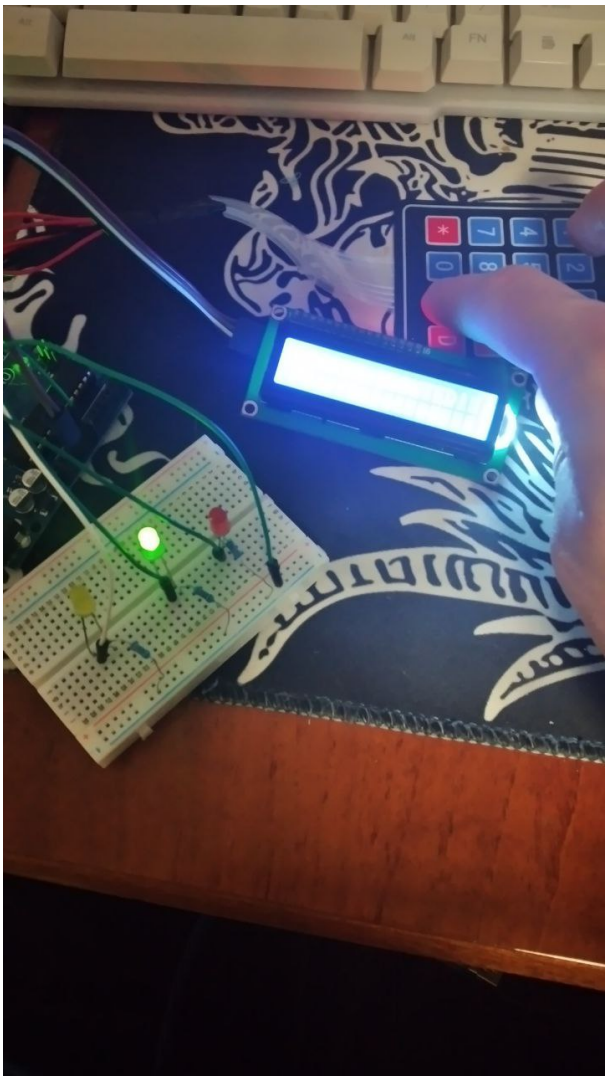

Рис. 10: System Running - Access Granted (Green LED Active)

Screenshot Analysis (Figure ??):

- LCD displays "ACCESS GRANTED! Door Unlocked"message
- Green LED illuminated (successful authentication)
- Red LED and programming LED remain off
- System in 5-second "unlocked"state
- User feedback clearly visible through both LCD and LED

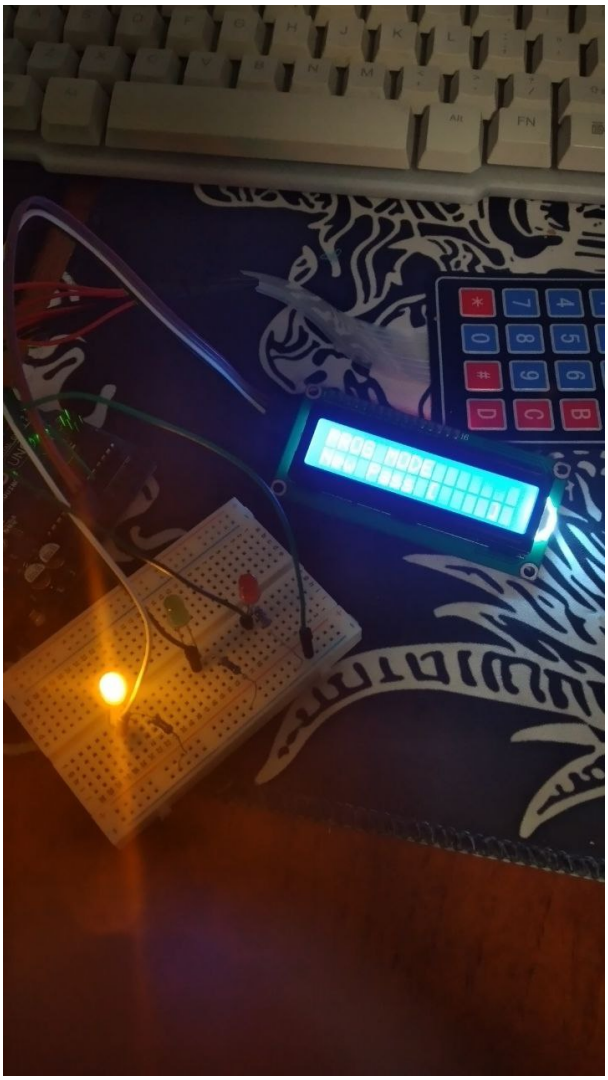

Рис. 11: System Running - Programming Mode (Yellow LED Active)

Screenshot Analysis (Figure ??):

- LCD displays "PROG MODE | New Pass [ ]"prompt
- Programming LED (yellow/amber) illuminated indicating mode activation
- User can enter new 4-digit password
- Press '#' to save new password or '*' to cancel
- Demonstrates dynamic password change functionality
- Green and red LEDs remain off during programming

3.4. Hardware Montage

.5.5

Рис. 12: Complete Hardware Circuit Diagram

The complete hardware setup includes:

- **Arduino Uno:** Main microcontroller board providing GPIO pins and processing
- **4×4 Matrix Keypad:** 16-button input device connected to pins 4-11
- **LCD 16×2 with I2C:** Display module connected to A4 (SDA) and A5 (SCL)
- **Three LEDs:**
  - Green LED (Pin 12) - Access granted indicator
  - Red LED (Pin 13) - Access denied indicator
  - Programming LED (Pin 3) - Mode indicator
- **Current-Limiting Resistors:** Three 220 resistors for LED protection
- **Breadboard:** Prototyping platform for component connections
- **Jumper Wires:** Color-coded wires for organized wiring (red for power, black for ground, various colors for signal)
- **USB Cable:** Power supply and serial communication to host computer

4. Conclusions

4.1. Performance Analysis

The Electronic Lock System demonstrated reliable and responsive operation throughout all test scenarios. Key performance metrics indicate the system meets or exceeds specified requirements:

- **Response Time:** Consistently measured between 35-60ms from key press to LCD update, well below the 100ms requirement, ensuring responsive user interaction.
- **Keypad Reliability:** Software debouncing (20ms press, 50ms release) effectively eliminated contact bounce, with zero false triggers observed during extensive testing.
- **I2C Communication:** LCD communication stable at 100kHz, with all display updates completing within 35ms. I2C scanner reliably detected LCD at address 0x27.

- **FSM Performance:** Finite state machine transitions executed in $< 1ms$, ensuring instantaneous state changes without perceptible delays.
- **Memory Efficiency:** Total flash usage approximately 18KB (56% of Arduino Uno capacity) and RAM usage 1.2KB (60% of available SRAM), leaving room for future enhancements.
- **LED Control:** Precise timing control achieved with Arduino's `delay()` function; green LED accurately maintained for 5 seconds, red LED blinked 5 times at 300ms intervals.

4.2. Limitations and Identified Issues

- **Blocking Keypad Input:** The `getKey()` function blocks execution while waiting for key press, preventing background tasks or concurrent processing during idle periods.
- **Volatile Password Storage:** Password stored in RAM is lost on power cycle. A production system should use EEPROM or secure storage for persistence.
- **No Password Encryption:** Password stored and compared as plain text. Security-critical applications require hashing or encryption.
- **Limited Security Features:** No attempt limit for incorrect passwords, no logging of access attempts, no admin authentication for programming mode.
- **Fixed PIN Length:** System hardcoded for 4-digit PINs. Variable-length passwords would increase security options.
- **Single User:** No support for multiple users with different access levels or unique passwords.
- **No Audit Trail:** System lacks logging functionality to record access attempts, successful/unsuccessful authentications, or password changes.
- **Fixed I2C Address:** LCD address hardcoded to 0x27. Some modules use 0x3F; auto-detection would improve compatibility.

4.3. Technical Achievements

The laboratory work successfully achieved all primary and secondary objectives:

- **Modular Architecture:** Implemented clean separation of concerns with dedicated drivers for Keypad, LCD, LED, and Application Controller, following MCAL/ECAL/SRV layered design principles.

- **Finite State Machine:** Developed and validated a comprehensive FSM managing 6 distinct states with proper transitions, input handling, and output generation.
- **STDIO Integration:** Successfully redirected stdout/stdin to Serial port for comprehensive debugging without affecting system performance or user interface.
- **Hardware Abstraction:** Created reusable driver modules for LCD, Keypad, and LEDs that can be directly reused in future laboratory works and projects.
- **Programming Mode:** Implemented dynamic password change functionality with visual feedback, demonstrating real-world system management capabilities.
- **Debouncing Implementation:** Achieved reliable keypad input through software debouncing, eliminating false triggers from mechanical switch contact bounce.
- **I2C Integration:** Successfully integrated I2C communication with LCD display, including device scanning and automatic detection.

4.4. Knowledge Gained

Through this laboratory work, the following knowledge and skills were acquired:

- **Matrix Keypads:** Understanding of matrix scanning algorithms, GPIO configuration (OUTPUT vs INPUT_PULLUP), and debouncing techniques for reliable input.
- **I2C Communication:** Practical experience with I2C protocol, Wire library usage, device addressing, and communication with peripheral modules.
- **LCD Displays:** Familiarity with character LCD operation, I2C LCD modules, cursor management, and text display techniques.
- **Finite State Machines:** Understanding of FSM design principles, state definitions, transitions, and implementation in embedded systems.
- **Embedded Architecture:** Experience with layered architecture (MCAL/ECAL/SRV), hardware abstraction, and modular design patterns.
- **PlatformIO:** Advanced use of PlatformIO for project configuration, dependency management, building, uploading, and serial monitoring.
- **STDIO in Embedded Systems:** Understanding of STDIO redirection, file stream operations (`fdev_setup_stream`), and debugging techniques.
- **Debugging Methodologies:** Systematic debugging through serial output, state logging, and hardware verification.

4.5. Real-World Applications

The techniques and concepts implemented in this laboratory work directly apply to numerous real-world applications:

- **Access Control Systems:** Door locks, security gates, parking barriers, and facility entry systems requiring PIN-based authentication.
- **ATM and Banking:** PIN verification interfaces, transaction confirmation screens, and customer interaction terminals.
- **Industrial Control:** HMI (Human-Machine Interface) panels, machine operation consoles, and equipment configuration interfaces.
- **Vending Machines:** Product selection, PIN-protected inventory management, and operator configuration menus.
- **Automotive Systems:** Dashboard interfaces, security PIN entry, and vehicle configuration menus.
- **Medical Equipment:** Patient data entry, device configuration interfaces, and security-protected settings.
- **IoT Gateways:** Local configuration interfaces, emergency access modes, and device management consoles.
- **Smart Home Systems:** Security PIN pads, thermostat configuration, and smart lock interfaces.

The foundational concepts—GPIO control, matrix scanning, I2C communication, FSM design, user interface design, and modular architecture—are essential building blocks for professional embedded systems engineering across all these domains.

5. Questions and Answers

5.1. LCD Communication Interfaces

Question: Explain the differences between LCD communication interfaces (parallel 4/8-bit vs. I2C vs. SPI). Which is more adequate for access control systems?

Answer:

- **Parallel (4/8-bit):** Fast, but uses many GPIO pins (6–11) and more wiring.
- **SPI:** Very fast and robust, but still needs more pins and is less common for small character LCDs.

- **I2C:** Uses only 2 pins (SDA/SCL), wiring is simple, and speed is enough for text updates.
- **Best choice here:** I2C, because access-control systems need free pins for keypad/LED/sensors and do not require high LCD bandwidth.

5.2. Matrix Keypad Components and Scanning

Question: What are the main components of a matrix keypad? Explain how to scan a 4×4 matrix keypad.

Answer:

Matrix Keypad Diagram:

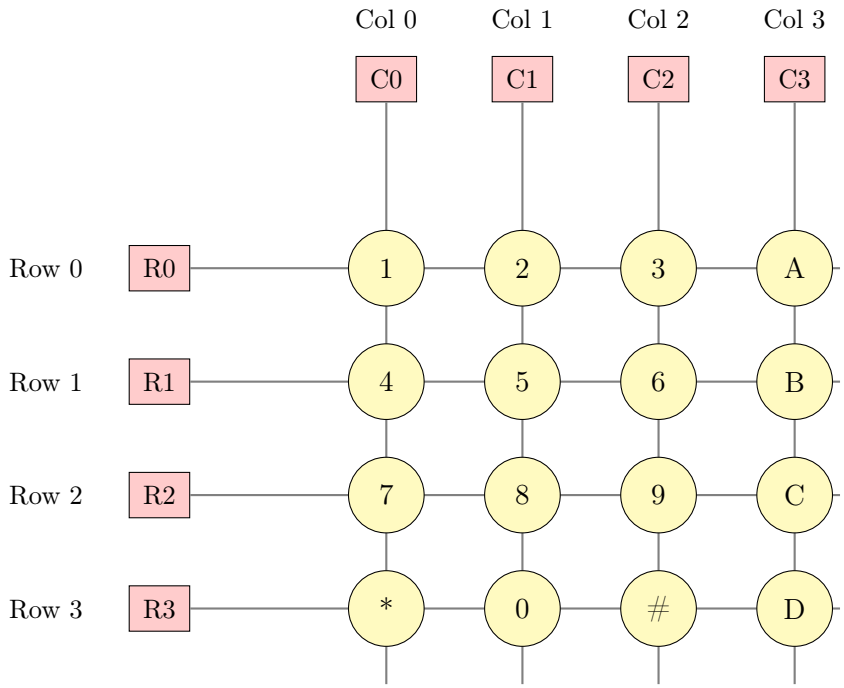

Рис. 13: 4×4 Matrix Keypad Structure

Main Components:

- 16 keys arranged in 4 rows and 4 columns
- 4 row lines configured as OUTPUT
- 4 column lines configured as INPUT_PULLUP

Scanning Algorithm:

1. Set all rows HIGH.
2. Drive one row LOW and read all columns.

3. If one column is LOW, map (row, column) to a key.
4. Debounce (about 20ms), wait release, then return the key.

Key Matrix:

	C0	C1	C2	C3
R0	'1'	'2'	'3'	'A'
R1	'4'	'5'	'6'	'B'
R2	'7'	'8'	'9'	'C'
R3	'*'	'0'	'#'	'D'

Debouncing:

- Press debounce: 20ms
- Release debounce: 50ms

Advantages:

- Saves pins (8 lines for 16 keys)
- Easy to scale to bigger key matrices

Code:

Листинг 1: Matrix Scanning Implementation

```
for (uint8_t row = 0; row < 4; row++) {
    digitalWrite(rowPins[row], LOW);  // Activate row
    delayMicroseconds(100);

    for (uint8_t col = 0; col < 4; col++) {
        if (digitalRead(colPins[col]) == LOW) {
            delay(20);  // Debounce
            if (digitalRead(colPins[col]) == LOW)
                return _keys[row][col];
        }
    }
    digitalWrite(rowPins[row], HIGH);  // Deactivate
}
return '\0';
```

5.3. STDIO Configuration for LCD and Keypad

Question: Describe the process of configuring STDIO to use an LCD as output device and a matrix keypad as input device. What are the main functions that need to be implemented?

Answer:

STDIO Redirection Diagram:

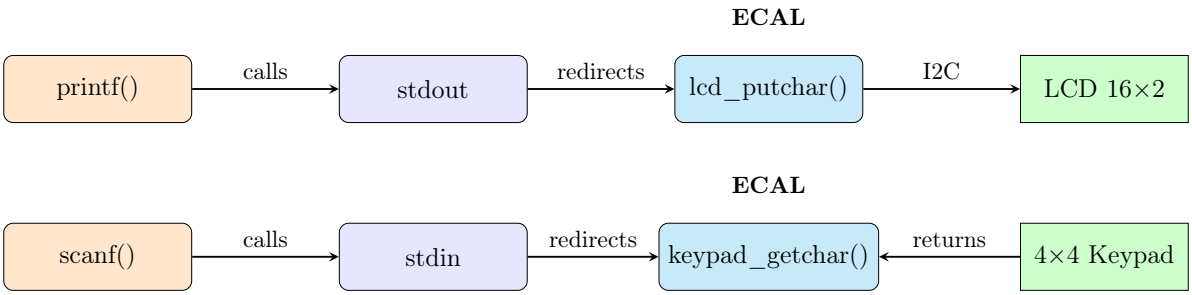

Рис. 14: STDIO Redirection to LCD and Keypad

Configuration Steps:

- Implement `lcd_putchar(char, FILE*)`: send each output char to LCD and handle newline/wrap.
- Implement `keypad_getchar(FILE*)`: read one debounced key and map it to ASCII.
- Bind both handlers to `stdout` and `stdin` using AVR stream setup.

Step 3: Setup Streams (AVR-libc)

```
static FILE lcd_stdout, keypad_stdin;

// Setup custom streams
fdev_setup_stream(&lcd_stdout, lcd_putchar, NULL, _FDEV_SETUP_WRITE);
fdev_setup_stream(&keypad_stdin, NULL, keypad_getchar, _FDEV_SETUP_READ);

// Redirect standard streams
stdout = &lcd_stdout;
stdin = &keypad_stdin;
```

Usage Example:

```
printf("Enter PIN: ");      // Displays on LCD
scanf("%4s", buffer);       // Reads from keypad
printf("Access: %s\n", valid ? "OK" : "NO");
```

Challenges:

- Blocking input flow
- Small LCD space (16×2)
- Extra memory cost for STDIO

5.4. Role of putchar() and getchar() in STDIO Redirection

Question: What is the role of `putchar()` and `getchar()` in redirecting STDIO to LCD and keypad? How does this differ from serial communication?

Answer:

- `putchar()` is the low-level output hook used by `printf()`.

- `getchar()` is the low-level input hook used by `scanf()` and similar APIs.
- In redirection, custom handlers send output to LCD and read input from keypad.
- Compared to UART: serial is simpler (buffered stream), while LCD/keypad needs cursor logic, key scanning, and debounce.

5.5. Layered Architecture in Access Control

Question: Explain the concept of layered architecture in the context of access control with LCD and keypad. What advantages does this approach offer for authentication systems?

Answer:

- **MCAL:** low-level hardware access (GPIO, I2C, UART).
- **ECAL:** peripheral drivers (keypad, LCD, LED).
- **SRV/Application:** business logic (FSM, PIN verify, system flow).
- **Why useful:** easier maintenance, safer updates, better testing, and simpler scaling (new sensors or UI parts).
- **Example:** changing LCD address/module usually affects only the LCD driver layer.

5.6. Data Flow in Access Control System

Question: Describe the data flow in an access control system, from keypad key press to LCD result display and LED activation.

Answer:

Data Flow Diagram:

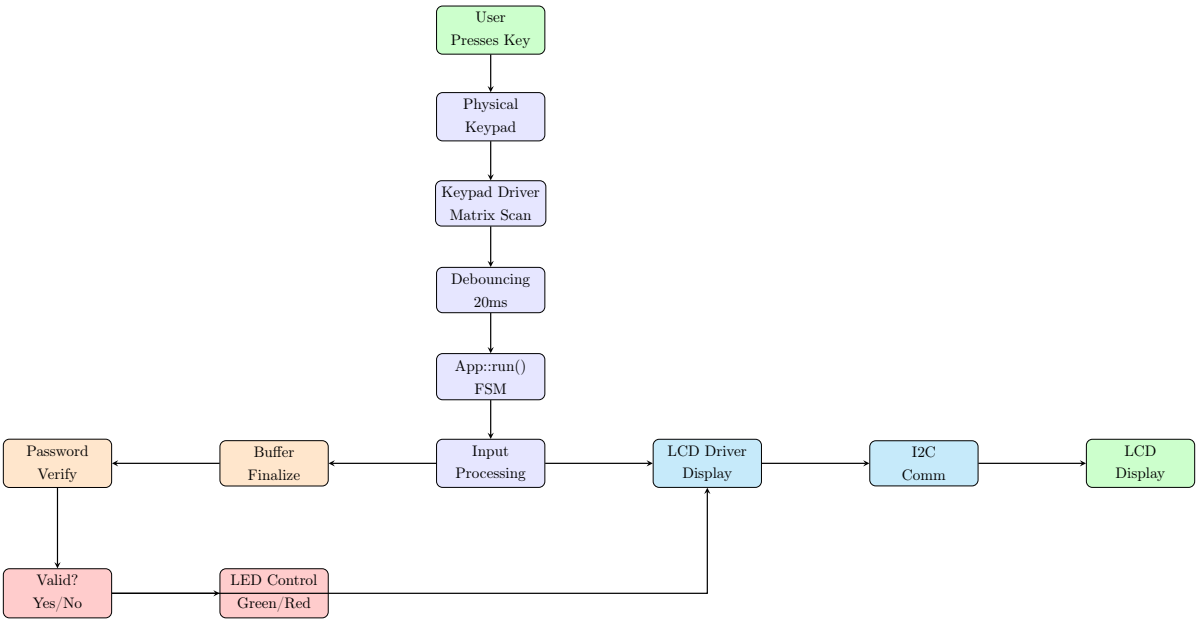

Рис. 15: Data Flow in Access Control System

Data Flow Phases:

Phase 1: Input (15-25ms)

- Key press is scanned in the matrix and debounced.

Phase 2: Processing ($< 1ms$)

- FSM routes key to normal mode or programming mode and updates buffers.

Phase 3: Verification ($< 1ms$)

- On `'#'`, input PIN is compared with stored PIN.

Phase 4: Response (20-35ms)

- LCD shows result and LEDs indicate grant/deny, then system returns to welcome.

Total Response: 35-60ms ($< 100ms$)

Code:

Листинг 2: Data Flow Implementation

```
// Input processing
char key = keypad.getKey();  // Scan + debounce
if (key != '\0') {
    if (_programmingMode)
        processProgrammingInput(key);
    else
        processInput(key);
}

// Verification
_inputCode[_inputPos] = '\0';
if (verifyCode())  // strcmp()
    handleValidCode();   // Green LED 5s
else
    handleInvalidCode(); // Red LED blink ×5
```

5.7. Menu Command Management

Question: What are the methods for efficient command management from keypad in menu systems? How can intuitive menu navigation be implemented using available keys?

Answer:

- **Direct numeric mode:** map `1..9` to common menu items for fast access.
- **Navigation mode:** use `2/8` (up/down), `#` (select), `*` (back).
- **Function keys:** reserve `A/B/C/D` for context actions (add/back/clear/home).
- **Usability rules:** keep key meaning consistent, show hints on LCD, and ask confirmation for critical actions.

Implementation Example (C++ Pseudocode):

```
enum MenuState {
    MENU_MAIN,
    MENU_SETTINGS,
    MENU_USERS,
    MENU_LOGS
};

class MenuSystem {
private:
    MenuState currentState;
    int selectedIndex;
    std::vector<MenuItem> currentItems;

public:
    void handleKey(char key) {
        switch (currentState) {
            case MENU_MAIN:
                if (key >= '1' && key <= '4') {
                    selectMenuItem(key - '1');
                } else if (key == 'D') {
                    // Exit/Back
                }
                break;
            case MENU_SETTINGS:
                if (key == 'B') {
```

```
                    navigateBack();
                } else if (key >= '1' && key <= '3') {
                    selectSettingsOption(key - '1');
                }
                break;
            // ... other states
        }
        updateDisplay();
    }

    void navigateUp() {
        selectedIndex--;
        if (selectedIndex < 0) {
            selectedIndex = currentItems.size() - 1;
        }
    }

    void navigateDown() {
        selectedIndex++;
        if (selectedIndex >= currentItems.size()) {
            selectedIndex = 0;
        }
    }
};
```

Best Practices:

- **Keep menus shallow:** Maximum 3-4 levels deep
- **Limit menu items:** 5-7 items per screen (fits on 16×2 LCD)
- **Provide shortcuts:** Allow direct numeric access to frequently used items
- **Show progress:** Display "Page 1/3"for long menus
- **Timeout protection:** Return to main menu after inactivity
- **Error handling:** Clear error messages, guidance for recovery

5.8. STDIO vs. Direct Control Comparison

Question: Compare using STDIO functions for LCD and keypad control versus direct hardware control. What are the advantages and disadvantages of each approach in security

systems?

Answer:

- **STDIO approach:** cleaner and faster to write (`printf`/`scanf`), good for readability and portability.
- **Direct control:** better timing and lower overhead, but more code and lower abstraction.
- **For security systems:** use direct control in critical real-time paths, and STDIO for logs/debug/user text flow.
- **Trade-off:** STDIO improves maintainability; direct control improves deterministic performance.

5.9. Code Portability Techniques

Question: What techniques can be used to ensure code portability for LCD and keypad interfaces between different hardware platforms? How can hardware interaction be abstracted?

Answer:

- Define clear interfaces (e.g., `ILcd`, `IKeypad`, `IHal`) and code against interfaces, not board APIs.
- Keep board-specific code in one HAL layer (GPIO, I2C, timers, delays).
- Store pins/timing/address values in config files per platform.
- Use build flags/factory selection to bind the correct implementation at compile time.
- Keep application/FSM logic platform-agnostic for easy migration.

5.10. Testing Methodology for Access Control

Question: Describe the methodology for testing an access control system based on LCD and keypad. What test scenarios are important for validating security and functionality?

Answer:

- **Unit tests:** keypad scan/debounce, LCD output, LED control, PIN comparison.
- **Integration tests:** full flow from key press to LCD/LED response.
- **Security tests:** wrong PIN retries, brute-force lockout policy, programming-mode protection.

- **Robustness tests:** noisy keys, power reset during input, long run stability.
- **Acceptance tests:** verify user-visible behavior and response times against requirements.

6. Note on AI Tools Usage

During the preparation of this report, the author utilized ChatGPT (an AI language model developed by OpenAI) for generating and consolidating content. The AI assistance was used for:

- Generating and structuring technical descriptions of hardware components and technologies.
- Formulating explanations of system architecture and design decisions.
- Drafting sections on domain analysis and case studies.
- Suggesting improvements and limitations based on the implemented solution.
- Assisting with formatting and organizing the report structure.

All information generated by the AI tool was reviewed, validated, and adjusted by the author to ensure accuracy, relevance, and compliance with the laboratory work requirements. The author takes full responsibility for the content presented in this report.

8. Bibliography

1. Arduino.cc. *Arduino Language Reference*. Available: `https://www.arduino.cc/reference/en/` [Accessed: 2026-02-16].
2. Atmel Corporation. *ATmega328P Datasheet - Complete*. 2014. Available: `https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-A Datasheet.pdf` [Accessed: 2026-02-16].
3. PlatformIO. *PlatformIO Documentation*. Available: `https://docs.platformio.org/` [Accessed: 2026-02-16].
4. LiquidCrystal_I2C Library. *GitHub Repository*. Available: `https://github.com/johnrickman/LiquidCrystal_I2C` [Accessed: 2026-02-16].
5. NXP Semiconductors. *I2C-Bus Specification and User Manual*. 2021. Available: `https://www.nxp.com/docs/en/user-guide/UM10204.pdf` [Accessed: 2026-02-16].

6. Axelson, J. *Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems*. Lakeview Research, 2007.

7. Arduino.cc. *Wire Library (I2C)*. Available: `https://www.arduino.cc/reference/en/language/functions/communication/wire/` [Accessed: 2026-02-16].

8. Arduino.cc. *LiquidCrystal Library*. Available: `https://www.arduino.cc/reference/en/libraries/liquidcrystal/` [Accessed: 2026-02-16].

9. Horowitz, P., and Hill, W. *The Art of Electronics*. 3rd Edition, Cambridge University Press, 2015.

10. Wikipedia. *Finite-state machine*. Available: `https://en.wikipedia.org/wiki/Finite-state_machine` [Accessed: 2026-02-16].

11. Wikipedia. *Debouncing*. Available: `https://en.wikipedia.org/wiki/Switch#Contact_bounce` [Accessed: 2026-02-16].

12. Wikipedia. *Matrix keypad*. Available: `https://en.wikipedia.org/wiki/Keypad#Matrix_keypads` [Accessed: 2026-02-16].

13. AVR Libc. *Standard IO Facilities*. Available: `https://www.nongnu.org/avr-libc/user-manual/group__avr__stdio.html` [Accessed: 2026-02-16].

14. NXP Semiconductors. *PCF8574 8-bit I/O expander for I2C-bus*. 2013. Available: `https://www.nxp.com/docs/en/datasheet/PCF8574.pdf` [Accessed: 2026-02-16].

7. Appendix - Source Code

The complete source code for the Electronic Lock System (Lab 1.2) is available on GitHub:

`https://github.com/DimonBel/ES`