COURSE PROJECT

EE1003

SCIENTIFIC PROGRAMMING

Scientific Calculator

Contents

1 Introduction

A scientific calculator does arithematic operations and also has mathematical funtions. I have built a calculator that has the following functions:

1) Basic arithematic operations
2) Trignometric (sin, cos, tan)
3) Inverse Trignometric ($\sin^{-1}, cos^{-1}, tan^{-1}$)
4) Logarithm (base 10, e)
5) $e^{\square}$ and $10^{\square}$
6) Power
7) Factorial
8) Square root

It has π and e as values of input along with the numbers and the decimal place.

The calculator interprets the expression with the help of a recursive decent parser and calculates values of functions using CORDIC algorithms.

2 Hardware Implementation

Component	Quantity	Description
Arduino	1	Microcontroller board for programming and controlling electronics
Push Buttons	30	Momentary switches for user input and digital signal triggering
Diodes	30	Electrical components that allow current to flow in one direction
16x2 LCD	1	Display module with 2 rows and 16 characters per row for output

TABLE 8: Components required

I made a button array containing 30 buttons for input. Output was done using a 16x2 LCD. The processing is done using an Ardiono Uno.

I used multiplexing of buttons to use the button array. See table 8 for list of components, fig. 8 for the schematic.

3 Software Implementation

The software can be split into two parts

1) Hardware Driver
2) Parser algorithm
3) Calculator

3.1 Hardware Driver

I used Prof. GVV's LCD driver code to drive the LCD. For using the button array, I used multiplexing, as metioned before. This process works the following way. One of the pins is set to `PINOUT` and set to `LOW` and rest are set to `PIN_PULLUP`. Then loop through

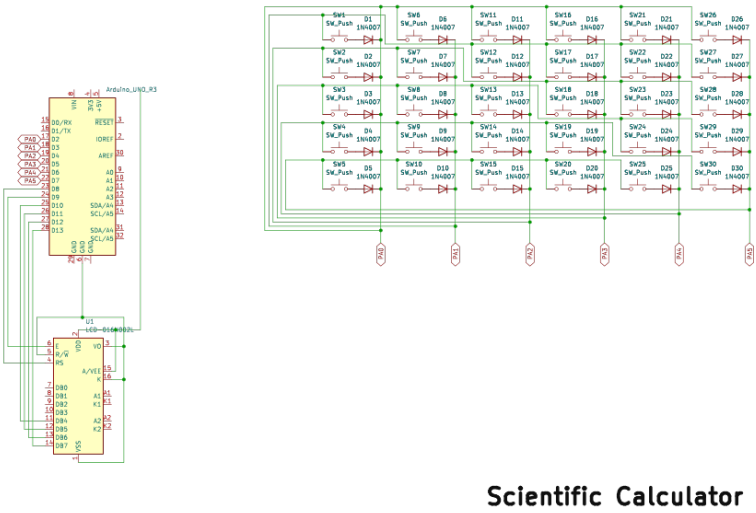

Fig. 8: Circuit diagram for the calculator

the `PIN_PULLUP` pins and check if it is `LOW`. If it is, then it detected as a press. Then, it sets the next pin to `PINOUT` and sets it to `LOW` and the rest of the pins as `PIN_PULLUP`, and this process is repeated until all the buttons are checked. Then it starts over again after processing the button presses. By using n pins of the arduino, we can connect $n(n+1)$ buttons to the board. Here we use 6 pins, thus giving us ability to use 30 buttons.

3.2 Parser algorithm

To parse the expression entered, I used `tinyexpr.h`. It is a recursive decent parser developed by Lewis Van Winkle (Github username: codeplea). Using this, I was able to parse the entered expression efficently and it was small enough to fit into Arduino's flash memory.

I had to make some changes from the actual parser to make it compactible with our use case and compress the filesize further.

3.3 Calculator

I tried implementing numerical methods for the calculator, but there were couple issues I faced.

1) When the step size was a bit too large, the values given weren't accurate enough.
2) Smaller step sizes took a while to compute, especially for bigger inputs.

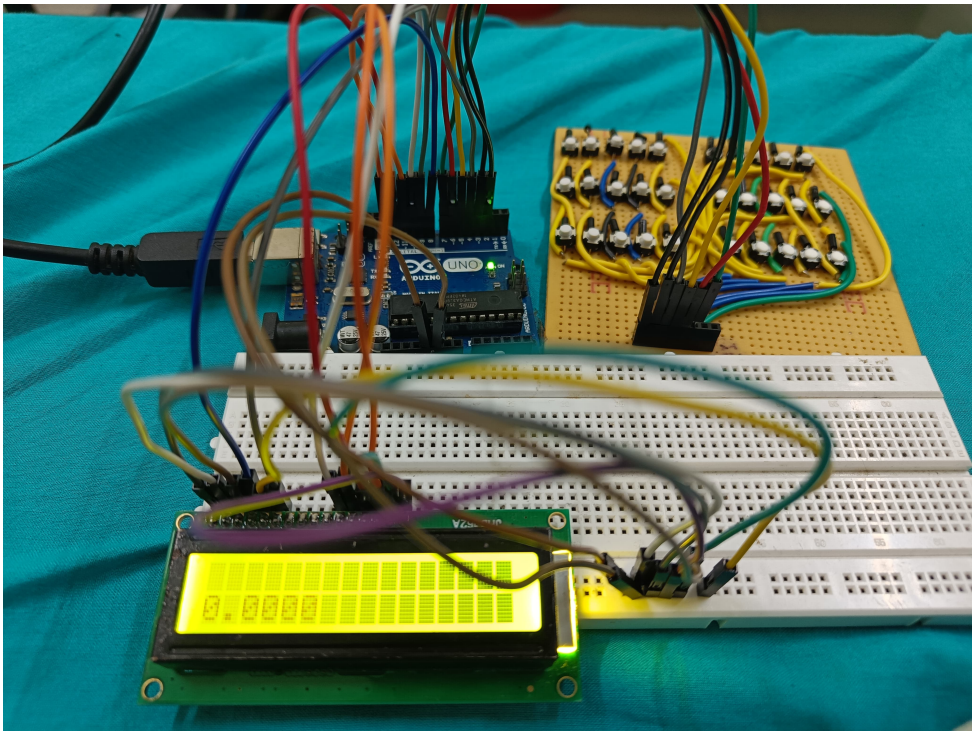

Fig. 8: Physical Build

For those reasons, and to learn more about how calculators actually work, I decided to look into CORDIC algorithms.

CORDIC stands for - COordinate Rotation DIgial Computer. There are different variations of this algorithm to compute Trignometric functions, Hyperbolic functions, square roots, multiplications, divisions and such.

Advantage of this algorithm is, they are optimised for low-level finite state machines and can also be implemented as a physical system.

It's basic principle is rotation of vector, by a given known value and get our required output.

3.4 Codes

Main Driver Function:

./code/main.c

TinyExpr:

./code/tinyexpr.h ./code/tinyexpr.c

Math Functions:

./code/mathN.h