SCIENTIFIC CALCULATOR

By

K. AKHIL - EE24BTECH11035

March 24, 2025

Contents

I. Required Components

- Arduino Uno
- 16x2 LCD Display
- 10 Push Buttons (Digits 0-9)
- 2 Additional Push Buttons (Mode Selection)
- Potentiometer (For LCD Contrast Adjustment)
- Breadboard and Jumper Wires

II. Hardware Connections

Component	**Arduino Pin Connection**
LCD RS	Digital Pin 12
LCD EN	Digital Pin 11
LCD D4, D5, D6, D7	Digital Pins 5, 4, 3, 2
Numeric Buttons (0-9)	Digital Pins 6-10, A0-A4
Mode Selection Buttons	A5, D13
Potentiometer	LCD V0 (Contrast Control)

TABLE 0
Pin Connections for Arduino and Components

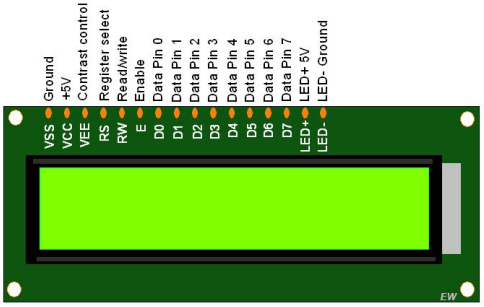

Fig. 0.1. LCD Display

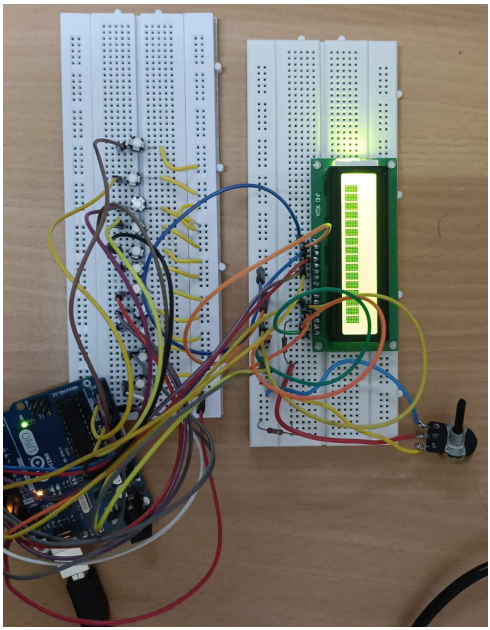

Fig. 0.2. Hardware

III. Working Explanation

- **LCD Driver:**
  - Implemented in `lcd.h`, it includes functions such as:
    - `lcd_init()`: Initializes the LCD.
    - `lcd_clear()`: Clears the LCD display.
    - `lcd_print()`: Prints a string or value on the LCD.
- **Button Handling:**
  - Manages the detection of button presses, including numeric inputs (0-9) and special function keys.
- **Mathematical Evaluation:**
  - Utilizes a recursive descent parser for parsing arithmetic expressions.
  - Supports operations such as addition, subtraction, multiplication, and division.
- **Custom Mathematical Functions:**
  - Approximates trigonometric (sin, cos) and logarithmic (log, ln) functions using iterative methods.

Button Mapping and Modes

Buttons facilitate numeric input and access various mathematical operations. The calculator operates in three modes:

- **Normal Mode:**
  - Allows entry of digits (0-9) and basic operations: +, −, ∗, /, =, and `Backspace`.
- **Shift Mode:**
  - Provides access to advanced mathematical functions such as:
    - sin, cos, e^x, and $\sqrt{x}$.
- **Extra Mode:**
  - Offers extended functionality with access to:
    - arcsin, arccos, arctan.
    - Logarithmic functions: log, ln.

IV. Code Outline Explanation

A. Library Inclusions

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "lcd.h"
```

Explanation:

- `avr/io.h`: Provides access to microcontroller registers.
- `util/delay.h`: Used for generating delays.
- `stdbool.h`: Provides Boolean logic support.
- `math.h`: Standard math library functions.
- `lcd.h`: Custom LCD driver for display operations.

B. Mathematical Functions Implementation

Custom iterative approximations are used for trigonometric and logarithmic functions.

1) Sine Approximation (Taylor Series):

```
float mySin(float x){
float rad = x * PI / 180.0;
float term = rad;
float sum = rad;
for (int n = 1; n < 10; n++){
term = -term * rad * rad / ((2 * n) * (2 * n + 1));
sum += term;
}
return sum;
}
```

2) Cos function:

```
float myCos(float x){
float rad = x * PI / 180.0;
float term = 1;
float sum = 1;
for (int n = 1; n < 10; n++){
    term = -term * rad * rad / ((2 * n - 1) * (2 * n));
```

```
    sum += term;
  }
  return sum;
}
```

3) Square Root Function using Newton method:

```
float mySqrt(float x){
  if (x < 0) return -1;
  float guess = x / 2.0;
  for (int i = 0; i < 10; i++){
    guess = (guess + x / guess) / 2.0;
  }
  return guess;
}
```

4) Natural logarithm:

```
float mySqrt(float x){
  if (x < 0) return -1;
  float guess = x / 2.0;
  for (int i = 0; i < 10; i++){
    guess = (guess + x / guess) / 2.0;
  }
  return guess;
}

}
```

5) Exponential Approximation (Euler's Method):

```
float myExp(float x){
int N = 100;
float h = x / N;
float y = 1.0;
for (int i = 0; i < N; i++){
y = y * (1 + h);
}
return y;
}
```

C. Expression Parsing (Recursive Descent)

Expressions are evaluated using a recursive descent parser.

```
float parseExpression(const char* s, int *pos) {
skipSpaces(s, pos);
float value = parseTerm(s, pos);
skipSpaces(s, pos);
while (s[*pos] == '+' || s[*pos] == '-') {
char op = s[(*pos)++];
float term = parseTerm(s, pos);
if (op == '+') value += term;
else value -= term;
skipSpaces(s, pos);
}
return value;
}
```

Parses and evaluates arithmetic expressions recursively.

D. Button Handling

Each button press is detected and processed accordingly.

```
bool button_pressed(uint8_t pinMask) {
return !(BUTTON_PORT & pinMask);
}
```

Checks if a button is pressed based on its pin state.

E. LCD Update

The LCD is updated with the current input string.

```
void updateLCD(void) {
lcd_clear();
lcd_print(input);
}
```

Clears and refreshes the display with new input.

F. Main Execution Loop

The program runs continuously, reading button inputs and updating the LCD.

```
int main(void) {
lcd_init();
lcd_clear();
lcd_print("Calculator␣Ready");
_delay_ms(1000);
lcd_clear();

while (1) {
// Read and process button inputs
updateLCD();
_delay_ms(10);
}
return 0;
}
```

Initializes the LCD and enters a loop to handle user inputs.

V. Conclusion

This project successfully implements a scientific calculator for the AVR microcontroller, allowing users to input expressions via buttons and view results on an LCD display. The key features include:

- Support for basic arithmetic and advanced mathematical functions.
- A recursive descent parser for accurate expression evaluation.
- Custom approximations for trigonometric, logarithmic, and exponential functions.

Future enhancements could include:

- Improving numerical accuracy by increasing iterations in Taylor series expansions.
- Expanding functionality to include additional mathematical functions.
- Optimizing memory usage for improved performance on low-memory microcontrollers.

VI. References

- AI Suggestions
- Hardware connections guide- Online Sites