Arduino Scientific Calculator: Implementation and Analysis

March 24, 2025

Abstract

This report provides a comprehensive explanation of an Arduino-based scientific calculator implementation. The calculator supports advanced mathematical operations through a Shunting Yard algorithm for expression parsing, along with custom mathematical functions. We detail both the electronic circuit design and the software architecture, focusing on the expression evaluation process and user interface implementation.

Contents

1 Introduction

The Arduino scientific calculator project implements a fully functional calculator capable of evaluating complex mathematical expressions with proper operator precedence. Key features include:

- Support for basic arithmetic operations (+, -, *, /)
- Advanced mathematical functions (sin, cos, tan, inverse trigonometric functions)
- Logarithmic functions (ln, log)
- Exponential operations (eˆx, power function)
- Parenthetical expressions with proper precedence handling
- Backspace functionality for error correction
- LCD display for input and result visualization

The calculator employs the Shunting Yard algorithm for expression evaluation and custom implementations of mathematical functions using Euler's method, without relying on external libraries.

2 Hardware Components and Circuit Design

2.1 Components List

- Arduino Uno R3 (ATmega328P microcontroller)
- 16x2 LCD display
- 8 push buttons for input
- Resistors (10kΩ for pull-up)
- Potentiometer (10kΩ for LCD contrast)
- Breadboard and connecting wires

2.2 Circuit Connections

The circuit follows a standard design with the following connections:

2.2.1 LCD Display Connection

The 16x2 LCD display is connected in 4-bit mode to save pins:

- RS (Register Select) - Arduino pin 12 (PB4)
- E (Enable) - Arduino pin 11 (PB3)
- D4-D7 - Arduino pins 5-2 (PD5-PD2)
- LCD VSS - GND
- LCD VDD - 5V
- LCD V0 - Connected to potentiometer for contrast adjustment

2.2.2 Button Connections

Eight push buttons are connected to Arduino inputs with pull-up resistors:

- DIGIT_BTN - Arduino pin A0 (PC0): Cycles through digits 0-9
- OPEN_PAREN - Arduino pin A1 (PC1): Adds opening parenthesis
- CLOSE_PAREN - Arduino pin A2 (PC2): Adds closing parenthesis
- OP_BTN - Arduino pin A3 (PC3): Cycles through operations

- FUNC_BTN - Arduino pin A4 (PC4): Cycles through functions
- SET_BTN - Arduino pin 7 (PD7): Commits selection to expression
- EQUALS_BTN - Arduino pin 8 (PB0): Evaluates expression
- CLEAR_BTN - Arduino pin 9 (PB1): Backspace functionality

3 Software Architecture

3.1 Code Organization

The software is structured into several logical components:

- **Hardware Initialization**: Setup of LCD, timer, and input buttons
- **User Input Handling**: Button detection and debouncing
- **Expression Building**: Construction of mathematical expressions
- **Expression Evaluation**: Shunting Yard algorithm implementation
- **Mathematical Functions**: Custom implementations of trigonometric, logarithmic, and exponential functions
- **Display Management**: LCD update and formatting

3.2 Key Data Structures

- **Expression String**: Stores the mathematical expression being built
- **Operator Stack**: Used in the Shunting Yard algorithm for operation precedence
- **Value Stack**: Stores operands and intermediate results during evaluation

4 Expression Evaluation: The Shunting Yard Algorithm

The calculator implements Dijkstra's Shunting Yard algorithm to convert infix expressions (normal mathematical notation) to postfix notation (Reverse Polish Notation) for evaluation while respecting operator precedence.

4.1 Algorithm Overview

The Shunting Yard algorithm processes an infix expression from left to right, using two stacks:

1. An operator stack for operators and functions
2. A value stack for numbers and calculation results

4.2 Operator Precedence

The calculator defines precedence levels for operations:

- Level 1: Addition (+) and subtraction (-)
- Level 2: Multiplication (*) and division (/)
- Level 3: Power ($\hat{}$)
- Level 4: Functions (sin, cos, tan, ln, etc.)

4.3 Implementation

The algorithm implementation handles:

- Tokenization of the input expression
- Operator precedence and associativity
- Function application
- Parenthetical expressions
- Error handling (division by zero, etc.)

```c
// Evaluate expression with proper operator precedence
float evaluate_expression(void) {
    // Copy expression so we can modify it
    char expr_copy[MAX_EXPR_LEN];
    strcpy(expr_copy, expression);

    // Stacks for Shunting Yard algorithm
    float value_stack[MAX_EXPR_LEN];
    int value_stack_top = 0;

    char operator_stack[MAX_EXPR_LEN];
    int operator_stack_top = 0;

    // Parse the expression
    int i = 0;
    while (i < strlen(expr_copy)) {
        // Algorithm implementation...
    }

    // Process remaining operators
    while (operator_stack_top > 0) {
        // Apply remaining operators...
    }

    // Final result on top of value stack
    return (value_stack_top > 0) ? value_stack[0] : 0.0f
        ;
}
```

5 Mathematical Functions Implementation

The calculator implements various mathematical functions without using external libraries, primarily using Euler's method for differential equations.

5.1 Trigonometric Functions

Sine and cosine are implemented using Euler's method to solve the system of differential equations:

$$\frac{d\sin(x)}{dx} = \cos(x) \tag{1}$$

$$\frac{d\cos(x)}{dx} = -\sin(x) \tag{2}$$

Starting with initial conditions $\sin(0) = 0$ and $\cos(0) = 1$, the functions are numerically integrated.

```c
float compute_sin_cos(float x, float* sin_val, float*
   cos_val) {
    float s = 0.0f;  // sin(0)
    float c = 1.0f;  // cos(0)
    float h = 0.0001f; // step size

    // Normalize x to [0, 2\pi)
    while(x >= 2*PI) x -= 2*PI;
    while(x < 0) x += 2*PI;

    for (float t = 0.0f; t < x; t += h) {
        float s_new = s + h * c;
        float c_new = c - h * s;
        s = s_new;
        c = c_new;
    }

    *sin_val = s;
    *cos_val = c;
    return s;
}
```

5.2 Exponential Function

The exponential function e^x is implemented by solving the differential equation:

$$\frac{dy}{dx} = y \tag{3}$$

With initial condition $y(0) = 1$, using Euler's method.

5.3 Logarithmic Functions

Natural logarithm is implemented using numerical integration:

$$\ln(x) = \int_1^x \frac{1}{t} dt \tag{4}$$

5.4 Inverse Trigonometric Functions

Inverse trigonometric functions (asin, acos, atan) are implemented using Newton's method to numerically solve equations like $\sin(y) = x$ for y.

6 User Interface and Interaction Flow

6.1 Input Method

The calculator uses a cycling input method due to the limited number of buttons:

- Digit button cycles through digits 0-9
- Operation button cycles through +, -, *, /
- Function button cycles through available functions
- SET button commits current selection to the expression

6.2 Display Feedback

The 16x2 LCD display shows:

- First line: Current expression being built
- Second line: Current selection (digit, operation, or function)
- Parenthesis depth counter to help with expression building

6.3 Expression Building Process

1. User cycles to desired digit/operation/function
2. SET button commits selection to expression
3. Repeat until complete expression is built
4. EQUALS button evaluates expression and displays result
5. CLEAR button acts as backspace for corrections

7 Conclusion

The Arduino scientific calculator demonstrates the implementation of complex mathematical algorithms on limited hardware. The use of the Shunting Yard algorithm enables proper handling of operator precedence, while custom implementations of mathematical functions allow for advanced calculations without external libraries.

Key achievements of this project include:

- Efficient expression parsing and evaluation
- Implementation of scientific functions using numerical methods
- Intuitive user interface despite limited input capability
- Proper handling of complex expressions with parentheses

Future improvements could include:

- Addition of more mathematical functions
- Memory functionality for storing results
- Improved display with scrolling for longer expressions
- Enhanced precision for floating-point calculations