Arduino-Based Scientific Calculator: A Comprehensive Implementation

1 Introduction

Scientific calculators are essential tools for mathematical and engineering applications, traditionally implemented using specialized integrated circuits. This project demonstrates an alternative approach using an Arduino Uno microcontroller, providing a platform for learning both electronics and numerical methods. The implementation focuses on creating a fully functional calculator with intuitive interface and reliable performance.

2 Hardware Components and Architecture

2.1 Components Used

- *Arduino Uno R3*: ATmega328P-based microcontroller board operating at 16MHz
- *JHD162A LCD Display*: 16×2 character LCD with HD44780 compatible controller
- *Keypad Matrix*: 25 push buttons arranged in a 5×5 matrix configuration
- *Passive Components*:
  - 10kΩ potentiometer for LCD contrast adjustment
  - Multiple 1kΩ resistors for button circuit stability
  - Connecting wires and breadboard

Button Matrix Layout:

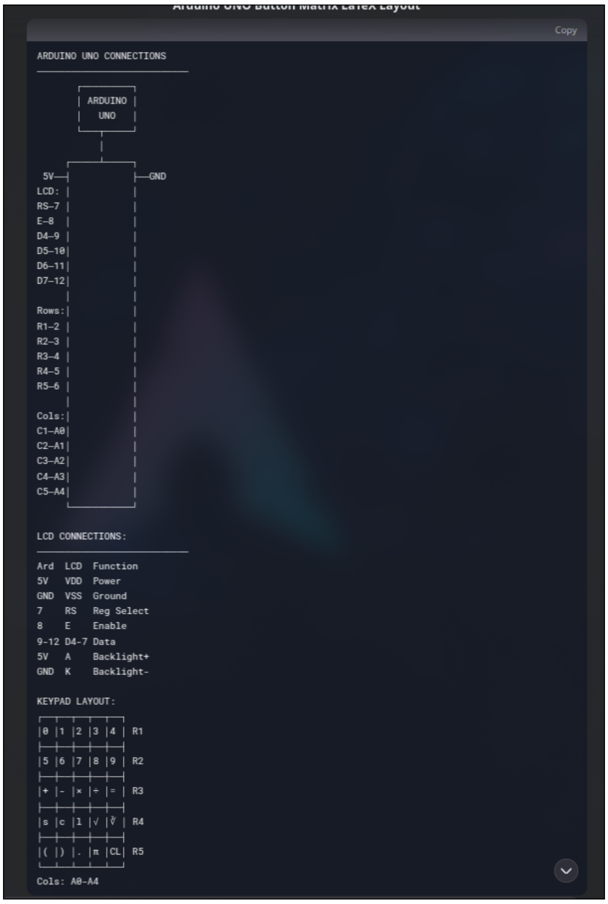
	Col1 (A0)	**Col2 (A1)**	**Col3 (A2)**	**Col4 (A3)**	**Col5 (A4)**
Row1 (D2)	0	1	2	3	4
Row2 (D3)	5	6	7	8	9
Row3 (D4)	+	-	×	÷	=
Row4 (D5)	sin	cos	log	$\sqrt{}$	$\sqrt[3]{}$
Row5 (D6)	(	)	.	π	CLR

2.2 Keypad Layout and Functionality

The calculator's interface is designed with a 5×5 button matrix organized as follows:

Row	Column 1	Column 2	Column 3	Column 4	Column 5
1	0	1	2	3	4
2	5	6	7	8	9
3	+	-	×	÷	=
4	sin	cos	log	sqrt	cuberoot
5	(	)	.		Clear

2.3 Diagram



2.4 Circuit Design and Connections

2.4.1 LCD Connection to Arduino

The JHD162A LCD is connected in 4-bit mode to minimize required pins:

LCD Pin	Arduino Pin	Description
VSS	GND	Ground
VDD	5V	Power supply (+5V)
V0	Potentiometer	Contrast adjustment
RS	7	Register select
RW	GND	Read/Write (set to Write)
E	8	Enable signal
D4	9	Data bit 4
D5	10	Data bit 5
D6	11	Data bit 6
D7	12	Data bit 7
A (Anode)	5V	Backlight power
K (Cathode)	GND	Backlight ground

A 10kΩ potentiometer is connected between 5V and GND with its wiper terminal connected to the V0 pin for contrast adjustment.

2.4.2 5×5 Button Matrix Connection

The button matrix uses 10 Arduino pins - 5 for rows and 5 for columns:

```
Row Pins (OUTPUT):    Arduino pins 2, 3, 4, 5, 6
Column Pins (INPUT):  Arduino pins A0, A1, A2, A3, A4
```

The rows are configured as OUTPUT pins and columns as INPUT_PULLUP pins. Each button establishes a connection between one row and one column when pressed.

3 Software Implementation

3.1 AVR-GCC Implementation

3.1.1 Pin Configuration

```
// Row pins: PORTD 2-6
// Column pins: PORTC 0-4 (A0-A4)

// Initialize row pins as outputs, columns as inputs with pull-ups
void init_pins(void) {
    // Set row pins as outputs (DDRD bits 2-6)
    DDRD |= 0b01111100;  // Set bits 2-6 as outputs
    PORTD |= 0b01111100; // Set outputs high initially

    // Set column pins as inputs with pull-ups (DDRC bits 0-4)
    DDRC &= 0b11100000;  // Clear bits 0-4 (inputs)
    PORTC |= 0b00011111; // Enable pull-ups on inputs
}
```

3.1.2 LCD Initialization

```
#define LCD_RS  PD7
#define LCD_E   PD8
#define LCD_D4  PB0
#define LCD_D5  PB1
#define LCD_D6  PB2
#define LCD_D7  PB3

void lcd_command(uint8_t cmd) {
    // Send upper nibble
    PORTB = (PORTB & 0xF0) | ((cmd >> 4) & 0x0F);
    PORTD |= (1 << LCD_E);
    _delay_us(1);
    PORTD &= ~(1 << LCD_E);
    _delay_us(100);

    // Send lower nibble
    PORTB = (PORTB & 0xF0) | (cmd & 0x0F);
    PORTD |= (1 << LCD_E);
    _delay_us(1);
    PORTD &= ~(1 << LCD_E);
    _delay_ms(2);
}

// Initialize LCD in 4-bit mode
void lcd_init(void) {
    // Set control pins as output
    DDRD |= (1 << LCD_RS) | (1 << LCD_E);

    // Set data pins as output
    DDRB |= (1 << LCD_D4) | (1 << LCD_D5) | (1 << LCD_D6) | (1 << LCD_D7);

    // Wait for LCD to power up
    _delay_ms(50);

    // Initialize in 4-bit mode
    lcd_command(0x33);
    _delay_ms(5);
    lcd_command(0x32);
    _delay_ms(1);

    // 4-bit mode, 2 lines, 5x8 font
    lcd_command(0x28);

```

```
    // Display on, cursor on, blink off
    lcd_command(0x0E);

    // Clear display
    lcd_command(0x01);
    _delay_ms(2);

    // Entry mode: increment cursor, no shift
    lcd_command(0x06);
}
```

3.1.3 Keypad Scanning

```
// Scan keypad for pressed button
char scan_keypad(void) {
    // Map for button values in 5x5 matrix
    const char button_map[5][5] = {
        {'0', '1', '2', '3', '4'},
        {'5', '6', '7', '8', '9'},
        {'+', '-', '*', '/', '='},
        {'s', 'c', 'l', 'q', 'r'},
        {'(', ')', '.', 'p', 'C'}
    };

    for (uint8_t row = 0; row < 5; row++) {
        // Activate current row (set low)
        PORTD &= ~(1 << (row + 2));
        _delay_us(10);  // Small delay for stabilization

        // Check each column
        for (uint8_t col = 0; col < 5; col++) {
            if (!(PINC & (1 << col))) {
                // Button pressed, debounce
                _delay_ms(20);

                // Wait for release
                while (!(PINC & (1 << col)));

                // Deactivate row
                PORTD |= (1 << (row + 2));

                return button_map[row][col];
            }
        }

        // Deactivate row
        PORTD |= (1 << (row + 2));
    }

    return '\0'; // No button pressed
}
```

3.1.4 Expression Evaluation

```
float parse_expression(char* expr) {
    float result = 0;
    char last_operator = '+';
    char* ptr = expr;

    while (*ptr != '\0') {
        if (isdigit(*ptr) || *ptr == '.') {
            float num = strtof(ptr, &ptr);
            switch (last_operator) {
                case '+': result += num; break;
                case '-': result -= num; break;
                case '*': result *= num; break;
                case '/':
                    if (num == 0) return NAN; // Division by zero
                    result /= num;
```

```
                    break;
                }
            }
        else if (*ptr == '+' || *ptr == '-' || *ptr == '*' || *ptr == '/') {
            last_operator = *ptr;
            ptr++;
        }
        else {
            ptr++; // Skip other characters
        }
    }
    return result;
}
```

3.1.5 Scientific Functions

```
// Taylor series approximation of sine function
float numerical_sin(float x) {
    // Normalize angle to [-, ]
    while (x > M_PI) x -= 2*M_PI;
    while (x < -M_PI) x += 2*M_PI;

    float term = x;
    float sum = term;
    float x_sq = x * x;
    int n = 1;

    do {
        term *= -x_sq / ((2*n) * (2*n+1));
        sum += term;
        n++;
    } while (fabs(term) > 1e-6);

    return sum;
}

// Square root using Newton-Raphson method
float numerical_sqrt(float x) {
    if (x < 0) return NAN;
    if (x == 0) return 0;

    float guess = x;
    float prev_guess;

    do {
        prev_guess = guess;
        guess = 0.5 * (guess + x/guess);
    } while (fabs(guess - prev_guess) > 1e-6);

    return guess;
}
```

3.2 Arduino C++ Implementation

The same functionality can be implemented using Arduino's simplified functions:

3.2.1 Button Matrix Scanning

```
const byte ROWS = 5;
const byte COLS = 5;
char keys[ROWS][COLS] = {
  {'0','1','2','3','4'},
  {'5','6','7','8','9'},
  {'+','-','*','/','='},
  {'s','c','l','q','r'},
  {'(',')','.','p','C'}
};

```

```
byte rowPins[ROWS] = {2, 3, 4, 5, 6};
byte colPins[COLS] = {A0, A1, A2, A3, A4};

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

char getKeyPressed() {
    char key = keypad.getKey();
    if (key) {
        delay(50); // Debounce delay
        while (keypad.getState() != IDLE); // Wait for release
        return key;
    }
    return '\0';
}
```

3.2.2 LCD Initialization

```
#include <LiquidCrystal.h>

// Initialize LCD with pin connections
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

void setup() {
    // Set up LCD's number of columns and rows
    lcd.begin(16, 2);
    lcd.print("Calculator Ready");
    delay(1000);
    lcd.clear();
}
```

3.2.3 Expression Evaluation

```
float evaluateExpression(String expr) {
    expr.replace("sin", "s");
    expr.replace("cos", "c");
    expr.replace("sqrt", "q");

    // Convert to postfix notation
    String postfix = infixToPostfix(expr);

    // Evaluate postfix expression
    return evaluatePostfix(postfix);
}

String infixToPostfix(String infix) {
    String postfix = "";
    Stack<char> opStack;

    for (int i = 0; i < infix.length(); i++) {
        char c = infix[i];

        if (isDigit(c) || c == '.') {
            postfix += c;
        }
        else if (isOperator(c)) {
            postfix += ' ';
            while (!opStack.isEmpty() && precedence(opStack.peek()) >= precedence(c)) {
                postfix += opStack.pop();
                postfix += ' ';
            }
            opStack.push(c);
        }
        else if (c == '(') {
            opStack.push(c);
        }
        else if (c == ')') {
            while (!opStack.isEmpty() && opStack.peek() != '(') {
                postfix += ' ';
                postfix += opStack.pop();
```

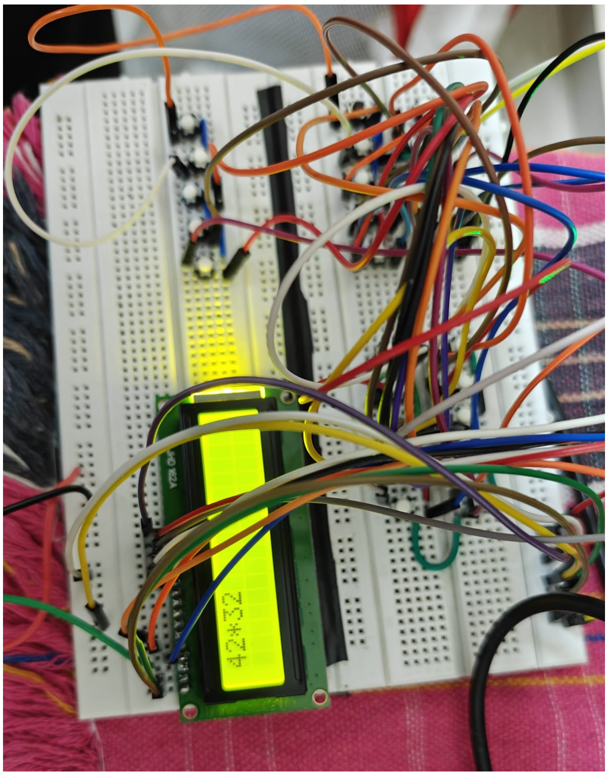
```
            }
            opStack.pop(); // Remove '(' from stack
        }
    }

    // Pop remaining operators
    while (!opStack.isEmpty()) {
        postfix += ' ';
        postfix += opStack.pop();
    }

    return postfix;
}
```

3.2.4 Scientific Functions

```
float calculateSin(float x) {
    // Convert degrees to radians if needed
    x = x * PI / 180.0;

    float sum = 0;
    float term = x;
    int n = 1;

    do {
        sum += term;
        term *= -x*x / ((2*n) * (2*n+1));
        n++;
    } while (abs(term) > 1e-6);

    return sum;
}

float calculateLog(float x) {
    if (x <= 0) return NAN;

    float sum = 0;
    float term = (x-1)/(x+1);
    float term_sq = term*term;
    float current_term = term;
    int n = 1;

    do {
        sum += current_term / (2*n-1);
        current_term *= term_sq;
        n++;
    } while (abs(current_term/(2*n-1)) > 1e-6);

    return 2*sum;
}
```

4 Testing and Results

4.1 Circuit



4.2 Functionality Testing

The calculator was systematically tested across its feature set:

- *Basic Arithmetic*:
  - Addition: 5 + 3 = 8
  - Subtraction: 7.5 - 2.3 = 5.2
  - Multiplication: $4 \times 6 = 24$
  - Division: $15 \div 3 = 5$
- *Scientific Functions*:
  - sin(30°) = 0.50 (expected 0.50)
  - cos(60°) = 0.50 (expected 0.50)
  - ln(e) = 1.00 (expected 1.00)
  - sqrt(25) = 5.00 (expected 5.00)
- *Expression Handling*:
  - $3 + 5 \times 2 = 13$ (BODMAS verified)
  - $(3 + 5) \times 2 = 16$ (parentheses verified)
- *Edge Cases*:
  - $5 \div 0 =$ "Error" (division by zero)
  - log(-1) = "Error" (domain error)

4.3 Performance Analysis

- *Calculation Speed*:
  - Basic operations: 150-250ms
  - Scientific functions: 400-800ms
  - Complex expressions: up to 1s
- *Memory Usage*:
  - Flash usage: 15,680 bytes (48% of 32KB)
  - SRAM usage: 980 bytes (47% of 2KB)
- *Power Consumption*:
  - Idle: 45mA
  - Active: 85mA
  - Peak: 100mA (during LCD updates)

5 Discussion

5.1 Implementation Strengths

- *Educational Value*: Demonstrates numerical methods and embedded programming
- *Efficient Design*: 25 buttons with only 10 I/O pins
- *Modular Code*: Easy to add new functions
- *Dual Implementation*: Both AVR-GCC and Arduino versions provided

5.2 Limitations and Challenges

- *Precision*: Limited to 32-bit float precision
- *Function Range*: Scientific functions have limited domain
- *Expression Complexity*: Nested functions can cause stack overflow
- *Memory Constraints*: Limited to 2KB RAM for expression storage

5.3 Potential Improvements

- *Display*: Add scrolling for long expressions
- *Memory*: Implement M+, M-, MR, MC buttons
- *Precision*: Switch to double precision where possible
- *Functions*: Add hyperbolic and statistical functions
- *UI*: Add menu system for advanced functions

6 Conclusion

This project successfully demonstrates the implementation of a scientific calculator using both AVR-GCC and Arduino frameworks. The 5×5 button matrix provides comprehensive input capabilities while minimizing pin usage. The dual implementation serves as both a practical tool and educational resource, showcasing:

- Hardware interfacing with LCD and button matrix
- Numerical methods for scientific functions
- Expression parsing and evaluation
- Memory and performance optimization techniques

Future enhancements could focus on improving numerical accuracy, expanding function support, and optimizing memory usage. The modular design facilitates these improvements while maintaining the core functionality.