ELEKTRONIK-LABORBERICHT

Von VEREBELY Paul-loup & *HugoDUPRAT* & *NoéDIBLASI*

Juni 2025

Contents

List of Figures

1 Einleitung

Dieses Dokument beschreibt den gesamten Prozess des AP3-Elektronik-Abschlussprojekts. Um eine klare und strukturierte Lektüre zu gewährleisten, ist der Bericht in mehrere verschiedene Abschnitte unterteilt.

Zunächst stellt der Kontext-Abschnitt die Projektziele, das ursprüngliche Lastenheft sowie die wichtigsten identifizierten Einschränkungen und Herausforderungen vor.

Der Abschnitt zur Dimensionierung legt anschließend alle Berechnungen, Hypothesen und technischen Entscheidungen dar, die das Team getroffen hat, um das elektronische System gemäß den Spezifikationen zu entwerfen.

Der Software-Abschnitt beschreibt detailliert die mit dem Projekt verbundene Softwareentwicklung, wobei die Programmierlogik, die Code-Architektur und die wichtigsten implementierten Funktionen spezifiziert werden.

Abschließend werden im Abschnitt über Tests und aufgetretene Schwierigkeiten die verschiedenen Schritte zur Systemvalidierung, die während des Projekts aufgetretenen technischen Probleme und die vom Team erarbeiteten Lösungen vorgestellt.

Was die Teamaufteilung betrifft, so besteht es aus 3 Mitgliedern, die alle an den verschiedenen Aspekten des Projekts teilgenommen haben.

Dieser Bericht soll auch als Beispiel dienen, falls das Projekt vom Leser reproduziert werden soll.

2 Ursprüngliches Lastenheft

Dieses Projekt ermöglicht die Konsolidierung aller im Laufe des Jahres erworbenen Elektronikkenntnisse. Um all dies in kürzester Zeit in ein machbares Projekt umzusetzen, hat Antoine PIROG ein vollständiges und präzises Lastenheft erstellt. Hier ist der Inhalt:

Das zu entwerfende System ist ein RGBW-LED-VU-Meter, das dazu bestimmt ist, den Schallpegel eines Audiosignals in Echtzeit zu visualisieren. Die Projektarchitektur gliedert sich in mehrere unten beschriebene Funktionsblöcke.

2.1 Stromversorgung

- Einfache Versorgungsspannung: **+5V DC**
- Diese Spannung versorgt die gesamte Elektronikplatine.
- Sie muss strikt eingehalten werden, um eine Beschädigung empfindlicher Komponenten (insbesondere der LEDs und des Mikrocontrollers) zu vermeiden.

2.2 Erfassung und Konditionierung des Audiosignals

- Das Eingangs-Audiosignal wird in zwei Pfade aufgeteilt:
  - einer zu den analogen Filtern,
  - einer zu einem Audioausgang (für aktive Lautsprecher).
- Die Signale werden dann durch 4 analoge Bandpassfilter gefiltert, die folgende Frequenzbänder definieren:
  - **Bässe**: < 250 Hz
  - **Tiefmitten**: 250 Hz – 1 kHz
  - **Hochmitten**: 1 kHz – 4 kHz
  - **Höhen**: > 4 kHz
- Jeder Filter muss unter Verwendung von Operationsverstärkern (Op-Amps) und passiven Komponenten korrekt dimensioniert werden.

2.3 Digitale Verarbeitung

- Der Mikrocontroller **PIC18F25K40** führt folgendes aus:
  - analoge Erfassung der Signale von den Filtern,
  - digitale Verarbeitung des Pegels in jedem Band,
  - Datenformatierung für die Anzeige.
- Der Code ist in **C-Sprache** für die allgemeine Logik und in **Assembler** für die LED-Steuerung (hochpräzises Bitbanging) geschrieben.

2.4 LED-Anzeige

- Anzeige auf einer Matrix von **64 SK6812 RGBW LEDs**, einzeln adressierbar.
- Jede LED wird über ein serielles Protokoll gemäß einer Sequenz von **2048 Bits** (64 LEDs × 4 Farben × 8 Bits) gesteuert.
- die Farbreihenfolge ist: **Grün**, **Rot**, **Blau**, **Weiß**.
- Die Intensität jeder Farbe ist auf 8 Bits codiert (empfohlene Werte: 16 bis 32, um übermäßige Helligkeit zu vermeiden).

2.5 Benutzeroberfläche und Tests

- **9 Test-LEDs** und **2 Drucktaster** stehen zur Verfügung um:
  - das Programm unabhängig von der LED-Matrix zu testen,
  - zusätzliche Funktionen zu implementieren (Pause, Anzeigemodi...).

2.6 Entwicklungsrahmenbedingungen

- **Projektdauer**: 7 Sitzungen
- **Erwartete Ergebnisse**:
  - Funktionaler Prototyp,
  - Vollständiger Quellcode (C + Assembler),
  - Schriftlicher Bericht einschließlich Schaltplänen, Berechnungen, Bode-Diagrammen, Oszillogrammen, Stückliste.
- **Durchzuführende Tests**:
  - Oszillogramme an Testpunkten,
  - Validierung der Filter mit Funktionsgenerator,
  - Kontrolle des Assembler-Timings mit Oszilloskop.

3 Elektronische Dimensionierung

In diesem Projekt ist die Dimensionierung ein wichtiger Teil, der nicht vernachlässigt werden darf. Mehrere Teile der Platine müssen korrekt dimensioniert werden, um ein hohes Leistungsniveau und eine gute Schaltungsintegrität zu gewährleisten. Um all dies zu beschreiben, wird die Dimensionierung in mehreren Blöcken erfolgen, um die Arbeit und das Verständnis zu erleichtern. Schließlich werden aus Sicherheitsgründen alle Dimensionierungstests auf Falstad und unter Verwendung der E6-Reihe durchgeführt.

3.1 Referenzspannung

Wenn man den elektrischen Schaltplan betrachtet, bemerkt man, dass ein guter Teil der Operationsverstärker mit Vref versorgt wird. Diese Spannung fungiert als Referenzspannung; sie ermöglicht es unserem Audiosignal, um 2,5V zu schwingen, da das PICKIT zwischen 0 und 5 Volt arbeitet. Hierfür finden wir einen Spannungsteiler, der mit 5V versorgt wird, sowie eine Spannungsfolger-Op-Amp-Konfiguration.

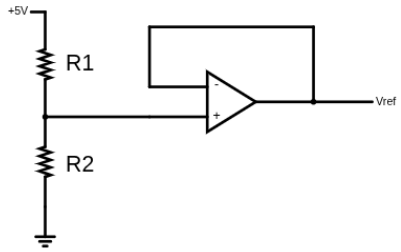

Figure 1: Schaltplan Spannungsteiler

Formel Spannungsteiler: $Vref = U * \frac{R2}{R2+R1}$ Formel Spannungsfolger-Op-Amp: $Vs = Ve$

Da unsere Eingangsspannung 5V beträgt und wir am Ausgang 2,5V wünschen, müssen wir die Spannung nur halbieren. Hierfür, und dank des Spannungsteilers, werden wir sehen, dass die Widerstände Rref1 und Rref2 gleich sein werden. Der Einfachheit halber wird für beide Widerstände der Wert von 1K Ohm beibehalten.

$$Vref = 5 * \frac{1 * 10}{2 * 10}$$

$$Vref = 2.5V$$

3.2 Vorverstärkung

Die Schaltung empfängt das Audiosignal über die erste Klinkenbuchse. Es ist zu beachten, dass die in diesem Kabel anliegende Spannung zwischen Kabeln und Computern variiert. Wichtig ist, dass sie sehr niedrig ist. Im Fall des Teams, wo Hugos PC verwendet wird, wird eine durchschnittliche Spannung von 150 mV beobachtet. Wir müssen dann einen Vorverstärkungsschritt durchlaufen, um das Eingangssignal zu erhöhen und korrekt damit arbeiten zu können, hier durchschnittlich 2,5V. Im Aufbau beobachten wir zwei Vorverstärker, einen rechten und einen linken, die jedoch identisch sind. Die Dimensionierung von nur einem ist daher notwendig.

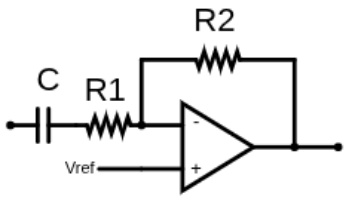

Figure 2: Schaltplan Vorverstärker

In diesem Aufbau beobachten wir eine nicht-invertierende Anordnung, der ein Kondensator vorgeschaltet ist. Hier wissen wir, dass unsere Ausgangsspannung maximal 5V und durchschnittlich 2,5V betragen muss. Hierfür werden wir die notwendige Verstärkung (Gain) berechnen und unsere Widerstände entsprechend dimensionieren.
Der Kondensator ermöglicht mehrere Dinge. Er fungiert als Barriere zwischen dem Eingangssignal mit unbekannter Durchschnittsspannung und der Durchschnittsspannung des Operationsverstärkers, die aufgrund seiner Vref-Versorgung bei 2,5V liegt. Dies vermeidet Konflikte zwischen diesen beiden Zonen, denn ohne ihn würde der Op-Amp, der seine Verstärkung um 2,5V anwendet, zwar verstärken, aber der Klinken-Teil würde nicht korrekt oder gar nicht verstärkt. Um ihn zu dimensionieren, können wir sagen, dass er mit dem folgenden Widerstand als Hochpassfilter wirkt. Dies ermöglicht auch die Eliminierung zu niedriger Frequenzen dank seiner Grenzfrequenz. Seine Dimensionierung erfolgt so, dass die Grenzfrequenz bei 20Hz liegt, der minimalen für den Menschen hörbaren Frequenz.
Formel Nicht-invertierender Op-Amp: $Vs = (1 + \frac{Rpre1}{Rpre0}) * Ve$

Formel Hochpassfilter: $\frac{1}{2\pi * Rpre0 * Cpre0}$

Was die Verstärkung betrifft, müssen wir das Verhältnis von 150mV und 2,5V bilden:

$$\frac{Vs}{Ve} = \frac{5}{0.15} = 16.6$$

Mit der E6-Reihe und unserer Formel finden wir:

$$\frac{Vs}{Ve} = (1 + \frac{75 * 10}{4.7 * 10}) = 16.95$$

Dank dessen finden wir für Rpre0 und Rpre1 Werte von 75K Ohm und 4.7K Ohm. Gehen wir zur Dimensionierung des Kondensators über: $20 = \frac{1}{2\pi * 4.7 * 10 * C}$

$C = \frac{1}{2\pi * 4.7 * 10 * 16.95} = 1.7 * e - 6$

Bei der Komponentenreihe kommt der Wert 2,2uF am nächsten, was uns eine Grenzfrequenz von 15,40Hz ergibt.

3.3 Links/Rechts-Mischung

Der Mischteil dient dazu zu entscheiden, ob wir die linke oder rechte Seite unseres Audiosignals wollen. In unserer Schaltung finden wir ein Potentiometer; es funktioniert wie 2 variable Widerstände, die an Vref angeschlossen sind, sein Maximalwert beträgt 10K Ohm. Wir finden auch eine etwas besondere Summierverstärker-Anordnung. Ihre Dimensionierung erfolgt schrittweise, um den Leser nicht zu verlieren.
Wie bei den Vorverstärkern können wir eine Seite der Schaltung untersuchen, da sich zwischen links und rechts nichts ändert. Um die Berechnungen zu vereinfachen, wird die Schaltung nicht mit Vref, sondern gegen Masse betrachtet. Hier ist unsere Schaltung:

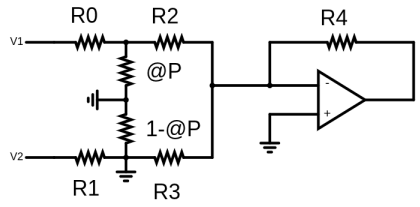

Figure 3: Schaltplan Mischung

Und hier ist, wie wir sie interpretieren werden:

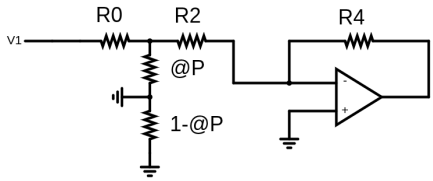

Figure 4: Interpretativer Schaltplan Mischung

Wir nehmen an, dass $V_2 = 0$ ist und dass das Potentiometer seine gesamte Bandbreite auf V_1 liefert. Wir werden zwei Berechnungen gemäß dem Satz von Millman durchführen, eine erste zwischen R_0, R_2 und αP, notiert @P, und eine zweite am invertierenden Anschluss der Schaltung.

Erster Millman an V^-:
Sei V_a die Spannung über R_2.

$$\frac{V_a}{R_2} + \frac{V_s}{R_4} = \frac{V_a R_4 + V_s R_2}{R_2 + R_4}$$

Zweiter Millman zwischen R_0, R_2 und αP zur Berechnung von V_a:

$$V_a = \frac{\frac{V_L}{R_0} + \frac{0}{\alpha P} + \frac{V^-}{R_2}}{\frac{1}{R_0} + \frac{1}{\alpha P} + \frac{1}{R_2}}$$
$$= \frac{V_L \alpha P R_2 + V^- R_0 \alpha P}{\alpha P R_2 + R_0 R_2 + R_0 \alpha P}$$

$$V^- = \frac{V_a R_4 + V_s R_2}{R_2 + R_4}$$
$$= \left(\frac{V_L \alpha P R_2 + V^- R_0 \alpha P}{\alpha P R_2 + R_0 R_2 + R_0 \alpha P} \right) \times \frac{R_4}{R_2 + R_4} + \frac{V_s R_2}{R_2 + R_4}$$
$$= \frac{V_L R_2}{R_0 + R_2 + \frac{R_0 R_2}{\alpha P}} \times \frac{R_4}{R_2 + R_4} + \frac{V_s R_2}{R_2 + R_4} = 0$$
$$V_L \times \frac{R2}{R_0 + R_2 + \frac{R_0 R_2}{\alpha P}} \times \frac{R_4}{R_2 + R_4} = -\frac{V_s R_2}{R_2 + R_4}$$
$$\leftrightarrow V_L \times \frac{R2}{R_0 + R_2 + \frac{R_0 R_2}{\alpha P}} \times \frac{R_4}{R_2 + R_4} \times \frac{R_2 + R_4}{R_2} = -V_s$$

$$V_L \times \frac{R2}{R_0 + R_2 + \frac{R_0 R_2}{\alpha P}} \times \frac{R_4}{R_2 + R_4} \times \frac{R_2 + R_4}{R_2} = -V_s$$
$$\leftrightarrow \frac{R_4}{R_0 + R_2 + \frac{R_0 R_2}{\alpha P}} = -\frac{V_s}{V_L}$$

Wir finden dann, dass das Verhältnis zwischen Ausgang und Eingang des Op-Amps ist:

$$-\frac{V_s}{V_L} = \frac{R_4}{R_0 + R_2 + \frac{R_0 R_2}{\alpha P}}$$

Für die Dimensionierung müssen wir Werte für R_0, R_2 und R_4 finden, die folgende Werte ergeben:

$$1 \text{ für } \alpha P = 0 \tag{1}$$
$$0.5 \text{ für } \alpha P = 0.5 \tag{2}$$
$$1 \text{ für } \alpha P = 0 \tag{3}$$

Nach Tests mit der Simulation von Herrn PIROG kam das Team zu dem Schluss, dass Werte von 47K, 68K und 470K für R_0, R_2 und R_4 vielversprechende Werte waren. Diese Werte gelten daher auch jeweils für R_1, R_3.

3.4 Hochpass- / Tiefpassfilter

Die Untersuchung der Filter beginnt mit den beiden Filtern: Hochpass und Tiefpass. Für ihre Dimensionierung ist das gesuchte Element die Grenzfrequenz bei -3dB. Um diese Frequenz zu berechnen, verwenden wir die Formel

$$F_{cutoff} = \frac{1}{2\pi RC}$$

3.4.1 Hochpassfilter

Beginnen wir mit dem Hochpassfilter. Seine Rolle ist es, nur hohe Frequenzen über 4KHz (seine Grenzfrequenz) mit einer Verstärkung von -3dB durchzulassen. Der Aufbau ist wie folgt konstruiert:

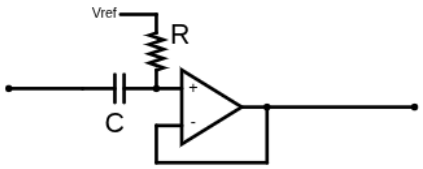

Figure 5: Hochpassfilter

Um diesen Filter zu dimensionieren, müssen wir also rechnen

$$\frac{1}{2\pi RC} = 4 \times 10^3 => RC = \frac{1}{2\pi \times 4 \times 10^3}$$

Immer noch mit der E6-Reihe wählt das Team einen Widerstandswert von 820 Ohm. Wir finden dann einen Kondensatorwert von 47n Farad, mit einer Verstärkung von -3,16dB bei 4KHz.

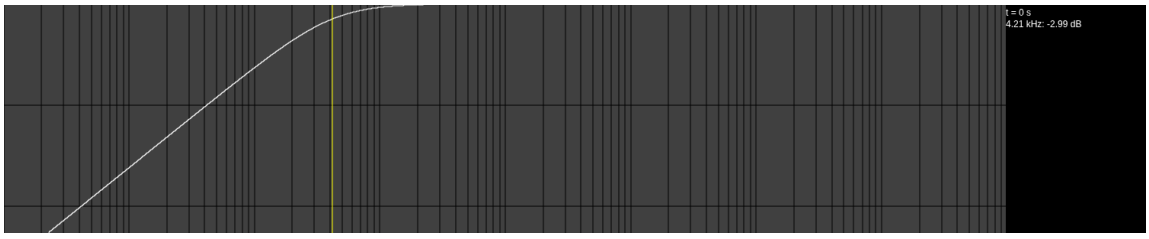

Figure 6: HP Grenzfrequenz

3.4.2 Tiefpassfilter

Die Rolle des Tiefpassfilters ist es, nur tiefe Frequenzen unter 250Hz (seine Grenzfrequenz) mit einer Verstärkung von -3dB durchzulassen. Der Aufbau erfolgt wie folgt:

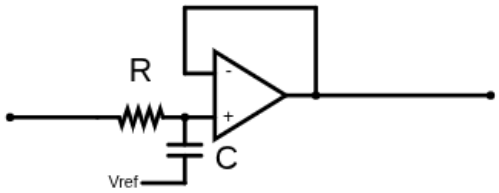

Figure 7: Tiefpassfilter

Durch Umstellen des Ausdrucks der Grenzfrequenz finden wir:

$$RC = \frac{1}{2\pi \times 250}$$

Das Team entschied sich, von einem 33K Ohm Widerstand auszugehen. Damit finden wir einen Wert für C von 22n Farad. Nachrechnen:

$$\frac{1}{2\pi \times 33 \times 10^3 \times 22 \times 10^{-9}} = 219Hz$$

(Der Wert, der 250Hz am nächsten kommt)

Wenn wir diese Werte in den Simulator eingeben, beobachten wir tatsächlich eine Verstärkung von -3dB um 242Hz, was genau dem entspricht, was wir suchen.

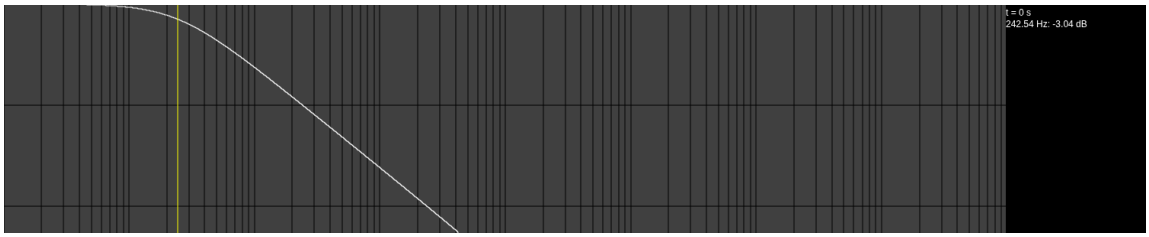

Figure 8: TP Grenzfrequenz

3.5 Bandpassfilter

Nachdem wir unsere ersten beiden Filter dimensioniert haben, die sich mit unseren Frequenzextremen befassen, müssen wir uns mit den Zwischenwerten befassen. Wir werden zwei Bandpassfilter verwenden, einen für die Tiefmitten und einen für die Hochmitten. Beachten Sie, dass das Lastenheft keine Verstärkung erfordert. Sie sind wie folgt aufgebaut:

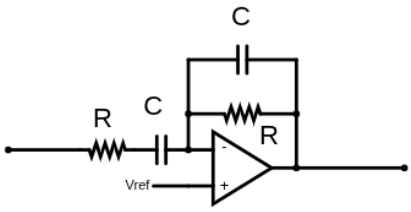

Figure 9: Bandpass-Schaltplan

Und um sie zu dimensionieren, verwenden wir die Formel:

$$f_0 = \frac{1}{2\pi RC}$$

3.5.1 Tiefer Bandpassfilter (Tiefmitten)

Dieser Filter muss ein Band zwischen 250Hz und 1KHz erzeugen. Bei Anwendung der Formel finden wir:

$$Für f_0 = 250Hz und f_1 = 1KHz :$$

$$RC_1 = \frac{1}{2\pi f_0} = \frac{1}{500\pi} = 6.367 \times 10^{-4}$$
$$RC_2 = \frac{1}{2\pi f_1} = \frac{1}{2000\pi} = 1.59 \times 10^{-4}$$

Wir setzen $R = 6.8K\Omega$:

$$6.8 \times 10^3 \times C_1 = 6,367 \times 10^{-4}$$
$$<=> C_1 = \frac{6,367 \times 10^{-4}}{6.8 \times 10^3} \approx 10 \times 10^{-8}$$

$$6.8 \times 10^3 \times C_2 = 1.59 \times 10^{-4}$$
$$<=> C_2 = \frac{1.59 \times 10^{-4}}{6.8 \times 10^3} \approx 2.52 \times 10^{-8}$$

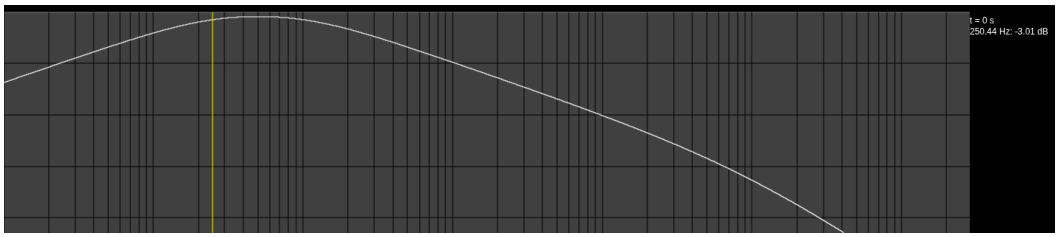

Figure 10: Bode-Diagramm mit Grenzfrequenz bei 250Hz

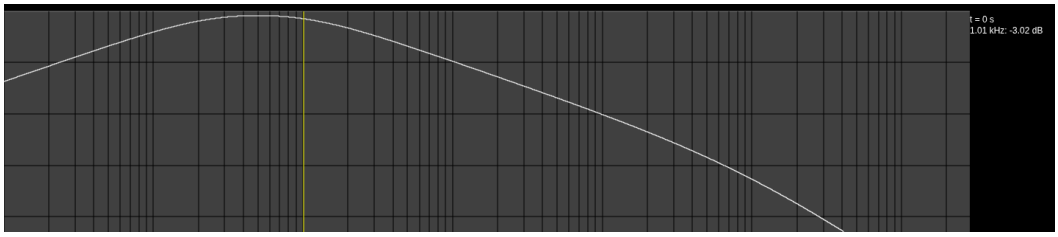

Figure 11: Bode-Diagramm mit Grenzfrequenz bei 1KHz

Wir finden dann $C_1 = 100n$ Farad und $C_2 = 2.52n$ Farad, was mit der E6-Reihe zu 22n Farad wird, da dies einen Wert näher an f_1 ergibt als 32n Farad.

Dies ergibt im Bode-Diagramm eine erste Grenzfrequenz bei 250 und eine zweite Grenzfrequenz bei 1KHz.

3.5.2 Hoher Bandpassfilter (Hochmitten)

Dieser Filter muss ein Band zwischen 1KHz und 4KHz erzeugen. Bei Anwendung der Formel finden wir:

$$\text{Für } f_0 = 1\text{KHz und } f_1 = 4KHz:$$

$$RC_1 = \frac{1}{2\pi f_0} = \frac{1}{2000\pi} = 1.59 \times 10^{-4}$$
$$RC_2 = \frac{1}{2\pi f_1} = \frac{1}{8000\pi} = 3.978 \times 10^{-5}$$

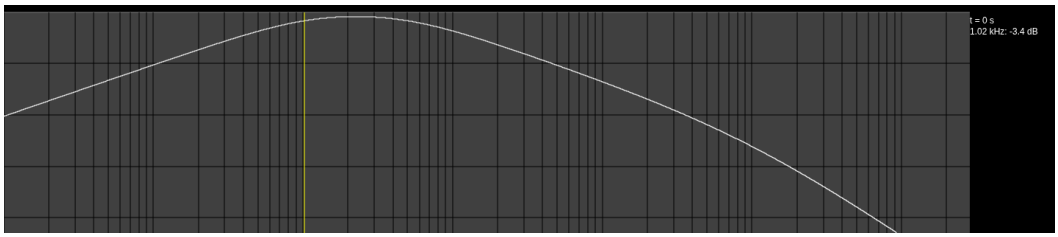

Figure 12: Bode-Diagramm mit Grenzfrequenz bei 1,1KHz

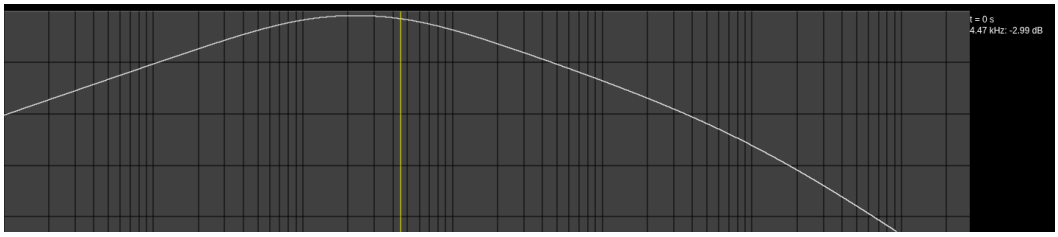

Figure 13: Bode-Diagramm mit Grenzfrequenz bei 4,5KHz

Wir setzen dann $R = 1.5K\Omega$:

$$1.5 \times 10^3 \times C_1 = 1.59 \times 10^{-4}$$
$$<=> C_1 = \frac{1.59 \times 10^{-4}}{1.5 \times 10^3} \approx 10 \times 10^{-8}$$

$$1.5 \times 10^3 \times C_2 = 3.978 \times 10^{-5}$$
$$<=> C_2 = \frac{3.978 \times 10^{-5}}{1.5 \times 10^3} \approx 26 \times 10^{-9}$$

Wir finden ähnliche Werte wie beim ersten Bandpass; wir werden den zweiten Kondensator ebenfalls auf 22nF setzen.

Dies ergibt im Bode-Diagramm eine erste Grenzfrequenz bei 1,1KHz und eine zweite Grenzfrequenz bei 4,5KHz, was kein Problem ist, da wir den Überschuss über das PICKIT-Programm verwalten können.

3.6 Signal-Hüllkurve

Eine der Herausforderungen des Projekts besteht darin, dass der Mikrocontroller die verschiedenen Pegel der 4 Filter über die LED-Matrix anzeigen muss. Um seine Arbeit zu erleichtern und insbesondere die Visualisierung dieser Pegel lesbarer zu machen, müssen die Signale eingehüllt (geglättet) werden. Das heißt, geglättet, wenn sie sich in einer absteigenden Phase befinden. Hierfür verwenden wir eine Hüllkurvenschaltung:

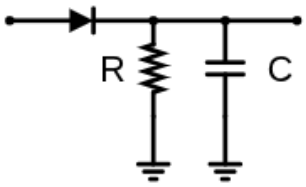

Figure 14: Schaltplan Hüllkurvenschaltung
black

Um diese Hüllkurven zu dimensionieren, können wir sagen, dass sie als Filter mit einer kleinen Besonderheit wirken. Sie besitzen eine Diode, die nur positive Halbwellen durchlässt; vereinfacht gesagt, wenn das Signal auf einer steigenden Flanke ist, lädt sich der Kondensator auf, dann, wenn das Signal auf eine fallende Flanke übergeht, wirkt die Diode als Barriere. Der Kondensator kann sich dann in Richtung des Mikrocontrollers entladen, was das Signal glättet, bis zur nächsten steigenden Flanke, die die aktuelle Spannung in der Schaltung überschreitet. Nach Rücksprache mit Herrn Pirog muss diese Reaktionszeit etwa 20ms betragen. Wir werden dann die Grenzfrequenzformel verwenden und umstellen:

$$f_{cutoff} = \frac{1}{2\pi RC}$$

Durch Umstellen erhalten wir:

$$RC = \frac{1}{2\pi \times f_{cutoff}}$$

Wir müssen nur die Dimensionsanalyse durchführen, um besser zu verstehen, woher diese Zeit kommt:

$$f_{cutoff} = s^{-1}$$

$$2\pi = 1$$

daher:

$$RC = \frac{1}{2\pi \times f_{cutoff}} = \frac{1}{1 \times s^{-1}} = s$$

Durch diese Analyse verifizieren wir, dass RC Sekunden ergibt.

Wir können daher die Gleichung $RC = 20 * 10^{-3}$ lösen. Wir benötigen einen ausreichend hohen Widerstand, damit der Kondensator klein ist und wenig Zeit zum Aufladen benötigt. Ein Wert von 100K Ohm passt perfekt. Mit E6-Werten finden wir einen Kondensator bei 220n Farad, was eine Reaktionszeit von 22ms ergibt, was weitgehend akzeptabel ist.

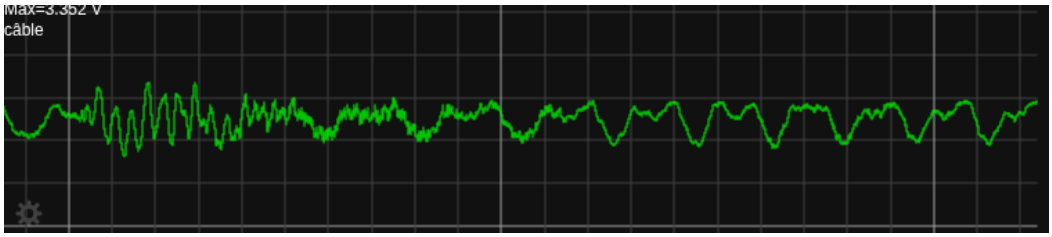

Figure 15: Beispiel vor Hüllkurve

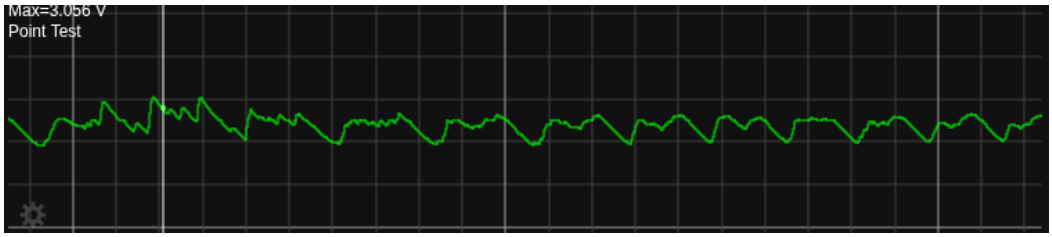

Figure 16: Beispiel nach Hüllkurve

4 Algorithmik und Programmierung

Um das Pickit zu programmieren, müssen zwei Codes erstellt werden: einer in C-Sprache für die Analog-Digital-Wandlung und Datenverarbeitung und einer in Assembler für den LED-Steuerungsteil.
Hier sind die beiden Codes:

4.1 Main.c

```
    /* ------------------------------------------------------------
 * Datei       :  main.c
 * Autor(en)   :
 * Beschreibung:
 * ------------------------------------------------------------ */

#include <xc.h>

// Hardware-Konfiguration des PIC :
#pragma config FEXTOSC = OFF            // Keine externe Taktquelle
#pragma config RSTOSC = HFINTOSC_64MHZ // Interner 64 MHz Takt
#pragma config WDTE = OFF              // Watchdog deaktivieren

#define _XTAL_FREQ 64000000 // Taktfrequenz - notwendig für Delay-Makros (_delay(N) ; __dela

// Definition von Masken, Makros, etc. :
// TODO


// Deklaration globaler Funktionen und Variablen, damit C- und ASM-Code sie teilen können
// Eine gleiche Funktion oder Variable auf der ASM-Seite wird mit einem Unterstrich präfixie
// Mit diesem Formalismus sind sie austauschbar und transparent nutzbar:
// | ---- asm ----- | ------------- C ----------------- |
// | _TX_64LEDS  <--|--> void TX_64LEDS(void)           |
// | _pC         <--|--> volatile const char * pC       |
// | _LED_MATRIX <--|--> volatile char LED_MATRIX [256] |

// Definition von Funktionen relativ zur LED-Matrix:
extern void TX_64LEDS(void); // In tx.asm definierte Funktion; Ermöglicht das Senden des Bef

// Definition von Konstanten / Variablen relativ zur LED-Matrix:
volatile char LED_MATRIX [256] ; // Definition einer 64 x 4 Byte Matrix mit R/G/B/W-Komponen
volatile const char * pC = LED_MATRIX; // Zeiger auf LED_MATRIX


char ad_read() {
    ADCON0bits.ADGO = 1;              // Start der Konvertierung
```

```
    while(ADCON0bits.ADGO == 1) {}    // Warten auf Ende
    return ADRESH;                    // Gibt die 8 MSB-Bits zurück (Wert zwischen 0 und 255
}


void get_color(unsigned char n, unsigned char color[4]) {
    // Gibt je nach Wert von ‘n‘ eine andere Farbe zurück
    if (n <= 3) {
        color[0] = 0;
        color[1] = 16;   // Grün
        color[2] = 0;
        color[3] = 0;
    } else if (n <= 5) {
        color[0] = 16;   // Orange (rot + ein wenig grün)
        color[1] = 8;
        color[2] = 0;
        color[3] = 0;
    } else {
        color[0] = 16;   // Rot
        color[1] = 0;
        color[2] = 0;
        color[3] = 0;
    }
}


void FILL_LED_MATRIX(unsigned char levels[4]) {
    for (int i = 0; i < 256; i++) {
        LED_MATRIX[i] = 0; // Gesamte Matrix zurücksetzen
    }

    for (int level = 0; level < 4; level++) {
        unsigned char l = levels[level];
        unsigned char color[4];
        get_color(l, color); // Farbe für diesen Pegel bestimmen

        for (int led_index = 0; led_index < l + 1; led_index++) {
            int led_index_adjacent = led_index + 8;
            for(int j = 0; j < 4; j++) {
                LED_MATRIX[led_index + j * level] = color[j];           // Haupt-LED
                LED_MATRIX[led_index_adjacent + j * level] = color[j]; // Benachbarte LED
            }
        }
    }
}
```

```
// - Hauptfunktion ------------------------------------------------------------------
void main(void) {
    // --- Konfiguration der analogen Eingänge (RA2 bis RA5) ---
    TRISAbits.TRISA2 = 1; // Eingang
    TRISAbits.TRISA3 = 1;
    TRISAbits.TRISA4 = 1;
    TRISAbits.TRISA5 = 1;

    ANSELAbits.ANSELA2 = 1; // Analoger Eingang
    ANSELAbits.ANSELA3 = 1;
    ANSELAbits.ANSELA4 = 1;
    ANSELAbits.ANSELA5 = 1;

    // --- ADC Konfiguration ---
    ADPCH = 0b000010;        // Auswahl des ersten Eingangs (RA2)
    ADCON0bits.ADON  = 1;    // Aktivierung des ADC-Moduls
    ADCON0bits.ADCONT = 0;   // Einzelne Konvertierung
    ADCON0bits.ADCS = 1;     // ADC Takt
    ADCON0bits.ADFM = 0;     // Ergebnis linksbündig
    ADCON0bits.ADGO = 0;     // Konvertierung inaktiv

    // --- Konfiguration der Ausgangsports ---
    TRISC &= 0x00;           // PORTC als Ausgang
    LATC = 0x00;

    TRISBbits.TRISB4 = 0;    // Ausgang zum Auslösen von TX
    TRISBbits.TRISB5 = 0;

    // --- Messvariablen ---
    char bas, bande_bas, bande_haut, haut;

    // --- Hauptschleife ---
    while(1) {
        LATBbits.LATB4 = 1; // Signalisierung oder Trigger
        //(gemäß Hardwareschema zu dokumentieren)

        // Lesen der 4 analogen Sensoren
        bande_bas = ad_read();
        ADPCH = 0b000011;
        bande_haut = ad_read();
        ADPCH = 0b000100;
        haut = ad_read();
        ADPCH = 0b000101;
        bas = ad_read();
        ADPCH = 0b000010;
```

```
        // Pegelberechnung auf 4 Bändern (0 bis 3)
        unsigned char levels[4];
        levels[0] = (bas - 128 < 0 ? 0 : bas - 128) / 32;
        levels[1] = (bande_bas - 128 < 0 ? 0 : bande_bas - 128) / 32;
        levels[2] = (haut - 128 < 0 ? 0 : haut - 128) / 32;
        levels[3] = (bande_haut - 128 < 0 ? 0 : bande_haut - 128) / 32;

        // Funktionsaufruf
        FILL_LED_MATRIX(levels);

        // Senden der Daten an LEDs
        TX_64LEDS();

        __delay_ms(50); // Kleine Pause um zu schnelles
        //Aktualisieren zu vermeiden
    }
}
```

4.2 Tx.asm

```
    #include <xc.inc>

; Wenn Assemblercode in ein psect platziert wird,
//kann er vom Linker als Ganzes
; manipuliert und im Speicher platziert werden.
psect   txfunc,local,class=CODE,reloc=2 ;
//PIC18 sollten ein reloc (Ausrichtungs-) Flag von
;2 für jedes psect haben, das ausführbaren Code enthält.

; ----------------------------------------------------------------
; GLOBALS
;
; Deklaration globaler Funktionen und Variablen
;die es C- und ASM-Code ermöglichen, sie zu teilen
; Eine gleiche Funktion oder Variable auf der ASM-Seite wird mit einem Unterstrich präfixie
;und auf der C-Seite nicht
; Mit diesem Formalismus sind sie austauschbar
;und transparent nutzbar :
; | ---- asm ----- | ------------- C ----------------- |
; | _TX_64LEDS  <--|--> void TX_64LEDS(void)           |
; | _pC         <--|--> volatile const char * pC       |
; | _LED_MATRIX <--|--> volatile char LED_MATRIX [256] |

; Globale Funktionen
global _TX_64LEDS ; In tx.asm definierte Funktion; Funktion die es ermöglicht
```

```
;den Befehl zur Steuerung der 64 LEDs zu senden,
;wie in LED_MATRIX beschrieben

; Globale Konstanten/Variablen
global _pC          ; In main.c definierte Konstante
; Zeiger auf LED_MATRIX
global _LED_MATRIX ; In main.c definierte Variable
; Array (256 Bytes = 64 x 4)
;der RGBW-Komponenten der LED-Matrix (1 Byte/Farbe/LED)


; ------------------------------------------------------------------------------
; Bit-Sendemakros für Kommunikation mit adressierbaren LEDs
; ------------------------------------------------------------------------------
; Sendet ein '1' Bit:
; Zeit im High-Zustand ? 750 ns (12 NOPs)
; Zeit im Low-Zustand  ? 250 ns (4 NOPs)
SEND_1 MACRO
    BSF LATB, 5        ; Setzt Pin RB5 auf 1 (HIGH)
    NOP                ; Warten (~62.5 ns bei 64 MHz)
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    BCF LATB, 5        ; Setzt Pin RB5 auf 0 (LOW)
    NOP
    NOP
    NOP
    NOP
ENDM

; Sendet ein '0' Bit:
; Zeit im High-Zustand ? 250 ns (4 NOPs)
; Zeit im Low-Zustand  ? 560 ns (9 NOPs)
SEND_0 MACRO
    BSF LATB, 5        ; Setzt Pin RB5 auf 1 (HIGH)
    NOP
    NOP
    NOP
    NOP
```

```
    BCF LATB, 5        ; Setzt Pin RB5 auf 0 (LOW)
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
ENDM

; ------------------------------------------------------------------------
; Routine SEND_8: Sendet 8 Bits (1 Byte) gelesen aus LED_MATRIX via FSR0
; ------------------------------------------------------------------------
SEND_8:
    MOVF POSTINC0, 0, 0   ; Holt das von FSR0 gezeigte Byte, inkrementiert dann
    ;den Zeiger

    ; Bit 7 (MSB)
    BTFSC WREG, 7         ; Testet, ob Bit 7 auf 1 ist
    GOTO SEND_BIT_7
    SEND_0
    GOTO END_B7
    SEND_BIT_7:
    SEND_1
    END_B7:

    ; Bit 6
    BTFSC WREG, 6
    GOTO SEND_BIT_6
    SEND_0
    GOTO END_B6
    SEND_BIT_6:
    SEND_1
    END_B6:

    ; Bit 5
    BTFSC WREG, 5
    GOTO SEND_BIT_5
    SEND_0
    GOTO END_B5
    SEND_BIT_5:
    SEND_1
    END_B5:
```

```
; Bit 4
BTFSC WREG, 4
GOTO SEND_BIT_4
SEND_0
GOTO END_B4
SEND_BIT_4:
SEND_1
END_B4:

; Bit 3
BTFSC WREG, 3
GOTO SEND_BIT_3
SEND_0
GOTO END_B3
SEND_BIT_3:
SEND_1
END_B3:

; Bit 2
BTFSC WREG, 2
GOTO SEND_BIT_2
SEND_0
GOTO END_B2
SEND_BIT_2:
SEND_1
END_B2:

; Bit 1
BTFSC WREG, 1
GOTO SEND_BIT_1
SEND_0
GOTO END_B1
SEND_BIT_1:
SEND_1
END_B1:

; Bit 0 (LSB)
BTFSC WREG, 0
GOTO SEND_BIT_0
SEND_0
GOTO END_B0
SEND_BIT_0:
SEND_1
END_B0:

RETURN                ; Rückkehr zum Aufrufer
```

```
; ------------------------------------------------------------------------
; Routine _TX_64LEDS : sendet die 256 Bytes von LED_MATRIX an die LEDs
; ------------------------------------------------------------------------
_TX_64LEDS:
    ; Laden des Zeigers auf LED_MATRIX in FSR0
    MOVFF _pC + 0, WREG  ; Lädt niedrige Adresse (LSB) von LED_MATRIX
    MOVWF FSR0L, 0       ; Weist FSR0L zu (indirekte Adresse)
    MOVFF _pC + 1, WREG  ; Lädt hohe Adresse (MSB)
    MOVWF FSR0H, 0       ; Weist FSR0H zu

    BANKSEL LATB         ; Auswahl des LATB-Registers (korrekte Bank)

    ; Schleife zum Senden der 256 Bytes von LED_MATRIX (64 LEDs × 4 Bytes)
    REPT 256
        CALL SEND_8      ; Sendet 1 Byte via SEND_8 Routine
    ENDM
```

5 Fazit und Tests

Aus Zeit- und Organisationsmangel wurden keine Fotos der Oszillogramme gemacht; ein Teil der Tests basiert daher auf Deduktion und Theorie.

Nachdem die Dimensionierung und Programmierung abgeschlossen waren, war es Zeit zu testen; leider wurde kein schlüssiger Test durchgeführt. Was wir sagen können ist, dass die Dimensionierung erfolgreich war; das Team konnte den Schallpegel einer der 4 Komponenten auf den LEDs der Platine anzeigen. Beim Wechsel zur LED-Matrix funktionierte es jedoch nicht; es scheint, dass dies ein Problem auf Ebene des Assembler-Codes war.

Dieses Projekt ermöglichte es dem Team, die Bedeutung einer ordnungsgemäßen Dimensionierung besser zu verstehen, ohne die das fertige Projekt unsicher sein oder nicht vollständig funktionieren könnte. Mit einem Minimum an zusätzlicher Zeit oder möglichem Zugang zur Elektronikplatine würde die Gruppe gerne weiter daran arbeiten, um die Möglichkeiten zu sehen, die das Projekt ihnen bietet.

5.1 Bonus

Als kleiner Bonus und um die Richtigkeit unserer Ergebnisse per Simulation zu überprüfen, hier ist die Simulation der kompletten Schaltung. Einfach auf "No File" doppelklicken und eine MP3-Datei einfügen. Schaltungssimulation

Und hier ist die ausgeschnittene Schaltung mit den Werten, die in Falstad kopiert/eingefügt werden können, um die Bode-Diagramme zu überprüfen: Ausgeschnittene Schaltung

5.2 Credits

PAUL-LOUP VEREBELY : BERICHT/DIMENSIONIERUNG/LÖTEN

HUGO DUPRAT : PROGRAMMIERUNG

NOE DIBLASI : DIMENSIONIERUNG/PROGRAMMIERUNG