Scientific Calculator

EE24BTECH11052 - Rongali Charan

Contents

1 Aim

Constructing the Scietific Calculator and using avr-gcc

2 Materials Required

- One 16X2 LCD display
- 5kΩ Potentiometer
- 2 X 180Ω resistors (for common anode current limiting)
- Bread Board
- Arduino UNO(atmega328p) and connecting cable
- Jumper wires for arduino and normal wires for seven segment displays
- Push Buttons(as many functions as needed)

3 Hardware Setup

The wiring process involves connecting the LCD display,Potentiometer to the Arduino pins.

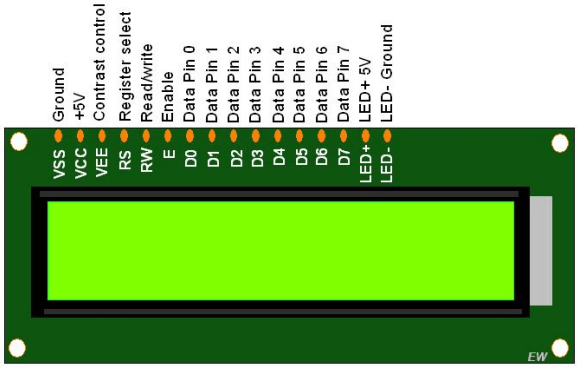

Fig. 0.1: LCD Display

3.1 LCD Connections

LCD Pin	Function	Arduino Pin	Additional Components
1 (VSS)	Ground	GND	—
2 (VDD)	Power Supply	5V	—
3 (V0)	Contrast Control	180 Ω Resistor	Potentiometer (5kΩ)
4 (RS)	Register Select	D2	—
5 (RW)	Read/Write	GND	—
6 (E)	Enable Signal	D3	—
7 (D0)	Data Bit 0	—	Not used
8 (D1)	Data Bit 1	—	Not used
9 (D2)	Data Bit 2	—	Not used
10 (D3)	Data Bit 3	—	Not used
11 (D4)	Data Bit 4	D4	—
12 (D5)	Data Bit 5	D5	—
13 (D6)	Data Bit 6	D6	—
14 (D7)	Data Bit 7	D7	—
15 (LED+)	Backlight Power	5V	—
16 (LED-)	Backlight Ground	GND	—

3.2 Button Connections

Button Index	Function	Arduino Pin
0	Button 0	D8
1	Button 1	D9
2	Button 2	D10
3	Button 3	D11
4	Button 4	D12
5	Button 5	A0
6	Button 6	A1
7	Button 7	A2
8	Button 8	A3
9	Button 9	A4
Shift Button	Shift Mode	A5
Extra Button	Extra Function	D13

4 Circuit Details

- The potentiometer (5kΩ) is connected between the VDD (5V) and GND pins, with the wiper going to V0 (LCD pin 3) to control the contrast.
- The LCD's backlight (LED+) is powered by 5V and LED- is connected to GND .

5 Common Issues

Floating Pins and False Triggering

Problem: When a button is not pressed, the pin might be in a floating state (neither HIGH nor LOW), causing false button presses or multiple triggers.
Cause: Without a resistor, the input pin may pick up noise or undefined signals, causing

random fluctuations.
Symptoms:

- Random and unexpected button presses.
- Inconsistent calculator behavior.

Solution:

- Use external pull-down resistors (10kΩ) or activate internal pull-up resistors to ensure stable pin states.

Button Debouncing Issues

Problem: Mechanical bouncing during button presses generates multiple signals, resulting in repeated key registrations.
Cause: No debounce mechanism is applied, allowing multiple signals from a single press.
Symptoms:

- Multiple digits displayed for a single press.
- Erratic or unexpected calculator operations.

Solution:

- Implement a debouncing mechanism in software or use capacitors to filter out noise.

Excessive Power Consumption

Problem: When buttons are pressed without resistors, a direct short may occur, causing excessive current draw.
Cause: Direct shorting without resistance protection.
Symptoms:

- Potential damage to the Arduino board.
- Overheating or instability.

Solution:

- Use current-limiting resistors (180-220 Ω) to prevent high current flow.
- Enable internal pull-ups as a temporary solution.

Voltage Spikes and EMI Issues

Problem: Button presses without resistors can generate voltage spikes and electromagnetic interference (EMI), disrupting the Arduino's stability.
Cause: Inductive effects and lack of resistance create transients.
Symptoms:

- Random resets of the Arduino.
- Display glitches or freezing.

Solution:

- Add small capacitors (0.1 μF) in parallel with the buttons to suppress noise.
- Use software filtering techniques to smooth out transient signals.

Incorrect Pin States and Code Logic Issues

Problem: Without resistors, the default pin state may fluctuate randomly, causing incorrect button readings.
Cause: Floating pins lead to unstable logic detection.
Symptoms:

- Inaccurate or unpredictable button presses.
- Incorrect calculations or display errors.

Solution:

- Use internal pull-up resistors or add external resistors to prevent floating states.

Key Takeaway

Even though resistors on buttons are not mandatory, using pull-down or pull-up resistors (either internal or external) ensures:

- Stable and reliable input readings.
- Prevention of floating states and false triggering.
- Avoidance of unwanted behavior such as multiple or missed triggers.

Recommendation: Use internal pull-up resistors to avoid the need for external resistors while maintaining stable behavior.

6 Code Implementation

LCD Display Functions

The calculator interfaces with an LCD to display results. The key functions include:

- `lcd_init()` - Initializes the LCD.
- `lcd_cmd(command)` - Sends commands to the LCD.
- `lcd_write(char data)` - Writes a character to the LCD.
- `lcd_clear()` - Clears the screen.
- `lcd_print(const char* str)` - Prints a string.

Mathematical Functions

Square Root: Newton-Raphson Method

The square root of a number x is computed using the iterative Newton-Raphson method:

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{S}{x_n}\right)$$

The implementation starts with an initial guess and iterates until convergence.

Listing 1: Square Root Function

```
double my_sqrt(double S) {
    double x = S / 2.0;
    for (int i = 0; i < 10; i++) {
        x = 0.5 * (x + S / x);
    }
    return x;
}
```

Trigonometric Functions

Trigonometric functions are computed using Taylor series approximations.

Sine Function: The sine function is computed using:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Listing 2: Sine Function

```
double my_sin(double x) {
    double term = x, sum = x;
    for (int n = 1; n < 10; n++) {
        term *= -x * x / ((2 * n) * (2 * n + 1));
        sum += term;
    }
    return sum;
}
```

Cosine Function: Similarly, cosine is computed as:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

Listing 3: Cosine Function

```
double my_cos(double x) {
    double term = 1, sum = 1;
    for (int n = 1; n < 10; n++) {
        term *= -x * x / ((2 * n - 1) * (2 * n));
        sum += term;
    }
    return sum;
}
```

Tangent Function: The tangent function is computed using:

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

Listing 4: Tangent Function

```
double my_tan(double x) {
    return my_sin(x) / my_cos(x);
}
```

Logarithm (Natural Log)

The natural logarithm is computed using a series expansion for $x > 0$:

$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots$$

Listing 5: Natural Log Function

```
double my_ln(double x) {
    double sum = 0, term = (x - 1) / (x + 1);
    double num = term * term;
    for (int i = 1; i < 10; i += 2) {
        sum += term / i;
        term *= num;
    }
    return 2 * sum;
}
```

Exponential Function

The exponential function e^x is computed using:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Listing 6: Exponential Function

```
double my_exp(double x) {
    double sum = 1, term = 1;
    for (int n = 1; n < 10; n++) {
        term *= x / n;
        sum += term;
    }
    return sum;
}
```

Expression Evaluation

The calculator evaluates arithmetic expressions using a recursive parser. The expression is tokenized, and the order of operations is handled correctly.

Listing 7: Expression Parsing

```
double evaluate_expression(const char* expr) {
    // Tokenizes and evaluates the expression recursively
}
```

7 Conclusion

This document provides a detailed explanation of the scientific calculator's implementation on the AVR platform. The calculator is capable of evaluating mathematical expressions and computing various functions such as square roots, trigonometric functions, logarithms, and exponentials using custom algorithms.

The mathematical functions were implemented using iterative methods (e.g., Newton-Raphson for square root) and series expansions (e.g., Taylor series for trigonometric and exponential functions) to achieve reasonable accuracy without relying on the standard C math library.

Additionally, the LCD interface functions ensure clear and user-friendly display of results. The recursive expression evaluator enables the calculator to handle complex arithmetic expressions efficiently.

Overall, this project demonstrates how fundamental mathematical operations and expression parsing can be efficiently implemented on resource-constrained embedded systems like AVR microcontrollers.

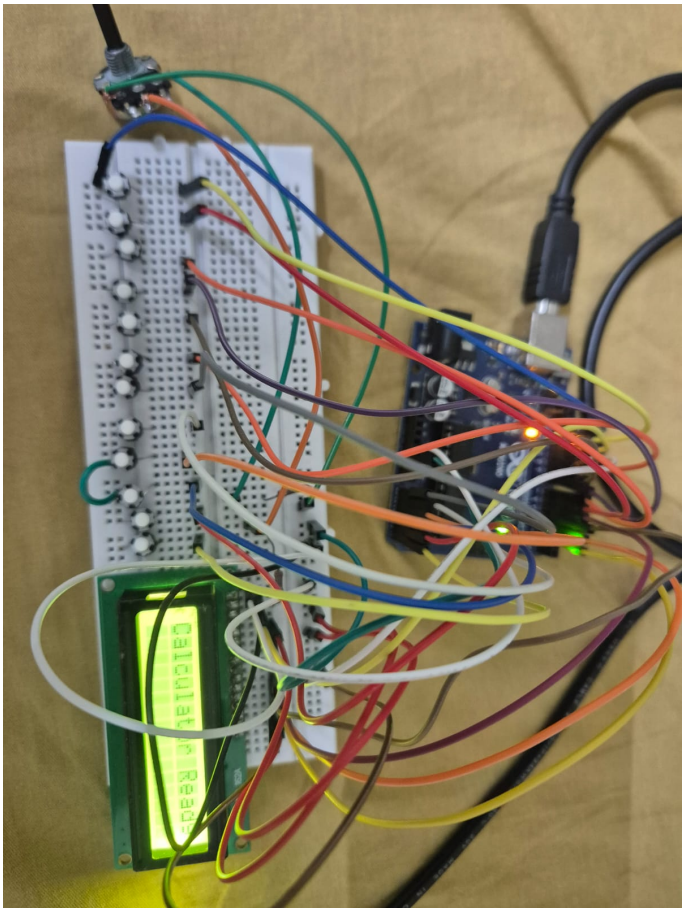

Fig. 0.2: Calculator