Scientific Calculator

John Bobby
ee24btech11032

1 Objective

The objective of this project is to design and implement a scientific calculator using an Arduino and push buttons. The calculator will be capable of performing basic arithmetic operations as well as evaluating trigonometric functions and their inverses. The implementation will be done using Embedded C.

2 Hardware Components

- Arduino
- 16 × 2LCD display
- 14 push buttons
- 10K potentiometer
- Jumper Wires

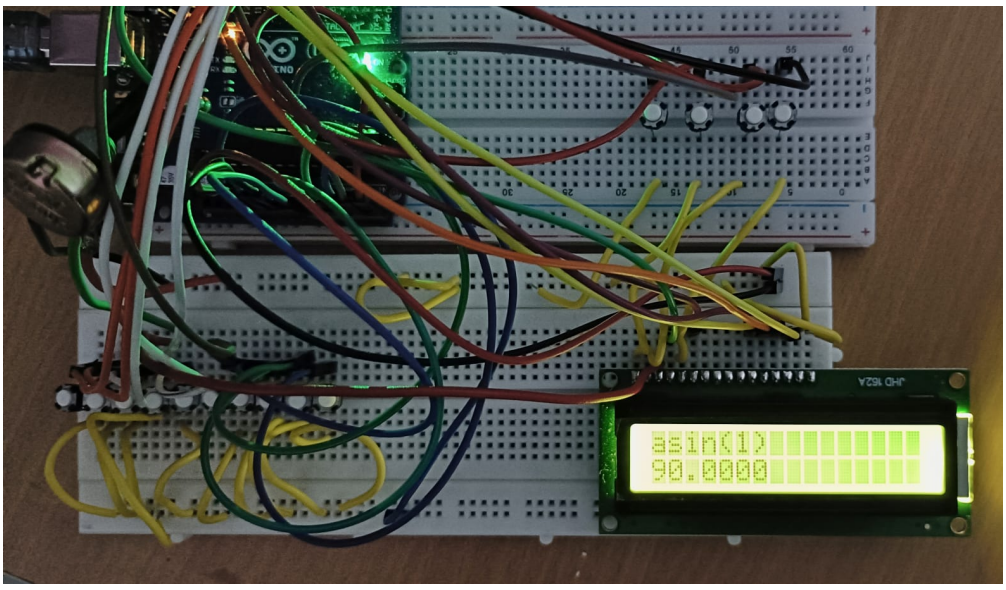

Figure 1: Calculator

3 Connections

Component	Arduino Pin	Function
LCD RS	12	Register Select
LCD E	11	Enable
LCD D4	10	Data Line D4
LCD D5	9	Data Line D5
LCD D6	8	Data Line D6
LCD D7	7	Data Line D7
Digit Button 0	2	Input for digit '0'
Digit Button 1	3	Input for digit '1'
Digit Button 2	4	Input for digit '2'
Digit Button 3	5	Input for digit '3'
Digit Button 4	6	Input for digit '4'
Digit Button 5	13	Input for digit '5'
Digit Button 6	A0	Input for digit '6'
Digit Button 7	A1	Input for digit '7'
Digit Button 8	A2	Input for digit '8'
Digit Button 9	A3	Input for digit '9'
Decimal Point Button	0	Input for decimal point
Binary Operation Button	A4	Select binary operator (+, -, *, /)
Equals Button	A5	Compute the result
Single Operation Button	1	Select single-operand function (sin, cos, etc.)

Table 1: Connections

4 Button Implementation and Detection

4.1 Button Circuit and Pin Configuration

- When the button is NOT pressed, the input reads HIGH due to the **internal pull-up** resistor.
- When the button is pressed, it connects to ground GND, and the input reads LOW.

This helps prevent floating states and ensures stable detection of button presses.

4.2 Debounce Logic for Button Presses

When we press a button, mechanical vibrations of the metal parts may give false reading.

- Read the button state (HIGH or LOW).
- If the state remains stable for a predefined debounce time (50 ms), confirm the button press.

5 Button Debounce in Code

```
uint8_t read_button(Button *b) {
    if ((*(b->pin) & (1 << b->bit)) == 0) {  // Button pressed (
       active low)
        _delay_ms(DEBOUNCE_DELAY_MS);        // Wait for debounce
           period
        if ((*(b->pin) & (1 << b->bit)) == 0) {  // Check if still
           pressed
            return 1;
        }
    }
    return 0;
}
```

Listing 1: Deboune Logic

- Checks if the button is pressed before and after the debounce time.

6 Digit Handling

```
if (read_button(&digitButtons[i])) {
    uint8_t digit = i;
    if (!decimalEntered) {
        currentNumber = currentNumber * 10 + digit;
    } else {
        currentNumber = currentNumber + digit * decimalMultiplier;
        decimalMultiplier *= 0.1;
    }
    append_digit(digit);
    lcd_clear();
    lcd_print(displayStr);
}
```

Listing 2: Digit Handling Logic

- A variable called decimalEntered is created which is True when we press the decimal point button.
- currentNumber variable is updated accordingly.
- All operations are handled with the help of if statements.

7 Conclusion

One of the key challenges in this project was ensuring **accurate button press detection**, which was efficiently handled using **debouncing techniques**.

Overall, this project demonstrates the feasibility of implementing a **low-cost, efficient, and user-friendly** calculator using microcontrollers. Future improvements could include **touch-based input**, **floating-point optimization**, or even **integration with external memory** to store previous calculations. This project not only reinforces fundamental concepts of **embedded systems** but also provides a foundation for more advanced digital electronics applications.