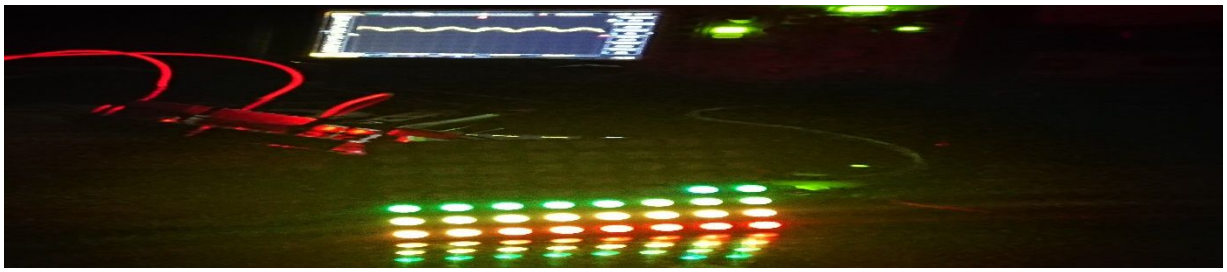


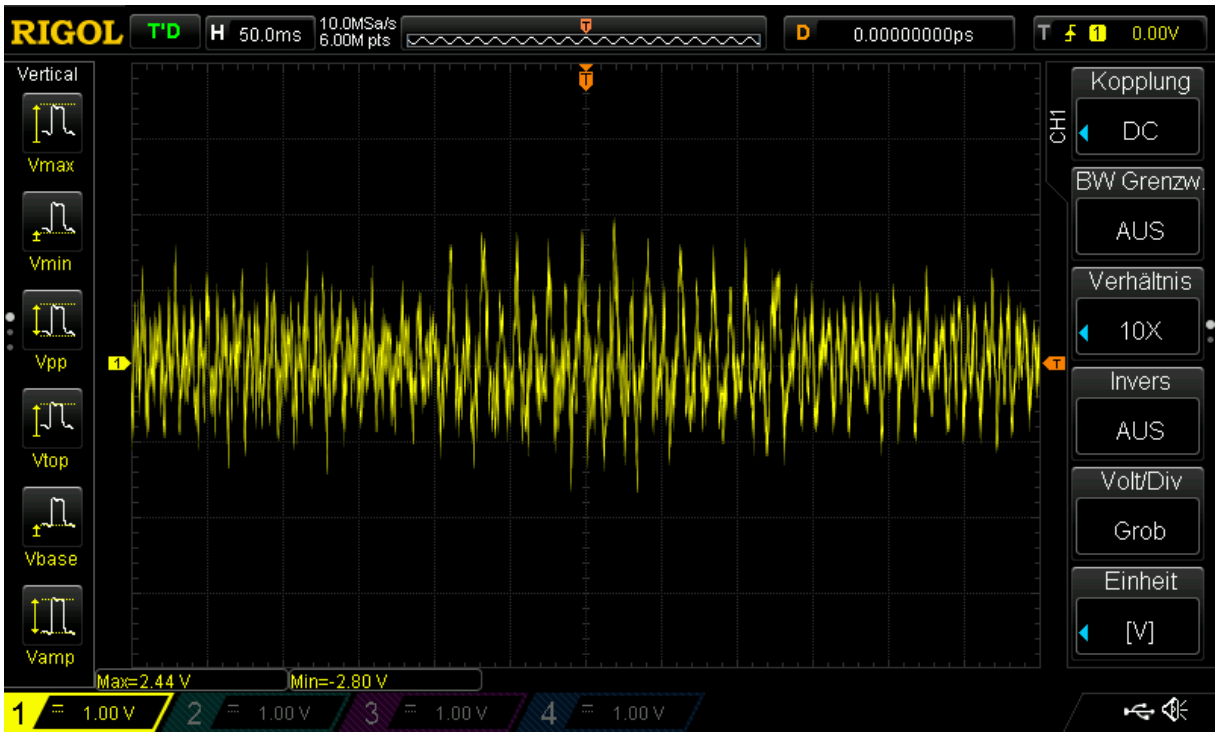
Selbstgebautes VU-Meter

Hallo und willkommen zu einem neuen und spannenden Blog. Heute begeben wir uns in die Niederfrequenz-Welt und bauen wir uns ein eigenes VU-Meter. Für alle, die die aus den 80ern bekannte Messgerät nicht mehr kennen: Ein VU-Meter (vu steht dabei für englischen Begriffe Volume Units, also in etwa „Lautstärkeneinheiten“) ist ein Aussteuerungsmesser, also ein Messinstrument zur Beurteilung der Signalstärke in der Tontechnik. Dabei wird die Signalstäke eines eingespeisten Audiosignales in seinem elektrischen Äquivalent gemessen.

Wir brauchen folgende Hardwareteile aus dem Shop um das VU Meter aufbauen zu können:

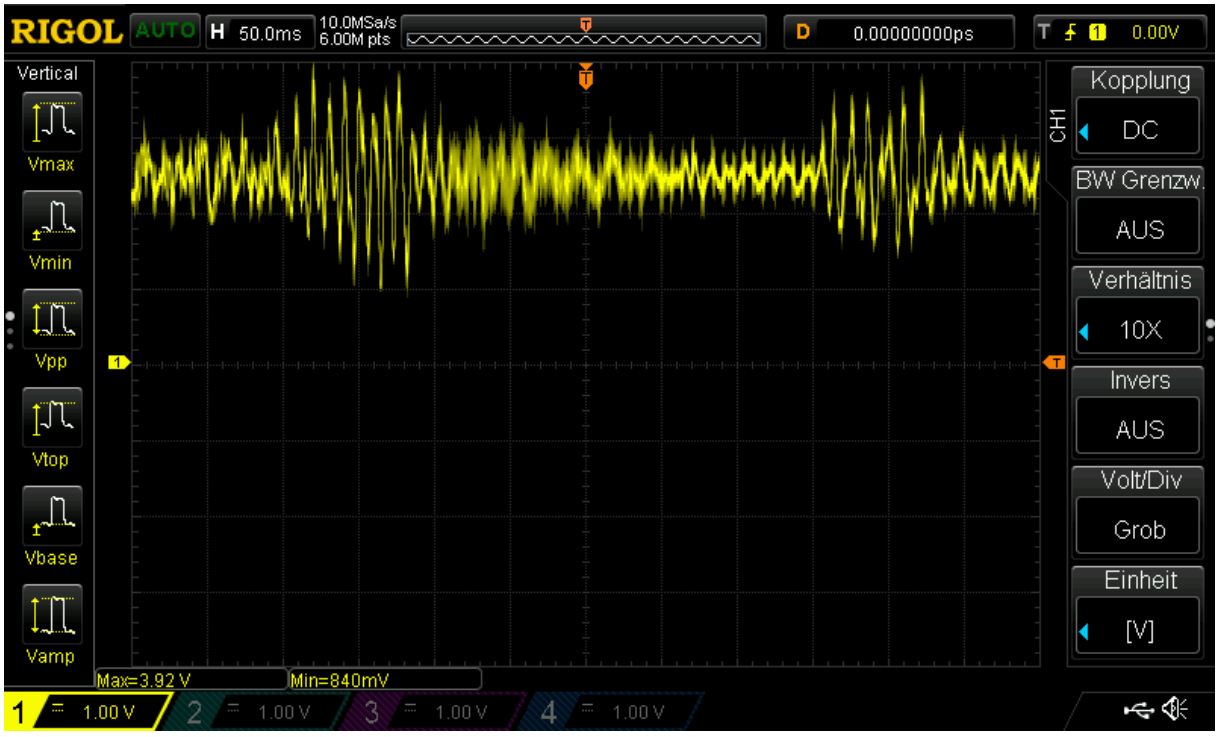
- 1x Arduino Nano (mit FTDI)
- 1x U64 LED Panel
- 2x 10 KOhm Wiederstände
- 1x YwRobot Breadboard Spannungsversorgung
- 1x 10 uF 64 Volt Elektrolytkondensator
- 1x Stereo Klinken-Steckverbinder 3.5 mm Buchse

Damit wir ein Audio Signal mit unserem Arduino vermessen können, müssen wir zuerst einige Vorüberlegungen anstellen. Die wichtigste zuerst: Das Audiosignal ist eine arbiträre Wechselspannung im KHz Bereich, die um den Nullpunkt schwingt. Zur Messung dieses Signals eignet sich somit kein Digitaleingang. Auch ein Analogeingang unsers Microcontrollers für die Messung des Musik Niederfrequenzsignals ist ohne externe Beschaltung nur bedingt geeignet, da dieser nur positive Spannungswerte im Bereich von 0 bis 5 Volt bei einer Auflösung von 10 Bit messen kann. Um dies zu visualisieren zeige ich nachfolgend mit dem Oszilloskop aufgezeichnete Audio NF Schwingungen, so wie diese auch an einem Lautsprecher anliegen.



Abbindung: NF-Signal am Lausprechereingang, ungefiltert

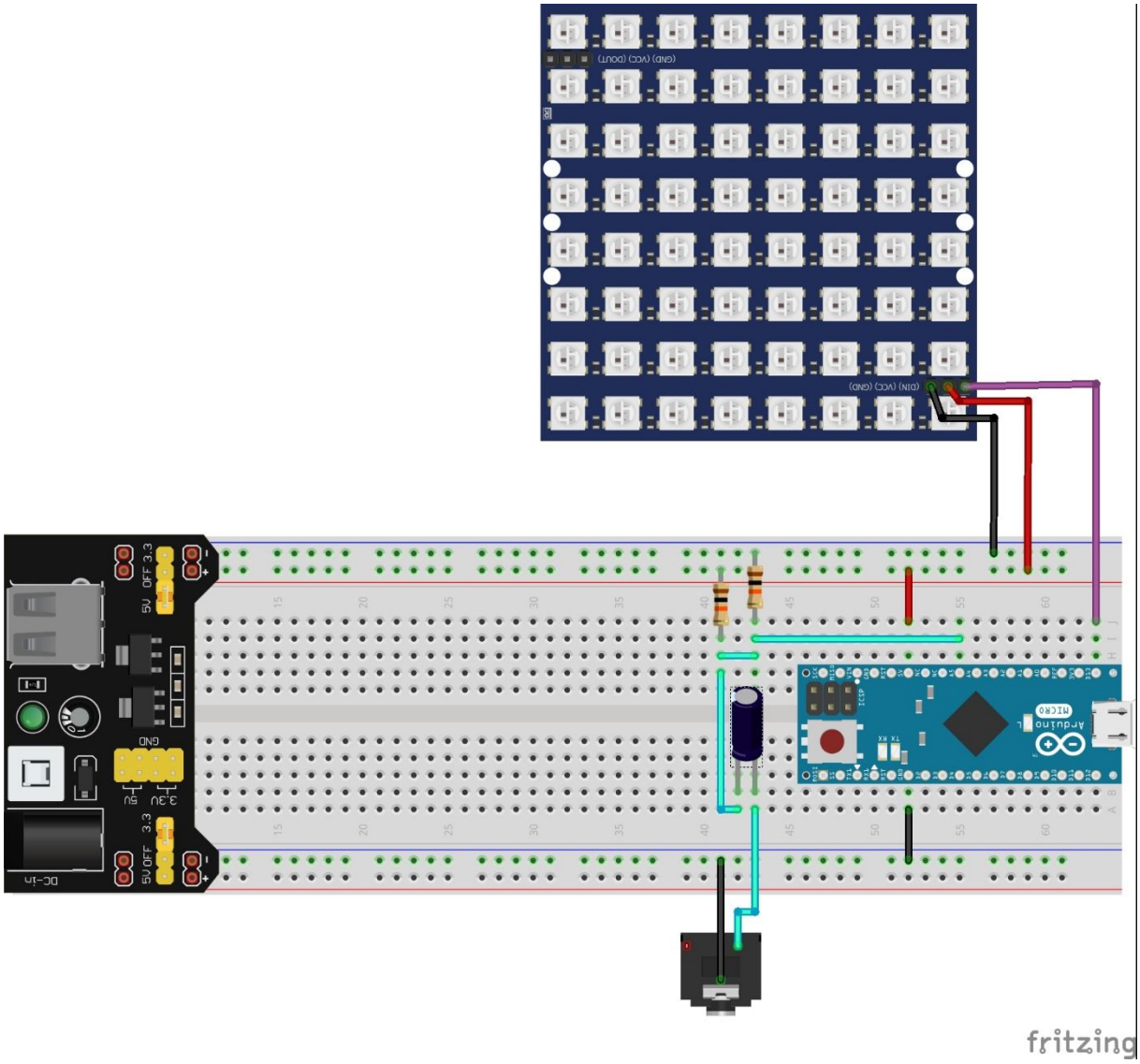
Wir bereiten daher im ersten Schritt unser Analoges NF Musik Signal mithilfe einer kleinen externen Wiederstands-Kondensator Schaltung so auf, dass es mit einem Offset von 2, 5 Volt in den positiven Spannungsbereich gehoben wird. Der Nulldurchgang in den negativen Bereich wird somit vermieden:



Abbindung: Gleiches NF Signal mit einem Gleichspannungsoffset.

Dies bewerkstelligen wir mit den in der Schaltungsplan zu sehenden zwei 10 KOhm Wiederständen und einem 10uF 63 Volt Elektrolytkondensator, dessen Kathode zu unserem Microcontroller zeigt.

Bauen wir nun das VU Meter wie auf folgender Abbildung zu sehen komplett auf:



Wichtig ist, dass das 66 Led Panel seine Stromversorgung durch das Power-Modul YW Robot bezieht und NICHT durch den Arduino Nano, da im Betrieb bis zu 2,5 A durch den U64-LED Panel in der Spitze verbraucht werden. Diese Stromstärke würde, falls diese durch den Arduino bereitgestellt werden müsste, unweigerlich die Spannung zusammenbrechen lassen, bzw. den Spannungsregler auf dem Nano beschädigen. Auch die Dimensionierung des Netzteils sollte entsprechend gewählt werden.

Nach dem Aufbau der Schaltung laden folgenden Code auf den Arduino hoch:

```
#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

// NeoPixel brightness, 0 (min) to 255 (max)
#define BRIGHTNESS 100

// 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

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

void setup()
{
  strip.begin();           // INITIALIZE NeoPixel strip object (REQUIRED)
  strip.show();            // Turn OFF all pixels ASAP
  strip.setBrightness(BRIGHTNESS); // Set BRIGHTNESS to about 1/5 (max = 255)
  Base_Left = analogRead(analogPinLeft);
  Base_Left += analogRead(analogPinLeft);
  Base_Left += analogRead(analogPinLeft);
  Base_Left += analogRead(analogPinLeft);
  Base_Left = Base_Left / 4;
  colorWipe(strip.Color(255,0,0), 5);   // Red
  colorWipe(strip.Color(255,255,0), 5); // yellow
  colorWipe(strip.Color(0,255,0), 5);   // Green
  rainbowFade2White(1, 1, 1);
}

void loop()
{
Left_VU_Meter(64,100);
delay(40);
}

void Left_VU_Meter(byte Level_Max_Pixels,int sensitivity)
```

```
{
  int val_left = 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;
  int Signal_Strength = val_left - Base_Left;
  if (Signal_Strength < 0) { Signal_Strength = - Signal_Strength; }
  byte VU_Led_Level =  map(Signal_Strength, 0, sensitivity , 0, Level_Max_Pixels);
  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...
    uint32_t hue = map(i, Level_Max_Pixels -1,0, 0, 21800);
    uint32_t rgbcolor = strip.ColorHSV(hue, 255, BRIGHTNESS); // Hue to RGB
Conversation
    strip.setPixelColor(i, rgbcolor);         //  Set pixel's color (in RAM)
  }
   strip.show();   //  Update strip to match
}

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 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();
    }
  }
}
```

Jetzt können wir ein NF Analogsignal eines Lautsprechers an die Audiobuchse anschließen. Line-In Level reichen hierbei NICHT aus, um unser VU Meter zu betreiben. Der Audio Eingang werden daher direkt mit dem Leistungsausgang der Audioquelle (max 30 Watt Peak) verbunden.

In den nächsten beiden Teilen dieser Reihe bauen wir einen zweiten Kanal, um auch Stereosignale mit dem VU Meter anzeigen zu können, und fügen einige Einstellungsmöglichkeiten, wie zb. Empfindlichkeit und Helligkeit hinzu.

Schreibt eure Fragen oder Wünsche wie immer in die Kommentare.

Ich wünsche viel Spaß bei Nachbau des VU Meters und bis zum nächsten Mal.