SCIENTIFIC CALCULATOR

EE24BTECH11023 - RASAGNA

1 Objective

- The objective of this project is to design and implement a scientific calculator using an LCD and AVR-GCC on an Arduino microcontroller.
- The calculator will be capable of performing basic arithmetic operations, trigonometric functions, logarithmic calculations, and other scientific computations.
- The project aims to develop an efficient, user-friendly interface for input and output, utilizing an LCD for display and a keypad for user interaction.
- Through this project, we explore the integration of embedded systems with mathematical operations, optimizing performance using AVR-GCC for precise and efficient computation.

2 Components and Equipment

S.No	Component	Quantity
1	Arduino	1
2	Breadboard	2
3	LCD display	1
4	USB A to USB B cable	1
5	OTG adapter	1
6	Jumper wires (Male-Male)	50
7	Resistors 15 000 Ω	9
8	Push Buttons	25

- **VSS** → GND (Ground)
- **VDD** → 5V (Power Supply)
- **V0** → 10k Potentiometer (middle pin) (Contrast Adjustment)
- **RS** → D7 (Register Select)
- **RW** → GND (Read/Write, set to Write)
- **E** → D6 (Enable)
- **D4** → D4 (Data Bit 4)
- **D5** → D5 (Data Bit 5)
- **D6** → D6 (Data Bit 6)
- **D7** → D7 (Data Bit 7)
- **A (LED+)** → 5V with a Resistor (Backlight Power)
- **K (LED-)** → GND (Backlight Ground)

3 Working Principle-Software Implementation

3.1 Software implementation using ADC Values

The scientific calculator utilizes the ADC (Analog-to-Digital Converter) of the Arduino microcontroller to read analog inputs, such as a voltage divider-based keypad. The ADC converts the analog input into a digital value, which is then processed to determine the corresponding keypress.

The ADC is configured in free-running mode with a prescaler to ensure accurate readings.

- **ADC Initialization:** Configure the ADC in the AVR-GCC environment by setting the appropriate registers.
- **Reading ADC Values:** Continuously read the ADC values from the analog pin where the keypad is connected.
- **Mapping ADC Values to Keypad Inputs:** Compare the ADC readings against predefined threshold values to determine which key is pressed.
- **Processing Key Inputs:** Perform mathematical operations based on the detected keypress and update the LCD accordingly.
- **Displaying Results:** Output the computed result on the LCD using 4-bit mode communication.

3.2 Debouncing

1) Mechanical buttons often generate noisy signals, causing bouncing, where the button state might change rapidly due to physical contact. This can cause multiple increments instead of just one.
2) To prevent this, a debounce delay is added, which ensures that only one increment happens for each button press (or jumper wire tap).
3) After detecting a state change, the program waits for a short period (e.g., 300 ms) before checking the button state again.
4) This debounce delay helps ignore any unintended multiple presses from the same action.

4 Advantages of AVR-GCC

- Direct access to AVR registers which results in smaller and faster machine code.
- No C++ overhead (such as classes, virtual functions, and dynamic memory allocation).
- We can directly manipulate registers (e.g., PORTB, DDRC) for faster execution.
- No need for functions like digitalWrite(), which are slower than direct register access.
- No unnecessary code from high-level abstractions or unused libraries.
- Avoids unexpected behavior caused by library overhead.
- AVR-GCC is faster, more efficient, and gives complete hardware control.

5 Precautions

5.1 Hardware

1) Always connect a Resistor between Backlight pin of LCD and 5V.
2) Connect the RW pin of the LCD to GND to keep it in write mode.
3) Use proper grounding to prevent noise and ensure stable operation.

5.2 Software

1) Verify correct pin assignments in the code to match hardware connections.
2) Use appropriate delays after sending commands to allow the LCD to process them.
3) Ensure proper handling of buffer overflows when processing user input.