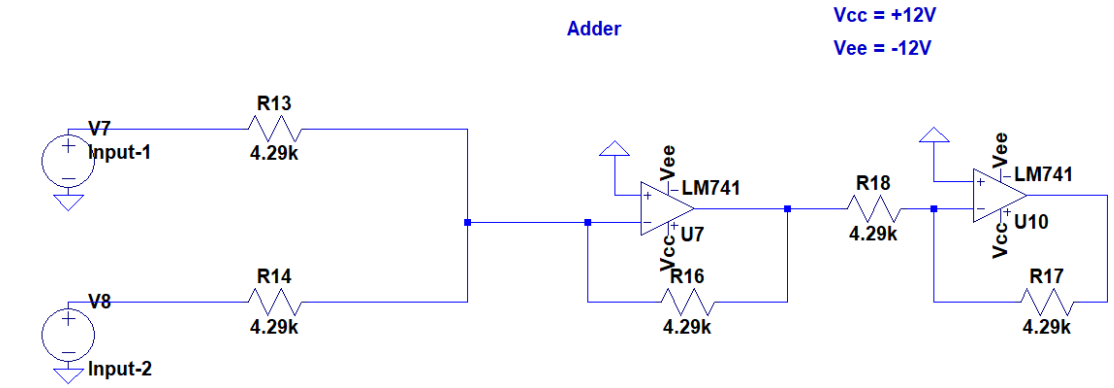
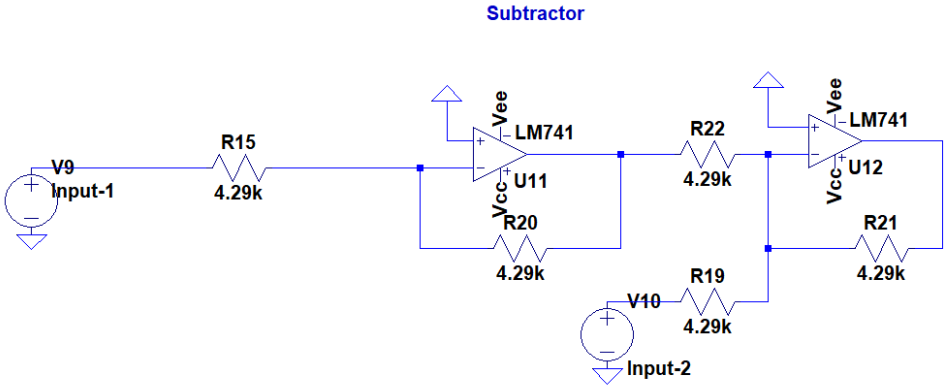
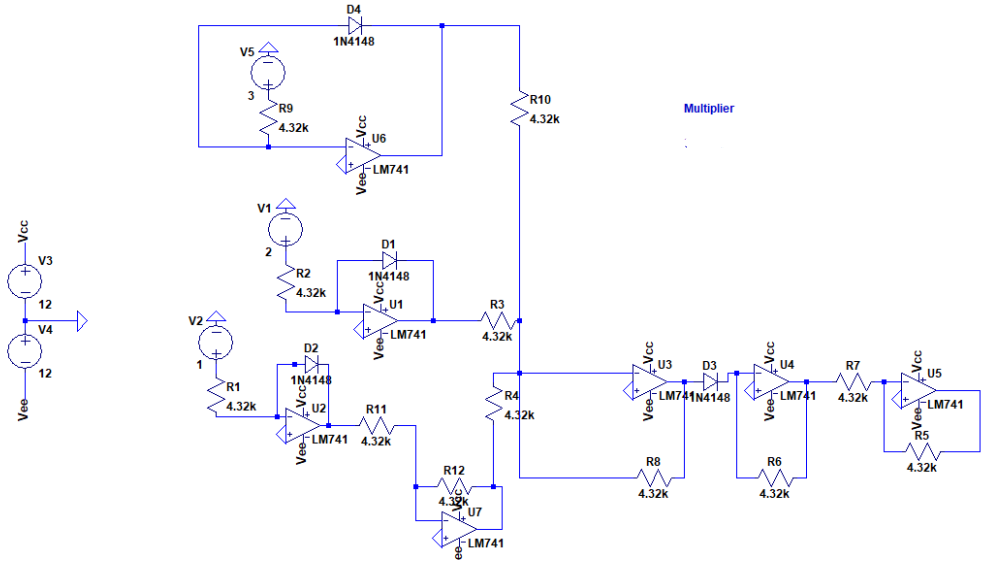
Circuit Schematics :

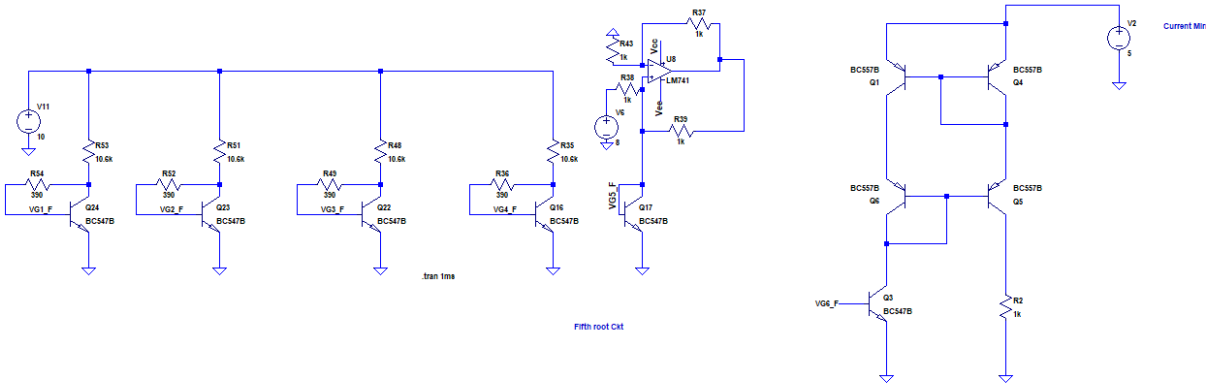




The multiplier circuit as shown below incorporates 3 inputs : V1 V2 and Vref the diagram shown below has V1=3V V2=2V and Vref=1V (always) . The output will be (V1*V2)/Vref so here we have the output always as V1*V2 giving us multiplication operation.



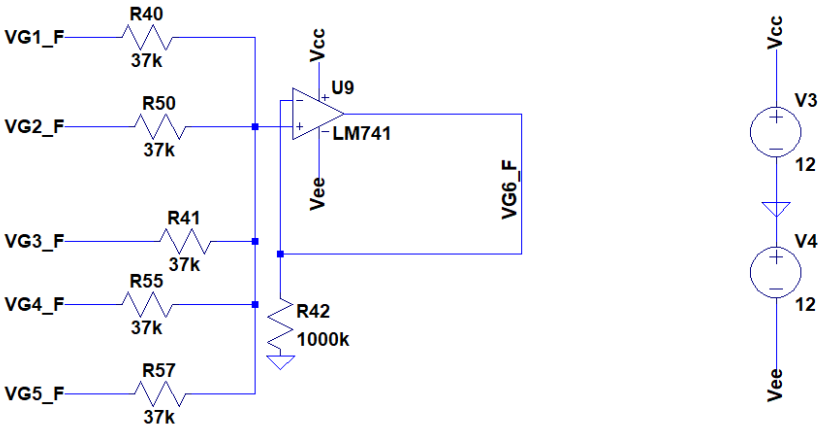
Same idea using BJTs but due to sensitive operations of BJT we improved the design to diode based with same idea. Both of them are scalable but diode one being more practical to implement.



The design exploits the exponential dependence of current and voltage to get the generalised the nth root of a number given as an input to the BJT Q17 in V6 in the above schematic.

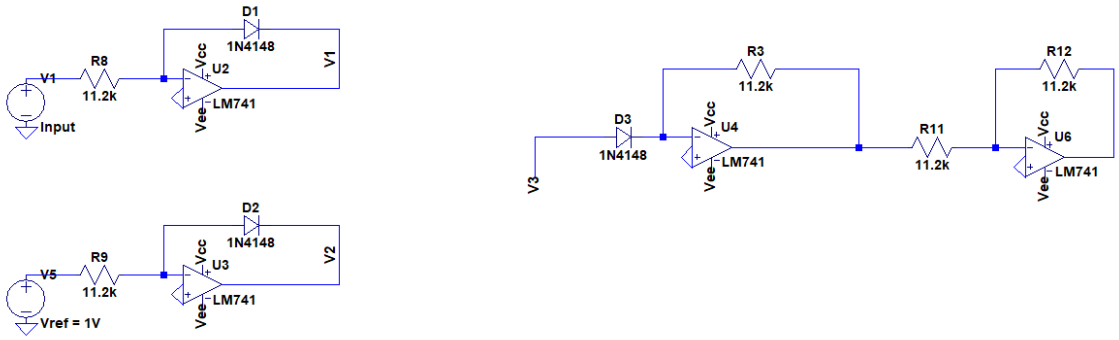
The output is taken as the voltage across the 1k ohm resistor in the circuit shown above which is the required nth-root of the input number.

Depending on what root we have to find we can add stages of the identical transistors starting from the right , and adding appropriate inputs in the average unit shown:



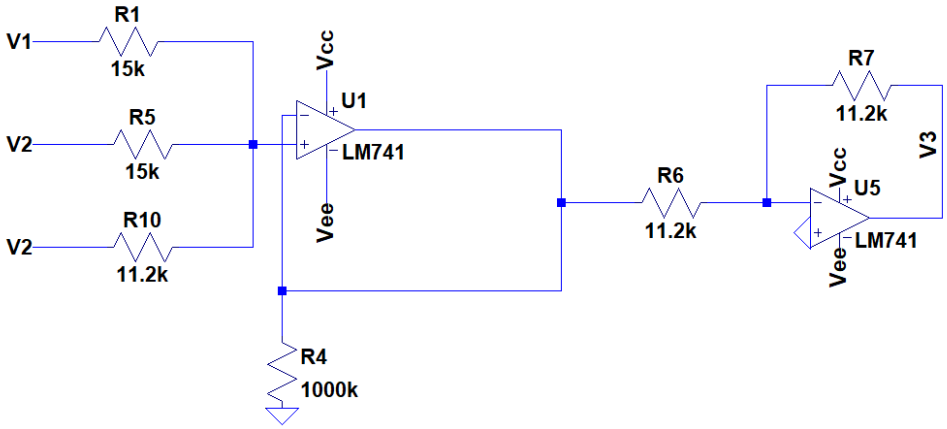
For simplicity this design can be converted to an diode equivalent design as shown which is also a generalised nth root.

Here we have an input voltage , and depending on what root we have to calculate we may increase the number of stages in the averager as shown.



Let say we need to evaluate the third root of 8 : we give 8V as input voltage as shown above and incorporate the average as shown with V1 being input and V2 V3 being the input reference voltage 1V.

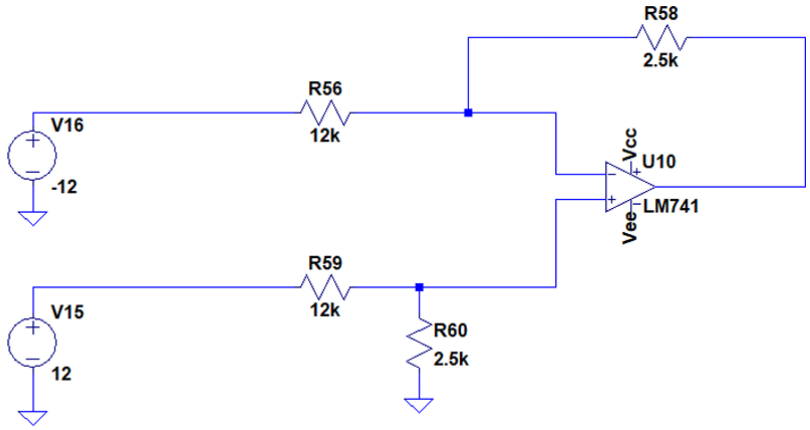
The output of the average is negative so it is fed to an inverter is placed to make it positive and fed to another diode as shown above. Since the output is again negative an inverter is placed after it to take the final output as output of op-amp U6.



Finally 2 more circuits are there to scale and de-scale the circuit output (-12V to 12V) to the microcontroller's acceptable range (0-5V)

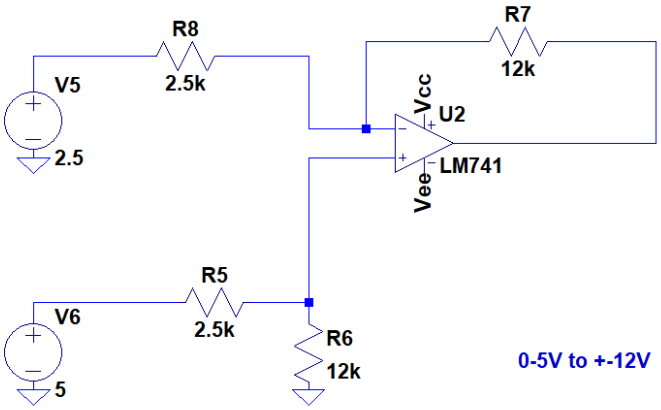
+-12V back to 0-5V

V16 is a fix reference input at -12V and V15 is the input signal.

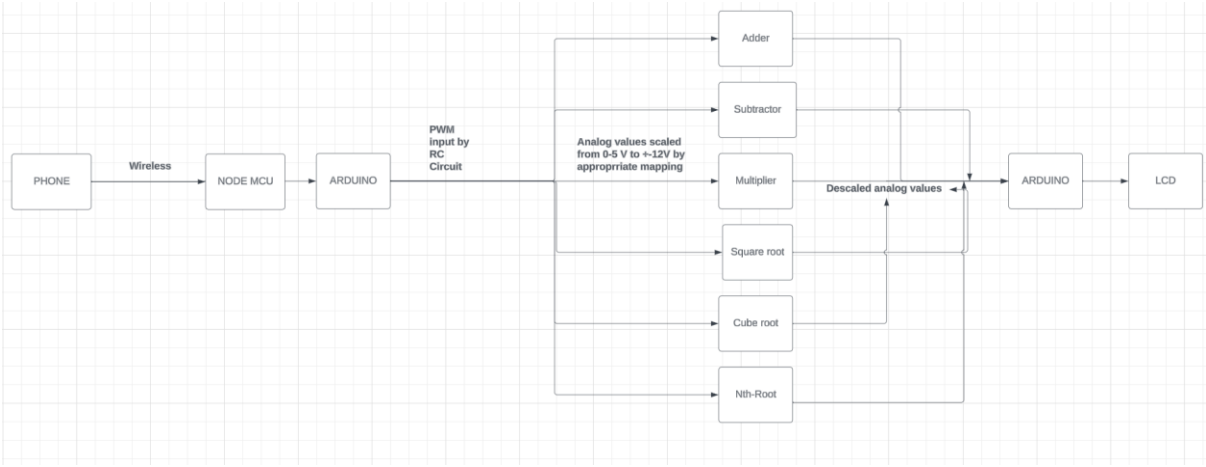


0-5V to +-12V

V6 is the variable input and V5 is a fixed reference voltage at 2.5V



Micro-controller Code used to evaluate the expression as expected and corresponding block diagram demonstrating communication between circuit and micro-controller.



Code for communication between Arduino and analog circuit :

```
#include <Arduino.h>
#include <StackArray.h> // Include Arduino Stack library for stack implementation

// Function to find precedence of operators.
int precedence(char op){
  if(op == '+' || op == '-')
    return 1;
```

```
 if(op == '*')
   return 2;
 if(op == '^') return 3;
 return 0;
}

// Function to perform arithmetic operations.
// Modified to Perform Arithmetic Operations and Read Analog Value 10 times for Averaging
float applyOp(float a, float b, char op) {
 int pwm1 = 255 * a / 5; // Convert input a to PWM value
 int pwm2 = 255 * b / 5; // Convert input b to PWM value
 int pin;

 if (op=='^')
 {
   pwm2 = 51;
   if (b==0.5)
     pin = A3;
   else if (b==0.3)
     pin = A4;
   else if (b==0.2)
     pin = A5;
 }

 // Write PWM values to the corresponding pins
 analogWrite(6, pwm1);
 analogWrite(9, pwm2);

 // Introduce a delay to allow the circuit to stabilize
 delay(5000); // Example delay, adjust based on operation

 float sumAnalogValue = 0.0; // Initialize variable to accumulate analog read values
 int readCount = 10; // Number of readings to take

 // Loop to read analog value 10 times
 for(int i = 0; i < readCount; i++) {
   delay(30);
   // Read and accumulate analog value from the appropriate pin based on operation
   switch (op) {
     case '+':
       sumAnalogValue += ((float)analogRead(A0))*5/1023; // Read analog value from pin A0
       break;
     case '-':
       sumAnalogValue += ((float)analogRead(A1))*5/1023; // Read analog value from pin A1
       break;
     case '*':
       sumAnalogValue += ((float)analogRead(A2))*5/1023; // Read analog value from pin A2
       break;
     case '^':
```

```
      sumAnalogValue += ((float)analogRead(pin))*5/1023; // Read analog value from the designated pin based on 'b' value
      break;
    }
  }

  // Calculate the average of the analog values
  float averageAnalogValue = sumAnalogValue / readCount;

  // Return the averaged analog value
  return averageAnalogValue;
}



//float version of isdigit
bool isFloat(String value) {
  bool containsDecimal = false;

  // Check if the string is empty
  if (value.length() == 0) {
    return false;
  }

  // Iterate through each character
  for (int i = 0; i < value.length(); i++) {
    // Check for an operation character
    if (value[i] == '+' || value[i] == '-' || value[i] == '*' || value[i] == '/') {
      return false;
    }
    // Check for a decimal point
    if (value[i] == '.') {
      // If already found a decimal point, it's not a valid float
      if (containsDecimal) {
        return false;
      }
      containsDecimal = true;
    }
    // Check for a digit or negative sign
    else if (!isdigit(value[i]) && value[i] != '-') {
      return false;
    }
  }

  return true;
}

// Function that returns value of expression after evaluation.
float evaluate(String tokens){
```

```
int i;

// stack to store integer values.
StackArray<float> values;

// stack to store operators.
StackArray<char> ops;

for(i = 0; i < tokens.length(); i++){

 // Current token is a whitespace, skip it.
 if(tokens[i] == ' ')
  continue;

 // Current token is an opening brace, push it to 'ops'
 else if(tokens[i] == '('){
  ops.push(tokens[i]);
 }

 // Current token is a number, push it to stack for numbers.
 else if(isFloat(String(tokens[i]))){
  float val = 0;
  // There may be more than one digit in number.
  String x="";
  while(i < tokens.length() && isFloat(String(tokens[i]))){
   // Serial.println(i);
   x+=tokens[i];
   // val = (val * 10) + (tokens[i] - '0');
   i++;
  }
   val=x.toFloat();

  values.push(val);

  // right now the i points to the character next to the digit,
  // since the for loop also increases the i, we would skip one
  // token position; we need to decrease the value of i by 1 to
  // correct the offset.
  i--;
 }

 // Closing brace encountered, solve entire brace.
 else if(tokens[i] == ')'){
  while(!ops.isEmpty() && ops.peek() != '('){
   float val2 = values.pop();
   float val1 = values.pop();
   char op = ops.pop();

   values.push(applyOp(val1, val2, op));
```

```
      }

      // pop opening brace.
      if(!ops.isEmpty())
        ops.pop();
    }

    // Current token is an operator.
    else {
      // While top of 'ops' has same or greater
      // precedence to the current token, which
      // is an operator. Apply operator on top
      // of 'ops' to top two elements in the values stack.
      while(!ops.isEmpty() && precedence(ops.peek()) >= precedence(tokens[i])){
        float val2 = values.pop();
        float val1 = values.pop();
        char op = ops.pop();

        values.push(applyOp(val1, val2, op));
      }

      // Push the current token to 'ops'.
      ops.push(tokens[i]);
    }
  }

  while(!ops.isEmpty()){
    float val2 = values.pop();
    float val1 = values.pop();
    char op = ops.pop();

    values.push(applyOp(val1, val2, op));
  }

  // Top of 'values' contains result, return it.
  return values.peek();
}

void setup() {
  Serial.begin(9600);

  pinMode(6, OUTPUT); // PWM pin for motor 1
  pinMode(9, OUTPUT); // PWM pin for motor 2

  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  pinMode(A2, INPUT);
  pinMode(A3, INPUT);
  pinMode(A4, INPUT);
```

```
  String s = "(1+1)*2^0.5";
  Serial.println(s);
  float ans;
  ans = evaluate(s);
  Serial.println(evaluate(s));
}

void loop() {
  // Nothing to do here
}
```

Code for wirelesss communication between device and microcontroller :

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

const char *ssid = "BarakKriti";
const char *password = "jaibarak";

ESP8266WebServer server(80);
String enteredExpression;       // Variable to store the entered expression
String simplifiedExpression;   // Variable to store the simplified expression
String evaluatedResult;         // Variable to store the evaluated result
bool displayMode = true;        // true: display expression, false: display simplified expression or result
unsigned long lastUpdateTime = 0;
const unsigned long updateInterval = 3500; // Update every 3.5 seconds

// Define your LCD parameters
const int LCD_I2C_ADDR = 0x27;
const int LCD_COLUMNS = 16;
const int LCD_ROWS = 2;
LiquidCrystal_I2C lcd(LCD_I2C_ADDR, LCD_COLUMNS, LCD_ROWS);

// Function to update the LCD
void updateLCD() {
  lcd.clear();
  lcd.setCursor(0, 0);

  if (displayMode) {
    lcd.print("Expression: ");
    lcd.setCursor(0, 1);
    lcd.print(enteredExpression);
  } else {
    lcd.print("Simplified: ");
    lcd.setCursor(0, 1);
```

```
    lcd.print(simplifiedExpression);

    // If in result mode, display the evaluated result
    if (!evaluatedResult.isEmpty()) {
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Result: ");
      lcd.setCursor(0, 1);
      lcd.print(evaluatedResult);
    }
  }
}

// Function to handle the web page
void handleRoot() {
  String html = "<html><body style='text-align:center;'>";
  html += "<h1>Barak : Kriti BeyondBits</h1>";
  html += "<div style='margin:auto; width:50%; text-align:left;'>";
  html += "<form action='/simplify' method='get'>";
  html += "Enter mathematical expression: <input type='text' name='expression'><br>";
  html += "<input type='submit' value='Send' style='margin-top:10px;'>";
  html += "</form>";
  html += "</div></body></html>";

  server.send(200, "text/html", html);
}

// Function to handle the simplification
void handleSimplify() {
  String expression = server.arg("expression");
  expression.replace(" ", ""); // Remove spaces from the expression

  // Save the entered expression to the variable
  enteredExpression = expression;


  if (!evaluatedResult.isEmpty()) {
    server.send(200, "text/plain", "Result: " + evaluatedResult);
    // Toggle display mode between expression, simplified expression, and result
    displayMode = !displayMode;
    // Update the LCD
    updateLCD();
    // Reset the timer for the next update
    lastUpdateTime = millis();
  } else {
    server.send(400, "text/plain", "Error: Unable to simplify expression");
  }
}
```

```
void setup() {
  Serial.begin(115200);

  // Set up the Access Point
  WiFi.softAP(ssid, password);
  IPAddress myIP = WiFi.softAPIP();
  Serial.println("Access Point IP address: " + myIP.toString());

  // Initialize the LCD
  lcd.init();
  lcd.backlight();

  // Set up routes
  server.on("/", HTTP_GET, handleRoot);
  server.on("/simplify", HTTP_GET, handleSimplify);

  // Start server
  server.begin();
  Serial.println("HTTP server started");

  // Initial LCD update
  updateLCD();
}

void loop() {
  server.handleClient();

  // Check if it's time to update the LCD
  if (millis() - lastUpdateTime >= updateInterval) {
    // Toggle display mode between expression, simplified expression, and result
    displayMode = !displayMode;
    // Update the LCD
    updateLCD();
    // Reset the timer for the next update
    lastUpdateTime = millis();
  }
}
```