Arduino-Based Scientific Calculator with LCD Display

Sai Akhila Reddy Turpu - EE24BTECH11055

I. Introduction

This project implements a scientific calculator using an Arduino Uno microcontroller and an LCD display. The system provides advanced mathematical functions including trigonometric operations, logarithms, and basic arithmetic, demonstrating the integration of microcontroller programming with user interface design.

II. Hardware Components

1) Arduino Uno
2) Push buttons
3) $15k\Omega$ and $1k\Omega$ resistors
4) Jumper wires and Conducting wires
5) LCD Display

III. Circuit Design

The hardware implementation uses a combination of direct pin connections and analog multiplexing:

A. *Display Interface*

The LCD module connects using a 4-bit parallel interface to conserve GPIO pins:

- Control lines: RS (Register Select) and E (Enable)
- Data lines: D4-D7 for nibble transfer
- Contrast adjustment via potentiometer

TABLE I: LCD Connections

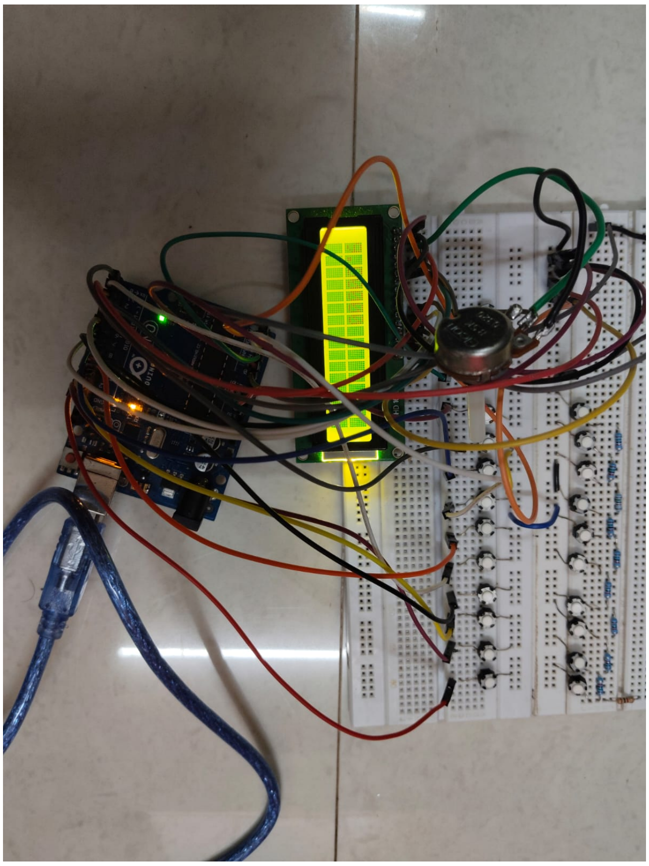
LCD Pin	Connection
1	Ground
2	5V
3	Ground via 1.5kΩ resistor
4	Arduino D2
5	Ground
6	Arduino D3
11	Arduino D4
12	Arduino D5
13	Arduino D6
14	Arduino D7
15	5V via 1kΩ resistor
16	Ground

TABLE II: Digital Pin Connections

Arduino Pin	Connection
D0	Button 16
D1	Button 17
D8	Button 18
D9	Button 19
D10	Button 20
D11	Button 21
D12	Button 22
D13	Button 23

TABLE III: Analog Pin Connections

Arduino Pin	Connection
A0	Buttons 1-10 (Digits)
A1	Button 15
A2	Button 14
A3	Button 13
A4	Button 12
A5	Button 11



IV. Analog-to-Digital Conversion (ADC)

A. ADC Fundamentals

The ATmega328P's 10-bit ADC converts analog voltages (0-5V) to digital values (0-1023). Key characteristics include:

- 10-bit resolution (1024 discrete values)
- 9-260μs conversion time (depending on clock prescaler)
- 8 multiplexed input channels (A0-A7)

B. Implementation in Calculator

- **Button:**
  - Digit buttons 0-9 connected via voltage divider to A0
  - Each button produces a unique voltage level (e.g., Button1=0.5V, Button2=1.0V,...,Button9=4.5V)
- **Configuration:**
  - Reference voltage: AVcc (5V)
  - Prescaler: 128 (ADC clock = 125kHz)
  - Channel selection: ADC0 (A0 pin)
- **Reading Process:**
  - Single conversion initiated via ADSC bit
  - `Conversion complete` flag (ADIF) checked for completion
  - Result read from ADC/ADCL registers

V. Mathematical implementation

A. Arithmetic Operations

- Four basic operations with floating-point precision
- Operator precedence handling (PEDMAS)
- Chained calculation capability

B. Scientific Functions

- Trigonometric functions (sin, cos, tan) with degree/radian modes
- Logarithmic functions (base 10 and natural log)
- Exponential functions

VI. Software Implementation

The main.c code can be found in the folder named 'codes'.

The following describes the key functions used in the implementation:

A. Core mathematical functions

1) Angle Conversion and Reduction:

- `double reduce_angle(double rad)`: Reduces any angle to the range $[0, 2\pi)$ by removing full rotations. Essential for trigonometric function stability.
- `double deg2rad(double deg)`: Converts degrees to radians using the formula $rad = deg \times \frac{\pi}{180}$.

2) Computing $\ln x$*:* `double compute_ln(double x)`: Calculates natural logarithm using numerical integration of $\int_1^x \frac{1}{t} dt$ with trapezoidal rule.

3) Numerical Methods:

- `double tangent_rk4(double radians, double h)`: Computes tangent using 4th-order Runge-Kutta method by solving the differential equation $\frac{d^2y}{dx^2} = -y$.
- `double power(double x, double n)`: Implements x^n using Euler's method on the differential equation $\frac{dy}{dx} = n \cdot y$.

B. Stack Operations

1) Operator Stack:

- `void initStack(Stack *s)`: Initializes operator stack.
- `void push(Stack *s, const char* val)`: Pushes operator onto stack.
- `const char* peek(Stack *s)`: Views top element without removal.

2) Number Stack:

- `void initNumStack(NumStack *s)`: Initializes operand stack.
- `void pushNum(NumStack *s, float val)`: Pushes number onto stack.
- `float popNum(NumStack *s)`: Removes and returns top number.

C. Expression Processing

1) Infix to Postfix Conversion:

- `void infixToPostfix(const char* infix, char* postfix)`: Converts standard mathematical notation to Reverse Polish Notation using Dijkstra's shunting-yard algorithm. Handles:
  - Parentheses grouping
  - Operator precedence (PEMDAS)
  - Special functions (trig, log)

2) Postfix Evaluation:

- `float evaluatePostfix(const char* postfix)`: Evaluates RPN expressions using a number stack. Supports:
  - Basic arithmetic $(+, -, \times, \div)$
  - Exponents (x^y)
  - Trigonometric functions $(\sin, \cos, \tan)$
  - Logarithmic functions $(\ln, \log_{10})$

D. Utility Functions

1) Fraction Conversion:

- `void decimal_to_fraction(double decimal, int *numerator, int *denominator)`: Converts floating-point numbers to simplified fractions using continued fractions approximation with tolerance 10^{-6}.

2) LCD Interface:

- `void lcd_init(void)`: Initializes 16x2 LCD in 4-bit mode with proper timing delays.
- `void lcd_print(const char* str)`: Outputs strings with character-by-character timing control.
- `void lcd_set_cursor(uint8_t col, uint8_t row)`: Positions cursor using DDRAM address mapping.

3) ADC Handling:

- `uint16_t adc_read(uint8_t channel)`: Reads analog inputs with:
  - 3-sample moving average
  - Hysteresis filtering (± 25 counts)
  - Channel auto-selection

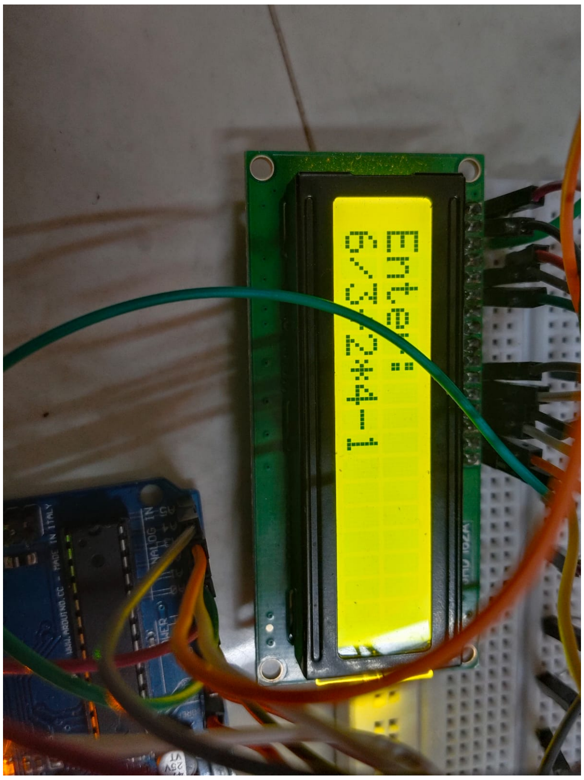
E. Main Program Flow

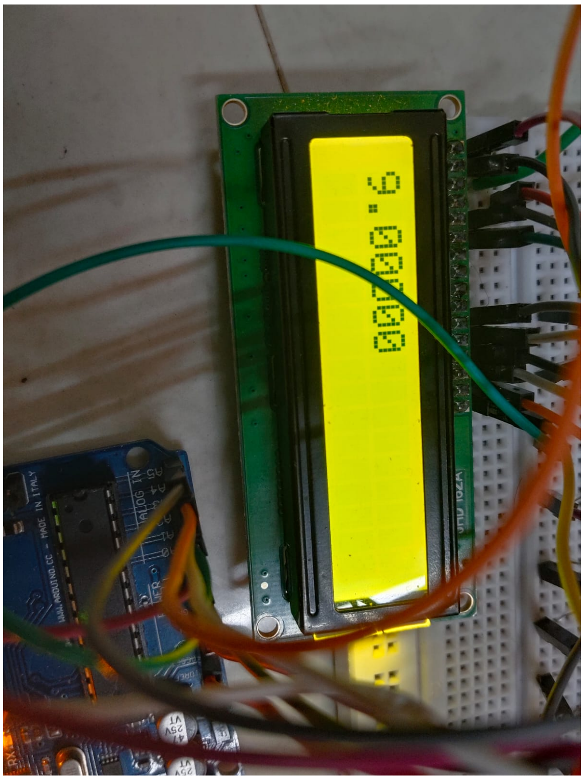
The calculator implements an event loop that:

1) Scans buttons via ADC and digital inputs
2) Processes keypresses with debouncing
3) Maintains input buffer (16-character limit)
4) Converts and evaluates expressions on ENTER
5) Displays results in decimal or fraction form

VII. Results and Discussion

The calculator worked correctly when pressing buttons and displayed the right numbers. All calculations were accurate during testing. The system ran smoothly without any problems for continuous operation and implemented PEDMAS effectively.
Example:(Implementing PEDMAS)





VIII. Conclusion

This project successfully demonstrates the implementation of a scientific calculator using embedded systems principles. The design makes efficient use of available microcontroller resources while providing comprehensive mathematical functionality.