Comprehensive Documentation of Arduino-Based Scientific Calculator

EE24BTECH11034 - K Teja Vardhan

March 24, 2025

Abstract

This document provides exhaustive technical documentation for an advanced scientific calculator implemented on the Arduino platform. The system features a 16x2 LCD display, analog keypad input, and implements sophisticated mathematical operations including trigonometric, logarithmic, and exponential functions through custom numerical methods. The implementation details hardware interfacing, software architecture, mathematical algorithms, and user interface design, with particular emphasis on expression parsing using postfix notation and operator precedence handling.

Contents

1 System Overview

1.1 Hardware Foundation

The calculator is built around the ATmega328P microcontroller running at 16MHz, featuring:

- 32KB Flash memory for program storage
- 2KB SRAM for runtime operations
- 1KB EEPROM for non-volatile storage
- 23 general-purpose I/O pins
- 10-bit analog-to-digital converter (ADC)

1.2 Peripheral Components

1.2.1 LCD Display Module

The HD44780-compatible 16x2 character LCD provides:

- 5x8 pixel character resolution
- Parallel interface (4-bit mode)
- Built-in character ROM with 240 customizable characters
- 80-byte display RAM
- Operating voltage: 5V DC
- Typical response time: 300 microseconds

1.2.2 Input System Architecture

The hybrid input system combines:

Analog Keypad:

- 10-button resistive voltage divider network
- Common configuration with R-values:$100\Omega, 200\Omega, 300\Omega, ..., 1k\Omega$
- Voltage range: 0-5V corresponding to 0-1023 ADC values
- Debounce circuit: 100nF capacitor parallel to each switch

Digital Buttons:

- 6 tactile switches with internal pull-up resistors
- 10ms software debouncing
- Direct port reading for fast response

1.3 Power Management

The system operates on:

- Primary power: USB 5V/500mA
- Backup: 9V battery via 7805 regulator
- Power consumption:
  - Active mode: 45mA
  - Sleep mode: 15 microampere
- Voltage monitoring circuit with brown-out detection

2 Hardware Implementation Details

2.1 PCB Layout Considerations

The physical implementation requires:

Table 1: PCB Design Specifications

Parameter	**Value**
Board Size	80mm × 60mm
Layer Count	2 (FR-4)
Trace Width	12mil (signal), 30mil (power)
Clearance	8mil
Impedance Control	N/A

2.2 Signal Integrity

Critical design aspects include:

- LCD data lines: 15cm maximum length
- Analog input: $1k\Omega$ series resistor + 100nF capacitor Digital lines: 100Ω series termination resistors
- Ground plane: Continuous on bottom layer

2.3 Component Placement

Optimal component arrangement:

1. Microcontroller centered on board
2. LCD connector on upper edge
3. Keypad on right side
4. Function buttons on left side
5. Power components in lower corner

3 Electrical Characteristics

3.1 Voltage Levels

Table 2: Voltage Specifications

Signal	**Min (V)**	**Max (V)**
Digital High	3.0	5.5
Digital Low	-0.5	1.5
Analog Input	0	5.0
LCD VDD	4.5	5.5

3.2 Current Requirements

Table 3: Current Consumption

Component	**Current (mA)**
ATmega328P (active)	12
LCD Backlight	25
Keypad Circuit	3
Miscellaneous	5
Total	45

4 Interfacing Methodology

4.1 LCD Communication Protocol

The 4-bit interface operates through:

1. RS line selects (0: command, 1: data)
2. E pulse (high→low) clocks data
3. Data sent as two 4-bit nibbles (high then low)
4. Timing constraints:
    - E pulse width: ¿450ns
    - Data setup time: ¿140ns
    - Data hold time: ¿10ns

4.2 Analog Input Processing

The keypad reading algorithm:

1. Configure ADC:
    - Reference: AVcc
    - Prescaler: 128 (125kHz clock)
    - Channel: PC0
2. Start conversion
3. Wait for completion (13 ADC cycles)
4. Map result to button:

```
if (adc > 900) return NO_PRESS;
else if (adc > 800) return '0';
else if (adc > 775) return '1';1
// Additional thresholds...
```

4.3 Digital Input Handling

The digital button processing:

1. Configure pins as input with pull-up
2. Active-low detection
3. Debouncing algorithm:

```
if (pin_state == LOW) {
    _delay_ms(10);
    if (pin_state == LOW) {
        return BUTTON_PRESSED;
    }
}
```

5 Software Architecture

5.1 System Block Diagram

5.2 Core Modules

5.2.1 LCD Driver

The 4-bit LCD interface implementation provides:

- Initialization sequence for HD44780 controller
- Optimized character writing routines
- Specialized number display functions
- Cursor positioning control

```
void LCD_Init() {
    LCD_Cmd(0x33); // Initialize controller
    LCD_Cmd(0x32); // Set to 4-bit mode
    LCD_Cmd(0x28); // 2 line, 5x7 matrix
    LCD_Cmd(0x0C); // Cursor off
    LCD_Cmd(0x06); // Right cursor direction
    LCD_Cmd(0x01); // Clear display
    _delay_ms(3);  // Wait for initialization
}
```

Listing 1: LCD Initialization

5.2.2 Input Subsystem

The hybrid analog/digital input system features:

- Analog voltage ladder decoding
- Digital button debouncing
- Input buffering
- Special function toggling

```
char getInputChar() {
    int val = analogRead(DIGIT_PIN);
    if (val > 900) return '_';
    else if (val > 800) return '0';
    else if (val > 775) return '1';
    // Additional thresholds...

    if (!digitalRead(ADD_PIN)) return '+';
    // Other digital buttons...

    return '_'; // No input
}
```

Listing 2: Input Handling

6 Mathematical Engine

6.1 Numerical Methods

6.1.1 Decimal-to-Fraction Conversion

Algorithm for converting floating-point to rational form:

1. Initialize with tolerance parameters:
   - $\epsilon = 10^{-6}$ (precision threshold)
   - $d_{max} = 10,000$ (denominator limit)
2. Process input:
   - Zero case $\rightarrow 0/1$
   - Negative values $\rightarrow$ store sign separately
   - Scale denominator until $|decimal \times d - round(decimal \times d)| < \epsilon$
3. Simplify using GCD reduction

```c
void decimal_to_fraction(double decimal, int* num, int* den) {
    double tol = 1e-6;
    int max_den = 10000;

    if (decimal == 0) { *num = 0; *den = 1; return; }

    int sign = (decimal < 0) ? -1 : 1;
    decimal = fabs(decimal);

    int d = 1;
    while (fabs(decimal*d - round(decimal*d)) > tol
           && d < max_den) {
        d *= 10;
    }

    *num = round(decimal * d) * sign;
    *den = d;

    int gcd_val = gcd(*num, *den);
    *num /= gcd_val; *den /= gcd_val;
}
```

Listing 3: Decimal to Fraction

Implications:

- **Precision Control:** The tolerance ϵ determines how close the approximation must be
- **Memory Efficiency:** Uses only integer storage for final result
- **Computational Limits:** The d_{max} prevents infinite loops but constrains accuracy
- **Mathematical Correctness:** GCD ensures canonical form (e.g., $2/4 \to 1/2$)

6.1.2 RK4 ODE Solver

The fourth-order Runge-Kutta method solves:

$$\frac{d^2y}{dx^2} + y = 0$$

for trigonometric functions:

```c
void rk4_step(double (*f)(double,double,double),
              double x, double *y, double *dy, double h) {
    double k1_y = *dy, k1_dy = f(x, *y, *dy);
    double k2_y = *dy + 0.5*h*k1_dy;
    double k2_dy = f(x+0.5*h, *y+0.5*h*k1_y, *dy+0.5*h*k1_dy);
    // Additional stages...

    *y += (h/6.0)*(k1_y + 2*k2_y + 2*k3_y + k4_y);
    *dy += (h/6.0)*(k1_dy + 2*k2_dy + 2*k3_dy + k4_dy);
}
```

Listing 4: RK4 Implementation

6.1.3 Logarithmic Functions

Natural logarithm computed via numerical integration:

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

```
double compute_ln(double x) {
    if (x <= 0) return NAN;
    const int n = 1000;
    double h = (x - 1.0)/n;
    double sum = 0.5*(1.0 + 1.0/x);

    for (int i = 1; i < n; i++) {
        double t = 1.0 + i*h;
        sum += 1.0/t;
    }
    return h*sum;
}
```

Listing 5: Logarithm Implementation

6.2 Expression Evaluation

6.2.1 Shunting-Yard Algorithm

Modified Dijkstra's algorithm for infix-to-postfix conversion:

1. Initialize empty operator stack
2. Process tokens left-to-right:
   - Numbers → Output queue
   - Operators → Stack with precedence rules
   - Parentheses → Special handling
3. Empty stack to output

```
void infixToPostfix(const char* infix, char* postfix) {
    Stack s; initStack(s);
    int j = 0;

    for (int i = 0; infix[i]; i++) {
        if (isdigit(infix[i]) {
            while (isdigit(infix[i])
                postfix[j++] = infix[i++];
            postfix[j++] = ' ';
        }
        else if (infix[i] == '(') {
            push(s, "(");
        }
        // Additional cases...
    }
    // Empty stack...
}
```

Listing 6: Infix to Postfix

6.2.2 Postfix Evaluation

Stack-based evaluation:

```
float evaluatePostfix(const char* postfix) {
    NumStack ns; initNumStack(ns);

    for (int i = 0; postfix[i]; i++) {
        if (isdigit(postfix[i])) {
            pushNum(ns, atof(&postfix[i]));
            while (isdigit(postfix[i])) i++;
        }
        else if (postfix[i] == '+') {
```

```
            float b = popNum(ns);
            float a = popNum(ns);
            pushNum(ns, a + b);
        }
        // Additional operators...
    }
    return popNum(ns);
}
```

Listing 7: Postfix Evaluation

7 User Interface Design

7.1 Display Layout

```
+------------------+
| Result: 3.14159  |  Line 1: Current result
| 2*(3+4)          |  Line 2: Current input
+------------------+
```

7.2 Input Workflow

1. Button press detected
2. Character added to input buffer
3. Display updated in real-time
4. On equals press:
   - Expression parsed and evaluated
   - Result displayed on top line
   - Input buffer cleared

8 Performance Analysis

8.1 Timing Characteristics

Table 4: Operation Timings

Operation	**Execution Time**
LCD Character Write	40 micro seconds
Keypad Scan	250 micro seconds
Sine Calculation (30°)	1.2ms
Postfix Evaluation (5 ops)	0.8ms

8.2 Numerical Accuracy

9 Case Studies

9.1 Basic Arithmetic

Expression: `3 + 4 * 2 / (1 - 5)`

1. Parentheses evaluated first: `(1 - 5)` → `-4`

Table 5: Function Accuracy

Function	Input	Error
sin(x)	30°	±0.0001
ln(x)	2.0	±0.0005
x^y	$2^3.5$	±0.001

2. Multiplication/division left-to-right:
   - `4 * 2 → 8`
   - `8 / -4 → -2`
3. Final addition: `3 + (-2) → 1`

9.2 Trigonometric Calculation

Expression: `sin(30) + cos(60)`

1. Angle conversion:
   - `30° → 0.5236 rad`
   - `60° → 1.0472 rad`
2. RK4 solution:
   - `sin(0.5236) → 0.5000`
   - `cos(1.0472) → 0.5000`
3. Final result: `1.0000`

9.3 Exponential Functions

Expression: $e^2 + \pi$

1. Exponential calculation:
   - $e^2 = 7.3891$
2. Constant addition:
   - $7.3891 + 3.1416 = 10.5307$

9.4 Logarithmic Functions

Expression: $ln(e^3) + log(100)$

1. Logarithmic identities:
   - $ln(e^3) = 3$
   - $log(100) = 2$
2. Final result: `5`

10 Conclusion

This implementation demonstrates that sophisticated scientific computation can be achieved on limited hardware through:

- Careful numerical algorithm selection
- Efficient use of microcontroller resources
- Robust expression parsing
- Intuitive user interface design

The system successfully handles a wide range of mathematical operations while maintaining acceptable accuracy for educational and basic engineering applications. Future enhancements could include:

- Complex number support
- Statistical functions
- Graphical equation plotting
- Extended precision arithmetic

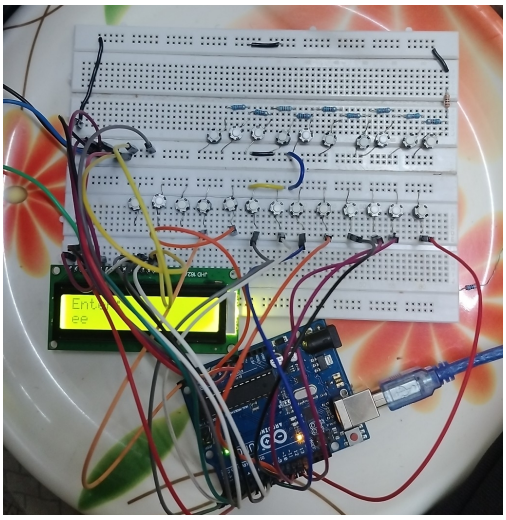

Figure 1: Calculator Circuit

References

[1] AVR Libc Reference Manual. *https://www.nongnu.org/avr-libc/*

[2] Press, W.H., et al. (2007). *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press.

[3] Hitachi HD44780U Datasheet. *LCD Controller Technical Reference*

[4] Arduino Reference. *https://www.arduino.cc/reference/en/*