Scientific Calculator Using Arduino

Developer: Jaafar Saleh

Project Overview

This calculator can compute all trigonometric ratios (24 functions), logarithmic values (common and natural), and several practical operations such as exponentiation, factorial, and square root. The displayed result is rounded to the nearest hundredth, but the internal calculations maintain full precision.

Equipment Used

- **Arduino Uno** board
- **Liquid Crystal Display (LCD)**
- **Push-button Matrix** (6×6 keypad)
- **Connecting Wires/Pins** (hidden wiring)
- **Labeled Buttons** for easy identification

Hardware Connections

- All **analog pins** on the Arduino are connected to the keypad rows.
- Digital **I/O pins (1–6)** are assigned to keypad columns.
- **LCD Connections:**
  - RS → IO13
  - E → IO12
  - (D4…D7) → (IO11…IO8)
  - VDD → 5V power supply
  - Other pins → Ground

Concept Behind the Project

The Arduino has a **central processing unit**, so why not treat it as an independent computer? The first program that comes to mind for such a computer is a **calculator**. However, treating the Arduino like this introduced several challenges, which are discussed below.

Normally, the Arduino is used for **automation** or **integration** into real-world systems, such as:

- **Fire alarm systems**, which activate only when smoke is detected.
- **Automated irrigation**, using soil moisture sensors that trigger watering when needed.

What If I Designed a Calculator Using Arduino?

Creating a calculator involves **more complexities** than simply reading sensor values and reacting. In a calculator, every **button press** sends data to the Arduino while simultaneously receiving immediate feedback, displayed on the screen.

What If This Calculator Were Scientific?

This idea was **entirely new**! After extensive online research, I realized there were **no scientific calculator projects** available to learn from. This marked the beginning of a **unique challenge**.

Challenges and Difficulties

Challenge 1: The Keypad Matrix

Commercial keypads typically come in predefined sizes **(4×4 or 6×4)**, which is barely enough for a **basic calculator**. To overcome this, I modified the **default keypad in Proteus** to create a **custom 4×8 keypad**.

However, since such keypads are **not available commercially**, I decided to **build my own** using basic **push-buttons**—which led to **Challenge 2**.

Challenge 2: Custom Button Matrix Construction

The button matrix consists of **separate rows and columns**:

- Each **row** contains **six buttons** wired in parallel.
- Each **column** contains **six buttons** wired in parallel.
- Each button acts as a **node**, connecting a row to a column.

In this setup:

- **Rows** are connected to **analog Arduino pins**, providing continuous high voltage.
- **Columns** are connected to **digital I/O pins**, read using the **keypad.h** library.

Since the keypad **only accommodates 36 buttons**, but **24 trigonometric functions** were needed, I introduced **modifier keys**:

- Pressing arc before sin converts it into **arcsin**.
- Pressing hyp before cos converts it into **cosh (hyperbolic cosine)**.
- Pressing both arc and hyp before cot converts it into **arccoth (arc hyperbolic cotangent)**.

This system mirrors the **Shift** and **Alpha** keys in Casio calculators.

Challenge 3: Display Management

The chosen **LCD screen** (**LM-016**) was available in university labs, making practical application possible. However, unlike a **console or CMD screen**, this LCD lacked **built-in text handling features**. As a result, I had to program **manual cursor positioning**, leading to a codebase of nearly **1000 lines**.

Challenge 4: Complex Code Structure

Without a **console**, every button press required **immediate logic execution** to store input in memory variables like **double num1**.

Additionally, most online Arduino tutorials rely on **pre-made libraries**, but I wanted a **custom solution**. Proteus allowed **uploading Arduino firmware** without external libraries, so I opted for a **bare-metal Arduino Uno setup**.

Challenge 5: Arduino Firmware Handling

Using an **external Arduino setup** vs. **Proteus emulation** posed another hurdle. I experimented with both configurations to find the most efficient approach for this project.