

Einstellbares Stereo VU Meter

Hallo und willkommen zum letzten Teil der „VU-Meter“ Reihe. In diesem Teil gehen wir auf einige eurer Verbesserungswünsche ein und versehen unser VU Meter mit neuen Einstellungsmöglichkeiten. Mithilfe von insgesamt 4 neuen Drehpotentiometern können jetzt sowohl die Helligkeit der Anzeige in einem weiten Spektrum eingestellt werden, als auch die Empfindlichkeit für jeden einzelnen Kanal aber auch im gesamten regeln!

Wir bitten jedoch auch um Verständnis, das nicht jeder eurer tollen Ideen und Verbesserungsvorschläge Beachtung geschenkt werden kann, da wir noch andere tolle Ideen für Projekte haben, die wir euch vorstellen wollen.

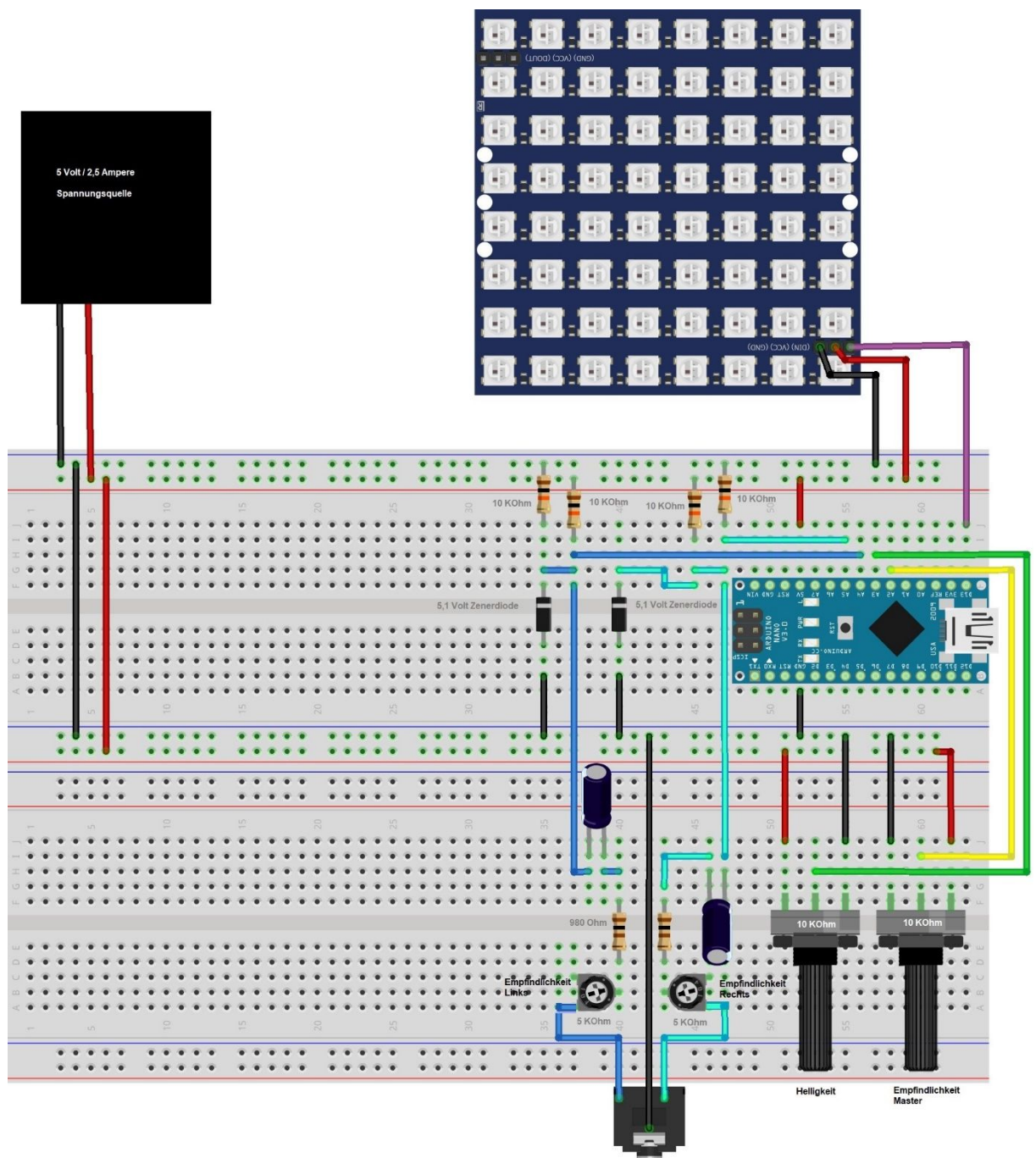
Befassen wir uns zunächst mit den heutigen Hardware Erweiterungen für unser VU Meter. Zunächst werden in den beiden Audiokanälen R und L ein 2 KOhm Potentiometer in Reihe geschaltet. Diese bilden zusammen einen individuell einstellbaren, wirksamen groben separaten Empfindlichkeitsregler für beide Kanäle. Darüber hinaus haben wir noch einen feinen 10 KOhm Empfindlichkeitsregler, der beide Kanäle erfasst, als separate softwareunterstützte Einstellungsmöglichkeit. Es empfiehlt sich, die beiden Eingangsregler als Printpotentiometer auszuführen, die einmal eingestellt, in dem Gehäuse des VU Meters eingestellt verbleiben. Der feine Empfindlichkeitsregler sollte dagegen von außen mit einem Drehknopf einstellbar, angebracht werden.

Als letztes fehlt uns noch eine Einstellungsmöglichkeit für die Helligkeit der Anzeige von 1-100%. Bisher haben wir die Helligkeit statisch im Code definiert, auch um die Strombelastung in Grenzen zu halten. Da dies nun mit unserem neuen, leistungsfähigeren Netzteil (5 Volt, 2,5 A min) nicht mehr nötig ist, geben wir auch diesen Parameter nach Außen über ein 10 KOhm Potentiometer zum Einstellen frei. Ich empfehle, auch dieses Potentiometer von außen mit einem Drehknopf einstellbar anzubringen.

Für letzten Teil der Reihe ergibt sich somit folgende, abschließende Teilliste:

- 1x [Arduino Nano \(mit FTDI\)](#)
- 1x [U64 LED Panel](#)
- 2x 10 KOhm Widerstände 1%
- 2x 680 Ohm Widerstand 5%
- 1x 5 Volt, min 2.5 Ampere Stromversorgung
- 2x 10 uF 64 Volt Elektrolytkondensator
- 1x Stereo Klinken-Steckverbinder 3.5 mm Buchse
- 2x 5,1 Volt Zener-Diode
- 2x 10 KOhm Drehpotentiometer (Helligkeit, Feinabstimmung Empfindlichkeit)
- 2x 2 KOhm Print-Drehpotentiometer (Grobabstimmung Empfindlichkeit R/L)

Wir verdrahten nun die Bauteile nachfolgendem Schaltplan:



fritzing

Nachdem wir die Schaltung komplett aufgebaut bzw. aktualisiert haben, können wir nun den angepassten Code auf unseren Arduino hochladen:

```
#include <Adafruit_NeoPixel.h>

// Which pin on the Arduino is connected to the NeoPixels?
// On a Trinket or Gemma we suggest changing this to 1:
#define LED_PIN 13
// How many NeoPixels are attached to the Arduino?
#define LED_COUNT 64

// Declare our NeoPixel strip object:
Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);
// Argument 1 = Number of pixels in NeoPixel strip
// Argument 2 = Arduino pin number (most are valid)
// Argument 3 = Pixel type flags, add together as needed:
// NEO_KHZ800 800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
// NEO_KHZ400 400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
// NEO_GRB Pixels are wired for GRB bitstream (most NeoPixel products)
// NEO_RGB Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
// NEO_RGBW Pixels are wired for RGBW bitstream (NeoPixel RGBW
products)

#define analogPinLeft A5 // Left Audio Channel, connected to analog pin A5
#define analogPinRight A4 // Left Audio Channel, connected to analog pin A4
#define analogPinBrightness A3 // Brightness Potentiometer 10 K, connected to
analog pin A3
#define analogPinSensitivity A2 // Sensivity Potentiometer 10 K, connected to
analog pin A2

#define Left_Channel_Deviation 5
#define Right_Channel_Deviation 5
#define Led_Brightness_Deviation 5
#define VU_Sensitivity_Deviation 7

int val_left_old = 0; // variable to store the value read from Channel Left
int Base_Left = 0; // 0 Basis
int val_right_old = 0; // variable to store the value read from Channel Left
int Base_Right = 0; // 0 Basis

int leftDropTime, rightDropTime;
int dropDelay = 4; // hold time before dropping the leds
int Led_Brightness = 512; // 50% Led Led_Brightness default
int Led_Brightness_old = 512;
int GlobalSensitivity = 512; // Init with Minimal Sensivity

byte LedRun_Brightness = 255; // 100% maximal Led_Brightness
int VU_Sensitivity = 0;
int VU_Sensitivity_old = 0;
float dropFactor = .98;
```

```

void setup()
{
  Serial.begin(9600);
  strip.begin();          // INITIALIZE NeoPixel strip object (REQUIRED)
  Led_Brightness = analogRead(analogPinBrightness);
  strip.setBrightness(map(Led_Brightness, 0, 1023, 1, 255)); // Set BRIGHTNESS
  according to Potentiometer
  strip.show();           // Turn OFF all pixels ASAP
  VU_Sensitivity = analogRead(analogPinSensitivity);
  VU_Sensitivity += analogRead(analogPinSensitivity);
  VU_Sensitivity += analogRead(analogPinSensitivity);
  VU_Sensitivity += analogRead(analogPinSensitivity);
  VU_Sensitivity = VU_Sensitivity / 4;

  Base_Left = analogRead(analogPinLeft);
  Base_Left += analogRead(analogPinLeft);
  Base_Left += analogRead(analogPinLeft);
  Base_Left += analogRead(analogPinLeft);
  Base_Left = Base_Left / 4;
  Base_Right = analogRead(analogPinRight);
  Base_Right += analogRead(analogPinRight);
  Base_Right += analogRead(analogPinRight);
  Base_Right += analogRead(analogPinRight);
  Base_Right = Base_Right / 4;
  Serial.begin(9600);
  colorWipe(strip.Color(255,0,0), 5); // Red
  Led_Brightness = analogRead(analogPinBrightness);
  strip.setBrightness(map(Led_Brightness, 0, 1023, 1, 255)); // Set BRIGHTNESS
  according to Potentiometer
  strip.show();
  colorWipe(strip.Color(255,255,0), 5); // yellow
  Led_Brightness = analogRead(analogPinBrightness);
  strip.setBrightness(map(Led_Brightness, 0, 1023, 1, 255)); // Set BRIGHTNESS
  according to Potentiometer
  strip.show();
  colorWipe(strip.Color(0,255,0), 5); // Green
  Led_Brightness = analogRead(analogPinBrightness);
  strip.setBrightness(map(Led_Brightness, 0, 1023, 1, 255)); // Set BRIGHTNESS
  according to Potentiometer
  strip.show();
  rainbowFade2White(1, 1, 1);
  Led_Brightness = analogRead(analogPinBrightness);
  strip.setBrightness(map(Led_Brightness, 0, 1023, 1, 255)); // Set BRIGHTNESS
  according to Potentiometer
  strip.show();
  Serial.println("Init complete. Virtual Zerolines:");
  Serial.print("Left : ");
  Serial.println(Base_Left);
  Serial.print("Right: ");
  Serial.println(Base_Right);
}

```

```

}

void loop()
{
  Left_VU_Meter(LED_COUNT / 2, GlobalSensitivity);
  Right_VU_Meter(LED_COUNT / 2, GlobalSensitivity);
  Read_External_Parameters();
  strip.show(); // Update strip to match
}

void Read_External_Parameters()
{
  Led_Brightness = 0;
  Led_Brightness = analogRead(analogPinBrightness);
  Led_Brightness += analogRead(analogPinBrightness);
  Led_Brightness = Led_Brightness / 2;
  if ((abs(Led_Brightness - Led_Brightness_old) > Led_Brightness_Deviation))
  {
    Led_Brightness_old = Led_Brightness;
    // Serial.print("New LED Brightness: ");
    // Serial.println(map(Led_Brightness, 0, 1023, 1, 255));
    strip.setBrightness(map(Led_Brightness, 0, 1023, 1, 255)); // Set BRIGHTNESS
    according to Potentiometer
    strip.show();
  }
  VU_Sensitivity = 0;
  VU_Sensitivity = analogRead(analogPinSensitivity);
  VU_Sensitivity += analogRead(analogPinSensitivity);
  VU_Sensitivity = VU_Sensitivity / 2;
  if ((abs(VU_Sensitivity - VU_Sensitivity_old) > VU_Sensitivity_Deviation))
  {
    VU_Sensitivity_old = VU_Sensitivity;
    GlobalSensitivity = map(VU_Sensitivity, 0, 1023, 30, 512);
    // Serial.print("New VU Sensivity: ");
    // Serial.println(GlobalSensitivity);
  }
}

void Left_VU_Meter(byte Level_Max_Pixels, int sensitivity)
{
  int val_left = 0;
  bool Overload = false;
  uint32_t rgbcolor;
  uint32_t hue;
  int Signal_Strength = 0;
  byte VU_Led_Level = 0;
  val_left = analogRead(analogPinLeft); // read the input pin
  val_left += analogRead(analogPinLeft); // read the input pin
  val_left += analogRead(analogPinLeft); // read the input pin
  val_left += analogRead(analogPinLeft); // read the input pin

```

```

val_left = val_left / 4;
if (!(abs(val_left - val_left_old) > Left_Channel_Deviation)) {
    val_left = val_left_old;
}
if (val_left < val_left_old)
{
    leftDropTime++;
    if (leftDropTime > dropDelay)
    {
        val_left = val_left_old * dropFactor;
        leftDropTime = 0;
    }
    else
    {
        val_left = val_left_old;
    }
}
val_left_old = val_left;
Signal_Strength = val_left - Base_Left;
if (Signal_Strength < 0) { Signal_Strength = - Signal_Strength; }
VU_Led_Level = map(Signal_Strength, 0, sensitivity, 0, Level_Max_Pixels);
if (VU_Led_Level > Level_Max_Pixels)
{
    Overload = true;
    VU_Led_Level = Level_Max_Pixels;
} else { Overload = false; }
for(int i=0; i<Level_Max_Pixels; i++) { strip.setPixelColor(i, 0,0,0); } // Clear pixel's
color (in RAM)
for(int i=0; i<VU_Led_Level; i++) { // For each pixel in strip...
    hue = map(i, Level_Max_Pixels -1,0, 0, 21800);
    if (Overload) { rgbcolor = strip.Color(255,0,0); } else { rgbcolor =
strip.ColorHSV(hue, 255, LedRun_Brightness); } // Hue to RGB Conversation
    strip.setPixelColor(i, rgbcolor); // Set pixel's color (in RAM)
}
}

void colorWipe(uint32_t color, int wait) {
    for(int i=0; i<strip.numPixels(); i++) { // For each pixel in strip...
        strip.setPixelColor(i, color); // Set pixel's color (in RAM)
        strip.show(); // Update strip to match
        delay(wait); // Pause for a moment
    }
}

void Right_VU_Meter(byte Level_Max_Pixels,int sensitivity)
{
    int val_right = 0;
    bool Overload = false;
    uint32_t rgbcolor;
    uint32_t hue;
    int Signal_Strength = 0;

```

```

byte VU_Led_Level = 0;
val_right = analogRead(analogPinRight); // read the input pin
val_right += analogRead(analogPinRight); // read the input pin
val_right += analogRead(analogPinRight); // read the input pin
val_right += analogRead(analogPinRight); // read the input pin
val_right = val_right / 4;
if (!(abs(val_right - val_right_old) > Right_Channel_Deviation)) {
    val_right = val_right_old;
}
if (val_right < val_right_old)
{
    rightDropTime++;
    if (rightDropTime > dropDelay)
    {
        val_right = val_right_old * dropFactor;
        rightDropTime = 0;
    }
    else
    {
        val_right = val_right_old;
    }
}
val_right_old = val_right;
Signal_Strength = val_right - Base_Right;
if (Signal_Strength < 0) { Signal_Strength = - Signal_Strength; }
VU_Led_Level = map(Signal_Strength, 0, sensitivity, 0, Level_Max_Pixels);
if (VU_Led_Level > Level_Max_Pixels)
{
    Overload = true;
    VU_Led_Level = Level_Max_Pixels;
} else { Overload = false; }
int ColorVector = 0;
for(int i=LED_COUNT-Level_Max_Pixels; i<LED_COUNT; i++) {
    strip.setPixelColor(i, 0,0,0); } // Clear pixel's color (in RAM)
int StartVector = LED_COUNT - VU_Led_Level;
for(int i=LED_COUNT-Level_Max_Pixels; i<LED_COUNT; i++) { // For each pixel
in strip...
    hue = map(ColorVector, Level_Max_Pixels -1,0, 21800, 0);
    ColorVector++;
    if ( i >=StartVector)
    {
        if (Overload) { rgbcolor = strip.Color(255,0,0); } else { rgbcolor =
strip.ColorHSV(hue, 255, LedRun_Brightness); } // Hue to RGB Conversation
        strip.setPixelColor(i, rgbcolor); // Set pixel's color (in RAM)
    }
}
}

void rainbowFade2White(int wait, int rainbowLoops, int whiteLoops) {
    int fadeVal=0, fadeMax=100;
    // Hue of first pixel runs 'rainbowLoops' complete loops through the color

```



```

// wheel. Color wheel has a range of 65536 but it's OK if we roll over, so
// just count from 0 to rainbowLoops*65536, using steps of 256 so we
// advance around the wheel at a decent clip.
for(uint32_t firstPixelHue = 0; firstPixelHue < rainbowLoops*65536;
    firstPixelHue += 256) {
    for(int i=0; i<strip.numPixels(); i++) { // For each pixel in strip...
        // Offset pixel hue by an amount to make one full revolution of the
        // color wheel (range of 65536) along the length of the strip
        // (strip.numPixels() steps):
        uint32_t pixelHue = firstPixelHue + (i * 65536L / strip.numPixels());
        // strip.ColorHSV() can take 1 or 3 arguments: a hue (0 to 65535) or
        // optionally add saturation and value (brightness) (each 0 to 255).
        // Here we're using just the three-argument variant, though the
        // second value (saturation) is a constant 255.
        strip.setPixelColor(i, strip.gamma32(strip.ColorHSV(pixelHue, 255,
            255 * fadeVal / fadeMax)));
    }
    strip.show();
    delay(wait);
    if(firstPixelHue < 65536) { // First loop,
        if(fadeVal < fadeMax) fadeVal++; // fade in
    } else if(firstPixelHue >= ((rainbowLoops-1) * 65536)) { // Last loop,
        if(fadeVal > 0) fadeVal--; // fade out
    } else {
        fadeVal = fadeMax; // Interim loop, make sure fade is at max
    }
}
for(int k=0; k<whiteLoops; k++) {
    for(int j=0; j<256; j++) { // Ramp up 0 to 255
        // Fill entire strip with white at gamma-corrected brightness level 'j':
        strip.fill(strip.Color(0, 0, 0, strip.gamma8(j)));
        strip.show();
    }
    for(int j=255; j>=0; j--) { // Ramp down 255 to 0
        strip.fill(strip.Color(0, 0, 0, strip.gamma8(j)));
        strip.show();
    }
}
}
}

```

Die erste Kalibrierung der Empfindlichkeit sollte an den getrennten R/L Grobreglern vorgenommen werden. Es sollte dabei darauf geachtet werden, dass der Fein Regler für die Empfindlichkeit auf „Minimum“ eingestellt wird. Weiterhin sollte darauf geachtet werden, dass während des Einschaltens des VU Meters und dem Abspielen des Intros die virtuelle Nulllinie eingelesen wird. Damit diese Kalibrierung korrekt ablaufen kann, ist es wichtig, während der Einschaltphase noch kein Audio Signal anzulegen. Erst wenn die Startanimationen beendet sind, kann ein Analogsignal angelegt werden.

Ich wünsche viel Spaß beim Musik hören und nachbauen.

Bis zum nächsten Mal.