Step-by-Step Guide for Measuring 0 to 100V DC Using a Microcontroller

1. Select a Microcontroller

Choose a microcontroller suitable for your requirements. Popular options include:

- **Arduino Uno (ATmega328P)**
- **ESP32**
- **PIC16F877A**

2. Select Suitable Compiler for Embedded C

Depending on your microcontroller, choose an appropriate compiler:

- **Arduino IDE** for Arduino boards
- **ESP-IDF** for ESP32
- **MPLAB X IDE** for PIC microcontrollers

3. Write Algorithm for Measurement

Draft an algorithm to measure 0 to 100V DC. The process generally involves:

- **Voltage Divider**: Use resistors to scale down the voltage to a safe level for the microcontroller ADC (Analog-to-Digital Converter).
- **Analog Reading**: Use the ADC to read the scaled voltage.
- **Calculation**: Convert the ADC value back to the original voltage.

Algorithm Example:

1. **Initialize ADC**.
2. **Read the analog input**.
3. **Convert the ADC value to voltage** using the voltage divider formula.
4. **Display or store the voltage value**.

4.Flow chart

↓
Initialize ADC
↓
Read ADC Value
↓
Convert ADC Value to Voltage
↓
Display/Store Voltage
↓

1. Write C Code Starting from

Mains #include <xc.h>

```
// Configuration bits
#pragma config FOSC = INTRC_NOCLKOUT // Oscillator Selection bits
#pragma config WDTE = OFF           // Watchdog Timer Enable bit #pragma
config PWRTE = OFF                  // Power-up Timer Enable bit #pragma
config MCLRE = ON                   // RE3/MCLR pin function select bit
#pragma config CP = OFF            // Code Protection bit
#pragma config CPD = OFF           // Data Code Protection bit #pragma
config BOREN = ON                   // Brown Out Reset Selection bits
#pragma config IESO = OFF          // Internal External Switchover bit
#pragma config FCMEN = OFF          //  Fail-Safe  Clock  Monitor  Enable  bit
#pragma config LVP = OFF           //  Low  Voltage  Programming  Enable  bit
#pragma config BOR4V = BOR40V   // Brown-out Reset Selection bit
#pragma config WRT = OFF           // Flash Program Memory Self Write Enable bits

#define _XTAL_FREQ 4000000          // Define crystal frequency
#define VOLTAGE_PIN 0              // Analog input pin (AN0)

// Function prototypes void
initADC(); unsigned int
readADC();
float calculateVoltage(unsigned int adcValue);

// Main function void
main(void) {
   unsigned int adcValue;
   float voltage;

   // Initialize ADC initADC();

   // Infinite loop while
   (1) {
      // Read ADC value
      adcValue = readADC();
      // Calculate voltage
      voltage = calculateVoltage(adcValue);
      // Add your display or logging code here
```

```
        // For example, send the voltage value to UART (assuming UART is initialized)
        // printf("Voltage: %.2fV\n", voltage);

        _delay_ms(1000); // Delay 1 second
    }
}

// Initialize ADC void
initADC() {
    ADCON0 = 0x41; // Turn on ADC and set clock
    ADCON1 = 0x80; // Right justify result, use VDD and VSS as reference
}
// Read ADC value
unsigned int readADC() {
    ADCON0 |= 0x02; // Start conversion
    while (ADCON0 & 0x02); // Wait for conversion to complete return
    ((ADRESH << 8) + ADRESL); // Return 10-bit ADC value
}

// Calculate voltage from ADC value
float calculateVoltage(unsigned int adcValue) { float
    voltage;
    voltage = adcValue * (5.0 / 1023.0); // Convert ADC value to voltage
    // Adjust according to your voltage divider
    voltage = voltage * ((R1 + R2) / R2); return
    voltage;
}
```

1. Calculate Achievable Theoretical Accuracy

The accuracy depends on the ADC resolution and the voltage reference used. For an Arduino with a 10-bit ADC:

Resolution: 5V / 1024 = 4.88mV Accuracy:
±1 bit or ±4.88mV
Consider voltage divider inaccuracies and noise in your calculations.