To measure 0 to 100V DC using a microcontroller pic16F877A

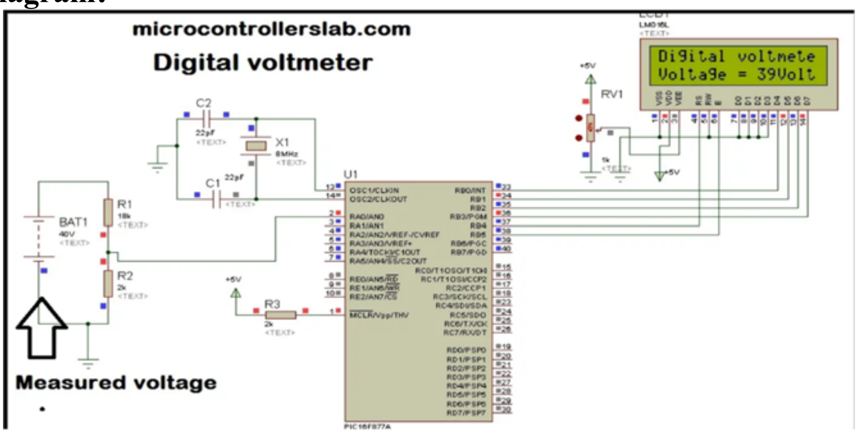
You can use a voltage divider circuit to step down the voltage to a range that the microcontroller's ADC (Analog-to-Digital Converter) can handle. The **pic16F877A** ADC typically operates at a maximum input voltage of 5.0V.

Here's a step-by-step guide on how to do this:

1. **Design the Voltage Divider:**
   - A voltage divider uses two resistors to scale down the input voltage to a lower voltage. The voltage divider formula is:

$$\mathbf{Vout=Vin\times (R2\ )/(R1+R2)}$$

   - For example, to scale down 100V to 3.3V:
   - 3.3V=100V×R2 /(R1+R2)
   - Solving for R2/(R1+R2)
   - R2/(R1+R2) = 3.3/100 =0.033
2. **Choose Resistor Values:**
   - Let's choose R2=3.3kΩ
   - Solving for R1:
   - R1= (3.3K/0.033) = 96.7K ohm
3. **Connect the Voltage Divider:**
   - Connect the high voltage (0 to 100V) to the input of the voltage divider.
   - Connect the junction of the two resistors to an ADC pin of the pic16F877A.
   - Ensure the ground of the high voltage source is common with the PIC ground.
4. **Protect the Microcontroller:**
   - To protect the ADC pin from over-voltage, you can add a Zener diode (e.g., 5.0V) across the ADC input and ground.
   - You can also add a small capacitor (e.g., 100nF) across the ADC input to filter out noise.
5. **Ciruit Diagram:**



Write the Code:

Here is code snippet for measuring the voltage using the PIC16F877A in MPLAB X IDE with XC8 compiler:

```
 #include <xc.h>
#inlude “clcd.h”
#define _XTAL_FREQ 20000000    // Define oscillator frequency for delay

void ADC_Init()
{
   ADFM=1; // Selecting Right Justification
   ADON= 1; // Starting the ADC Module
}


unsigned int ADC_Read(unsigned char channel)
{
   if(channel > 7)
   {
       return 0;  // ADC has 8 channels, 0-7
   }
   ADCON0 =ADCON0 & 0xC5;     // Clear existing channel selection bits
   ADCON0 =(ADCON0 &0XC7) | (channel << 3);  // Set new channel
   __delay_ms(2);  // Acquisition time to charge the hold capacitor
   GO= 1;  // Start ADC conversion
   while(GO==1);  // Wait for conversion to complete
   return (ADRESL | (ADRESH << 8) );  // Return result
}
void display(unsigned float adc_reg_val_1)
{
      char buff[5];
      int i;
      i = 3; // buff: "1 0 2 3"
     do
    {
         buff[i] = (adc_reg_val_1 % 10) + '0';
         adc_reg_val_1 = adc_reg_val_1 / 10;
     } while (i--);
     buff[4] = '\0';

      clcd_print(buff, LINE1(0));
}
```

```
void main()
{
   unsigned int adc_value;
   float voltage, input_voltage;



   TRISA = 0xFF;  // Configure PORTA as input
   ADC_Init();  // Initialize ADC

   while(1)
   {
      adc_value = ADC_Read(0);  // Read ADC value from channel 0
      voltage = (adc_value * 5.0) / 1023.0;  // Assuming Vref+ is 5V and 10-bit ADC resolution
      input_voltage = voltage * ((100.0 + 3.3) / 3.3);  // Convert to input voltage

      // Now you can use input_voltage as needed
      __delay_ms(500);  // Delay for a while
      display(input_voltage);
   }
}
```

Explanation of the Code:

1. **Initialization:**
   - `ADC_Init()`: Initializes the ADC module.
     - `ADCON0`: Configures ADCON0 register to turn on the ADC and set the conversion clock.
     - `ADCON1`: Configures ADC voltage reference and result format.
2. **Reading ADC Value:**
   - `ADC_Read()`: Reads the ADC value from the specified channel.
     - `ADCON0` configuration ensures correct channel selection.
     - `GO_nDONE`: Starts ADC conversion and waits for completion.
3. **Voltage Calculation:**
   - Convert the ADC value to voltage (`voltage`).
   - Calculate the actual input voltage using the voltage divider ratio (`input_voltage`).
4. **Loop:**
   - Continuously read and calculate the voltage in the `while` loop.
5. **Display:**

   Display the actual input voltage in clcd