Scientific Calculator Using Arduino Uno

Your Name

May 25, 2025

Abstract

This project demonstrates the implementation of a scientific calculator using an Arduino Uno, a 16x2 LCD, and a 5x5 button matrix. The device evaluates complex mathematical expressions, including nested functions and operations with full BODMAS precedence, using custom numerical methods and parsing techniques.

Hardware Description

- **Microcontroller:** Arduino Uno R3
- **Display:** 16x2 LCD (JHD162A) connected to pins 7 to 12
- **Input:** 5x5 matrix keypad using tactile switches
- **Pins Used:**
  - Rows: Digital Pins 2 to 6
  - Columns: Analog Pins A0 to A4
  - LCD: Pins 7 (RS), 8 (EN), 9-12 (D4-D7)
- **Power:** USB or external adapter via Arduino

Software Overview

The calculator firmware is written in Arduino C++ using the `LiquidCrystal` library for display and custom logic for expression parsing. The code includes:

- Button matrix scanning
- LCD display handling
- Recursive expression parser with BODMAS logic
- Numerical implementations of sin, cos, log, sqrt, cbrt

Code Explanation

1. Library and LCD Setup

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
```

This includes the LiquidCrystal library and defines the LCD pins connected to Arduino.

2. Keypad and Mapping

```
const int rowPins[5] = {2, 3, 4, 5, 6};
const int colPins[5] = {A0, A1, A2, A3, A4};
const char buttonMap[5][5] = {
    {'0', '1', '2', '3', '4'},
    {'5', '6', '7', '8', '9'},
    {'+', '-', '*', '/', '='},
    {'s', 'c', 'l', 'q', 'r'},
    {'(', ')', '.', 'p', 'C'}
};
```

Each button in the 5x5 matrix is mapped to a character (digits, operators, functions, etc).

3. Initialization

```
void setup() {
    lcd.begin(16, 2);
    lcd.print("Sci Calculator");
    delay(1500);
    lcd.clear();
    for (int i = 0; i < 5; i++) {
        pinMode(rowPins[i], OUTPUT);
        digitalWrite(rowPins[i], HIGH);
    }
    for (int j = 0; j < 5; j++) {
        pinMode(colPins[j], INPUT_PULLUP);
    }
}
```

Initializes the LCD and sets up the keypad: rows as outputs, columns with pull-ups.

4. Main Loop and Key Scan

```
void loop() {
    char key = scanKeypad();
    if (key != '\0') {
```

```
        handleKeyPress(key);
    }
}
```

```
char scanKeypad() {
    for (int i = 0; i < 5; i++) {
        digitalWrite(rowPins[i], LOW);
        for (int j = 0; j < 5; j++) {
            if (digitalRead(colPins[j]) == LOW) {
                delay(200);
                while (digitalRead(colPins[j]) == LOW);
                digitalWrite(rowPins[i], HIGH);
                return buttonMap[i][j];
            }
        }
        digitalWrite(rowPins[i], HIGH);
    }
    return '\0';
}
```

Scans each row by pulling it low, reads the columns to detect which button is pressed.

5. Handling Key Press

```
void handleKeyPress(char key) {
    if (key == 'C') {
        inputExpression = "";
        lcd.clear();
    } else if (key == '=') {
        float result = evaluateExpression(inputExpression);
        lcd.clear();
        if (result == -9999) lcd.print("Error");
        else lcd.print(result);
        inputExpression = "";
    } else if (key == 'p') {
        inputExpression += "3.14159";
        lcd.print("3.14159");
    } else {
        inputExpression += key;
        lcd.print(key);
    }
}
```

Handles building the expression string, clearing it, evaluating it, or inserting π.

6. Expression Evaluation

```
float evaluateExpression(String expr) {
    for (int i = 0; i < expr.length(); i++) {
        if (expr[i] == 's' || expr[i] == 'c' || ...) {
            int startIndex = i + 1;
            if (expr[startIndex] != '(') return -9999;
            int endIndex = findMatchingClosingParenthesis(expr,
               ↪ startIndex);
            ...
            float argument = evaluateExpression(argumentStr);
            float computedValue = computeFunction(expr[i], argument);
            expr = expr.substring(0, i) + String(computedValue) + expr.
               ↪ substring(endIndex + 1);
            i = -1;
        }
    }
    return parseBODMAS(expr);
}
```

Handles recursive parsing for nested functions like `sin(log(90))`.

7. Compute Math Functions

```
float computeFunction(char func, float value) {
    switch (func) {
        case 's': return numericalSin(...);
        case 'c': return numericalCos(...);
        case 'l': return numericalLog(value);
        case 'q': return numericalSqrt(value);
        case 'r': return numericalCubeRoot(value);
        default: return -9999;
    }
}
```

8. Parsing BODMAS

```
float parseBODMAS(String expr) {
    int startParen = expr.indexOf('(');
    while (startParen != -1) {
        int endParen = findMatchingClosingParenthesis(expr, startParen)
           ↪ ;
        String subExpr = expr.substring(startParen+1, endParen);
        float subResult = parseBODMAS(subExpr);
        expr = expr.substring(0, startParen) + String(subResult) + expr
           ↪ .substring(endParen+1);
        startParen = expr.indexOf('(');
```

```
    }
    return evaluateBasicBODMAS(expr);
}
```

Recursively evaluates expressions inside parentheses before computing the final result.

9. Operator Precedence Evaluation

```
float evaluateBasicBODMAS(String expr) {
    // +, -, *, / handled with correct order
    ...
}
```

Implements a simplified operator precedence evaluator without using stacks.

10. Numerical Math Implementations

```
float numericalSin(float x) { ... }
float numericalCos(float x) { ... }
float numericalLog(float x) { ... }
float numericalSqrt(float x) { ... }
float numericalCubeRoot(float x) { ... }
```

All functions are implemented using numerical methods like integration or Newton-Raphson.

Conclusion

This calculator is capable of evaluating highly complex expressions, supports function nesting, and mimics real scientific calculator behavior. With plans to port this to AVR-GCC, it also serves as a strong embedded systems and parsing logic demonstration.

Links

- GitHub: `https://github.com/mr21sk/my-projects/Scientific-Calculator`

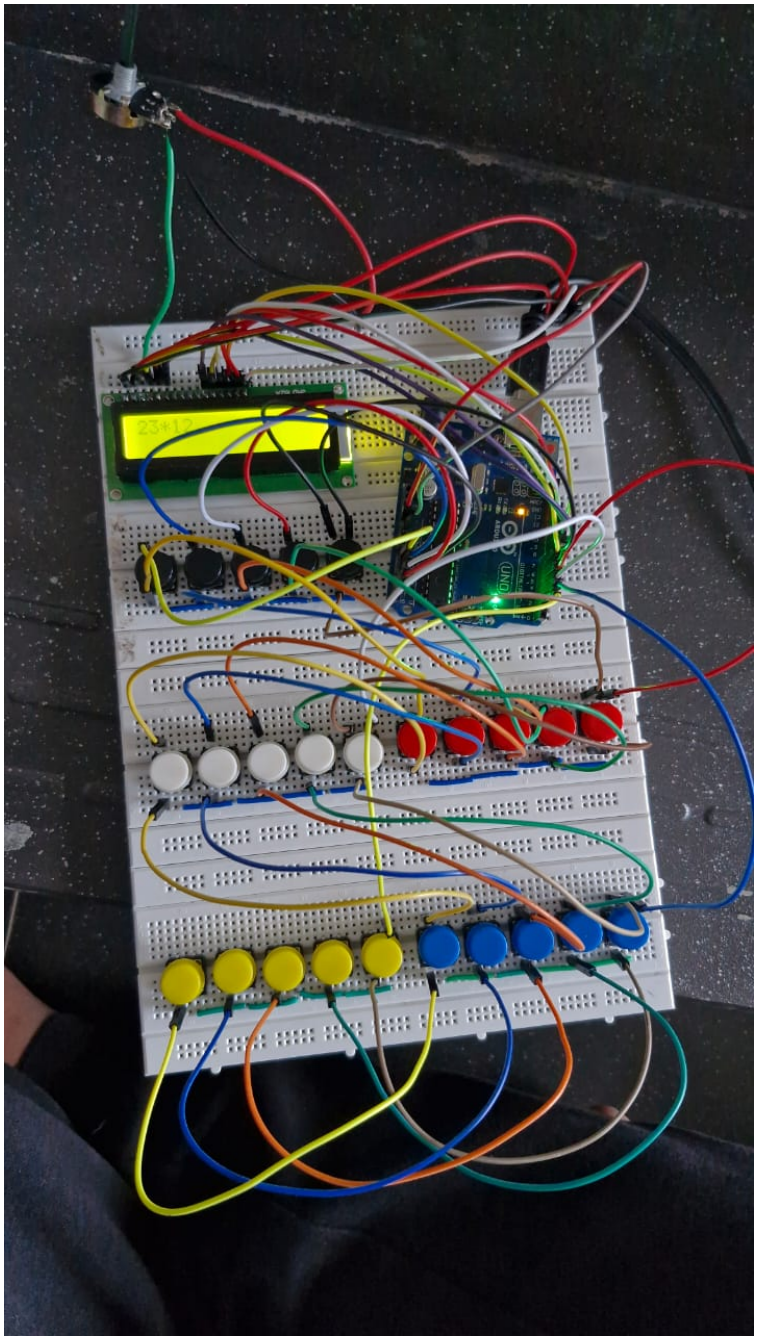

Figure 1: Scientific Calculator using Arduino