ELECTRONICS LAB REPORT

By VEREBELY Paul-loup & $HugoDUPRAT$ & $NoéDIBLASI$

June 2025

JUNIA Grande école d'ingénieurs

Contents

List of Figures

1 Introduction

This document outlines the entire process of the end-of-year AP3 electronics project. To ensure clear and structured reading, the report is organized into several distinct sections.

First, the context section presents the project objectives, the initial specifications, as well as the main constraints and challenges identified.

The sizing section then exposes all calculations, hypotheses, and technical choices made by the team to design the electronic system meeting the specifications.

The software section describes in detail the software development associated with the project, specifying the programming logic, code architecture, and main implemented features.

Finally, the tests and difficulties encountered section presents the different system validation steps, technical problems encountered during the project, and the solutions provided by the team to remedy them.

Regarding the team distribution, it is composed of 3 members who all participated in the different aspects of the project.

This report also aims to serve as an example in case of reproduction of the project by the reader(s).

2 Initial Specifications

This project allows for the consolidation of all knowledge acquired in electronics during the year. To put all this into a feasible project within a minimum amount of time, Antoine PIROG wrote a complete and precise set of specifications. Here is the content:

The system to be designed is an RGBW LED VU-meter intended to visualize the sound level of an audio signal in real-time. The project architecture is articulated around several functional blocks described below.

2.1 Power Supply

- Single supply voltage: **+5V DC**
- This voltage powers the entire electronic board.
- It must be strictly respected to avoid damaging sensitive components (notably the LEDs and the microcontroller).

2.2 Acquisition and Audio Signal Conditioning

- The input audio signal is divided into two paths:
  - one towards the analog filters,
  - one towards an audio output (to active speakers).
- The signals are then filtered by 4 analog band-pass filters defining the frequency bands:
  - **Bass**: < 250 Hz
  - **Low-mids**: 250 Hz – 1 kHz
  - **High-mids**: 1 kHz – 4 kHz
  - **Treble**: > 4 kHz
- Each filter must be correctly sized using Op-Amps and passive components.

2.3 Digital Processing

- The **PIC18F25K40** microcontroller performs:
  - analog acquisition of signals from the filters,
  - digital processing of the level in each band,
  - data formatting for display.
- The code is written in **C language** for general logic and in **Assembly** for LED control (high-precision bitbanging).

2.4 LED Display

- Display on a matrix of **64 SK6812 RGBW LEDs**, individually addressable.
- Each LED is controlled via a serial protocol, according to a sequence of **2048 bits** (64 LEDs $\times$ 4 colors $\times$ 8 bits).
- The color order is: **Green**, **Red**, **Blue**, **White**.
- The intensity of each color is coded on 8 bits (recommended values: 16 to 32 to avoid excessive brightness).

2.5 User Interface and Tests

- **9 Test LEDs** and **2 Push Buttons** available to:
  - test the program independently of the LED matrix,
  - implement additional features (pause, display modes...).

2.6 Development Constraints

- **Project duration**: 7 sessions
- **Expected deliverables**:
  - Functional prototype,
  - Complete source code (C + Assembly),
  - Written report including schematics, calculations, Bode plots, oscillograms, component bill of materials.
- **Tests to perform**:
  - Oscillograms on Test Points,
  - Validation of filters with function generator,
  - Control of assembly timing with oscilloscope.

3 Electronic Sizing

In this project, sizing is an important part that must not be neglected. Several parts of the board must be sized correctly to maintain a high level of performance and good circuit integrity. To describe all this, the sizing will be done in several blocks to facilitate the work and understanding. Finally, for safety reasons, all sizing tests will be done on Falstad and using the E6 series.

3.1 Reference Voltage

When looking at the electrical schematic, one can notice that a good part of the Op-Amps are powered with Vref. This voltage acts as a reference voltage; it will allow our audio signal to oscillate around 2.5V because the PICKIT operates between 0 and 5 Volts. For this, we find a voltage divider powered by 5V as well as a voltage follower Op-Amp configuration.

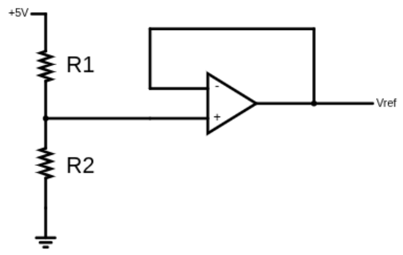

Figure 1: Voltage Divider Schematic

Voltage Divider Formula: $Vref = U * \frac{R2}{R2+R1}$ Voltage Follower Op-Amp Formula: $Vs = Ve$

Given that our input voltage is 5V and we want 2.5V at the output, we just need to divide the voltage by two. For this, and thanks to the voltage divider, we will see that resistors Rref1 and Rref2 will be equal. for simplicity, the value of 1K Ohm will be kept for both resistors.

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

$$Vref = 2.5V$$

3.2 Pre-amplification

The circuit receives the audio signal via the first jack socket. It should be noted that the voltage applied in this cable varies between cables and computers. The important thing to remember is that it is very low. In the team's case, where Hugo's PC is used, an average voltage of 150 mV is observed. We must then go through a pre-amplification step to increase the input signal and be able to work on it correctly, here averaging 2.5V. In the assembly, we observe two pre-amplifiers, a right and a left one, but they are identical. The sizing of only one will therefore be necessary.

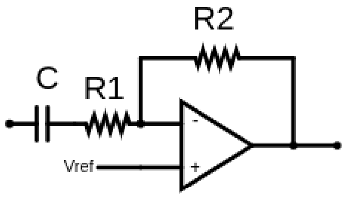

Figure 2: Pre-amplifier Schematic

In this assembly, we observe a non-inverting setup preceded by a capacitor. Here we know that our output voltage must be a maximum of 5V and an average of 2.5V. For this, we will calculate the necessary Gain and size our resistors accordingly.
As for the capacitor, it allows several things. It acts as a barrier between the input signal with an unknown average voltage and the average voltage of the Op-Amp which is at 2.5V due to its Vref supply. This avoids conflicts between these two zones because without it, the Op-Amp which applies its gain around 2.5V will certainly amplify, but the jack part will not be amplified correctly or at all. To size it, we can say that it acts as a high-pass filter with the resistor that follows it. This also allows eliminating frequencies that are too low thanks to its cutoff frequency. Its sizing will be done so that the cutoff frequency is at 20Hz, the minimum frequency audible to a human.
Non-inverting Op-Amp Formula: $Vs = (1 + \frac{Rpre1}{Rpre0}) * Ve$

High-pass filter Formula: $\frac{1}{2\pi * Rpre0 * Cpre0}$

As for the gain, we must make the ratio of 150mV and 2.5V:

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

With the E6 series and our formula we find:

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

Thanks to this, we find for Rpre0 and Rpre1 values of 75K Ohm and 4.7K Ohm. Let's move on to the capacitor sizing: $20 = \frac{1}{2\pi * 4.7 * 10 * C}$

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

With the component series, the value that comes closest is 2.2uF which gives us a cutoff frequency at 15.40Hz.

3.3 Left/Right Mixing

The mixing part serves to decide if we want the left or right side of our audio signal. In our circuit, we find a potentiometer; it works like 2 variable resistors connected to Vref, its maximum value is 10K Ohm. We also find a somewhat particular summing amplifier setup. Its sizing will be done step by step so as not to lose the reader.
Like for the pre-amplifiers, we can study one side of the circuit because nothing changes between left and right. To simplify calculations, the circuit will not be raised with Vref but to ground. Here is our circuit:

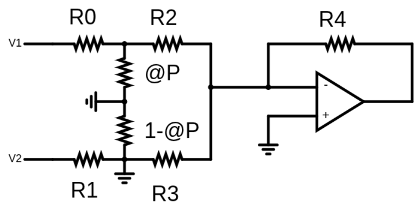

Figure 3: Mixing Schematic

And here is how we will interpret it:

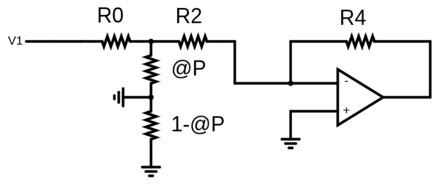

Figure 4: Interpretative Mixing Schematic

We assume that $V_2 = 0$, and that the potentiometer delivers its entire bandwidth on V_1. We will perform two calculations according to Millman's theorem, a first one between R_0, R_2 and αP, noted @P, and a second at the inverting terminal of the circuit.

First Millman on V^-:
Let V_a be the voltage across R_2.

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

Second Millman between R_0, R_2 and αP to calculate V_a:

$$\begin{aligned}V_a &= \frac{\frac{V_L}{R_0}+\frac{0}{\alpha P}+\frac{V^-}{R_2}}{\frac{1}{R_0}+\frac{1}{\alpha P}+\frac{1}{R_2}}\\ &= \frac{V_L\alpha P R_2 + V^- R_0\alpha P}{\alpha P R_2 + R_0R_2 + R_0\alpha P}\end{aligned}$$

$$\begin{aligned}V^- &= \frac{V_aR_4+V_sR_2}{R_2+R_4}\\ &= \left(\frac{V_L\alpha P R_2 + V^- R_0\alpha P}{\alpha P R_2 + R_0R_2 + R_0\alpha P}\right)\times\frac{R_4}{R_2+R_4}+\frac{V_sR_2}{R_2+R_4}\\ &= \frac{V_LR_2}{R_0+R_2+\frac{R_0R_2}{\alpha P}}\times\frac{R_4}{R_2+R_4}+\frac{V_sR_2}{R_2+R_4}=0\\ V_L &\times\frac{R2}{R_0+R_2+\frac{R_0R_2}{\alpha P}}\times\frac{R_4}{R_2+R_4}=-\frac{V_sR_2}{R_2+R_4}\\ &\leftrightarrow V_L\times\frac{R2}{R_0+R_2+\frac{R_0R_2}{\alpha P}}\times\frac{R_4}{R_2+R_4}\times\frac{R_2+R_4}{R_2}=-V_s\end{aligned}$$

$$\begin{aligned}V_L &\times\frac{R2}{R_0+R_2+\frac{R_0R_2}{\alpha P}}\times\frac{R_4}{R_2+R_4}\times\frac{R_2+R_4}{R_2}=-V_s\\ &\leftrightarrow\frac{R_4}{R_0+R_2+\frac{R_0R_2}{\alpha P}}=-\frac{V_s}{V_L}\end{aligned}$$

We then find that the ratio between the output and the input of the Op-Amp is:

$$-\frac{V_s}{V_L}=\frac{R_4}{R_0+R_2+\frac{R_0R_2}{\alpha P}}$$

For sizing, we need to find values for R_0, R_2 and R_4 that will yield the values:

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

After tests on Mr. PIROG's simulation, the team concluded that values of 47K, 68K, and 470K for R_0, R_2, and R_4 respectively were promising values. These values therefore also apply respectively for R_1, R_3.

3.4 High Pass / Low Pass Filter

The study of filters will begin with the two filters: high pass and low pass. For their sizing, the element sought is the cutoff frequency at -3dB. To calculate this frequency we use the formula

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

3.4.1 High Pass Filter

Let's start with the high pass filter. Its role is to let through only high frequencies above 4KHz (its cutoff frequency) with a gain of -3dB. The assembly is constructed as follows:

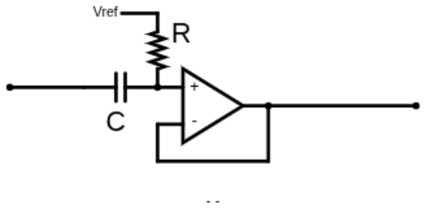

Figure 5: High Pass Filters

To size this filter, we must therefore do

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

Still with the E6 series, the team chooses a resistance value of 820 Ohm. We then find a capacitor value of 47n Farad, with a gain of -3.16dB at 4KHz.

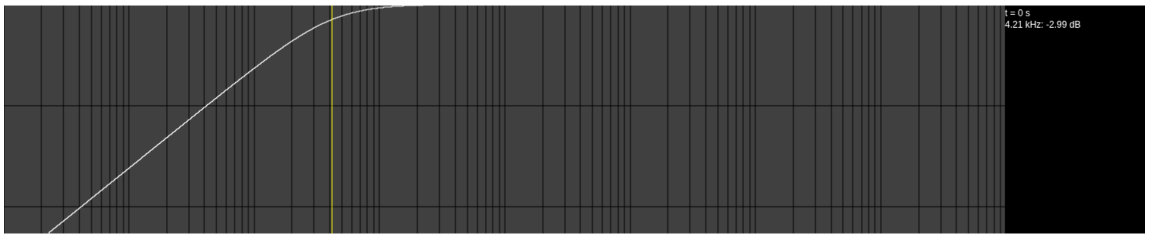

Figure 6: HP Cutoff Frequency

3.4.2 Low Pass Filter

The low pass filter's role is to let through only low frequencies below 250Hz (its cutoff frequency) with a gain of -3dB. The assembly is performed as follows:

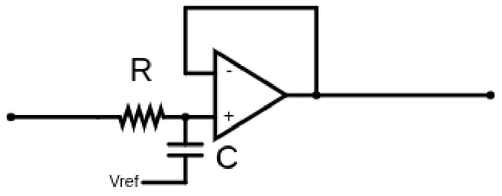

Figure 7: Low Pass Filter

By rearranging the expression of the cutoff frequency we find:

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

The team chose to base itself on a 33K Ohm resistor. With this, we find a value for C of 22n Farad. Recalculating:

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

(The value closest to 250Hz)

By passing these values to the simulator, we actually observe a gain of -3dB around 242Hz, which is exactly what we are looking for.

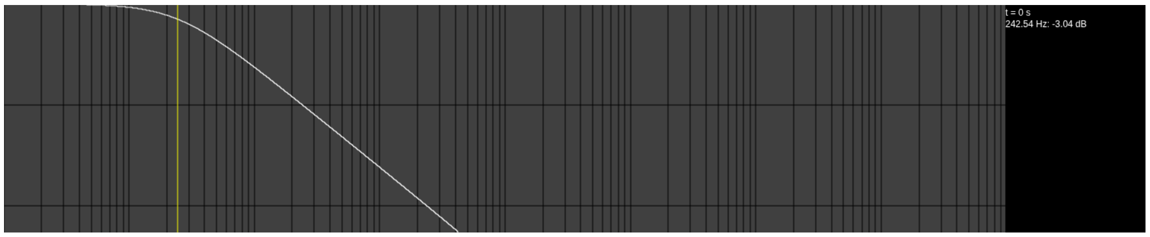

Figure 8: LP Cutoff Frequency

3.5 Band-Pass Filters

After having sized our first two filters which deal with our frequency extremes, we must deal with the intermediate values. We will use two band-pass filters, one for low-mids and one for high-mids. Note that the specifications require no gain. They are constructed as follows:

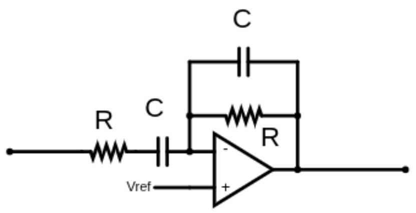

Figure 9: Band-Pass Schematic

And to size them we will use the formula:

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

3.5.1 Low Band-Pass Filter

This filter must create a band between 250Hz and 1KHz. When applying the formula we find:

$$For f_0 = 250Hz and f_1 = 1KHz :$$

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

We set $R = 6.8K\Omega$:

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

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

We then find $C_1 = 100n$ Farad and $C_2 = 2.52n$ Farad which will become 22n Farad with the E6 series because it gives a value closer to f_1 rather than 32n Farad.

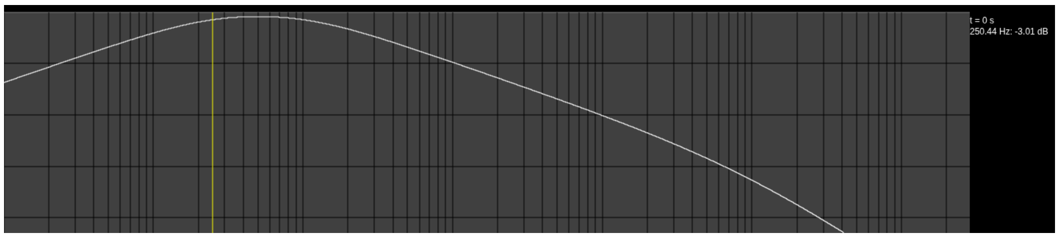

Figure 10: Bode Plot with cutoff frequency at 250Hz

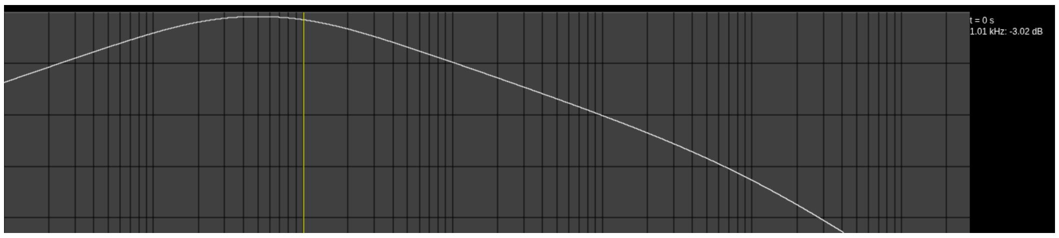

Figure 11: Bode Plot with cutoff frequency at 1KHz

This gives on the Bode plot a first cutoff frequency at 250 and a second cutoff frequency at 1KHz.

3.5.2 High Band-Pass Filter

This filter must create a band between 1KHz and 4KHz. When applying the formula we find:

$$\text{For } f_0 = 1\text{KHz and } f_1 = 4KHz:$$

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

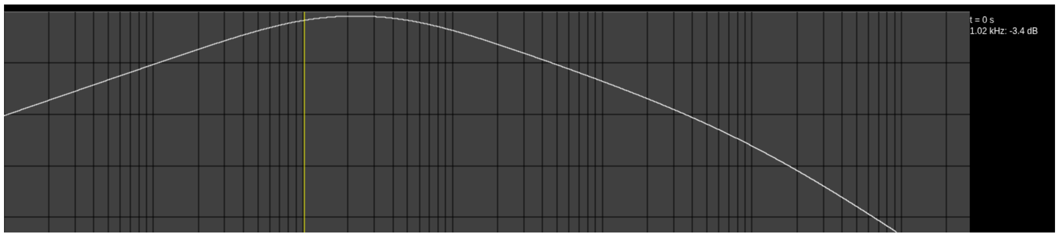

Figure 12: Bode Plot with cutoff frequency at 1.1KHz

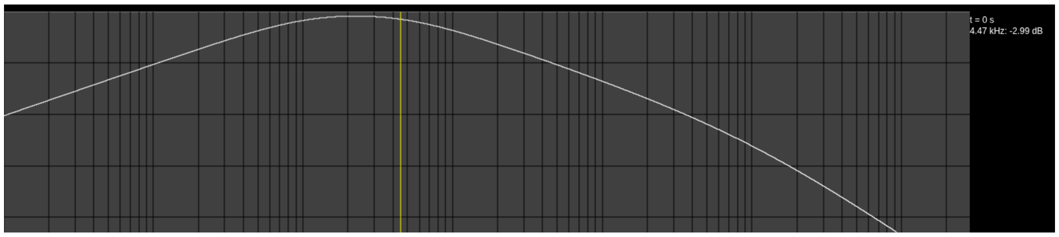

Figure 13: Bode Plot with cutoff frequency at 4.5KHz

We then set $R = 1.5K\Omega$:

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

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

We find similar values as on the first band-pass; we will also set the second capacitor to 22nF.

This gives on the Bode plot a first cutoff frequency at 1.1KHz and a second cutoff frequency at 4.5KHz, which is not an issue because we can manage the excess via the PICKIT program.

3.6 Signal Envelope

One of the challenges of the project is that the microcontroller must display the different levels of the 4 filters via the LED matrix. To facilitate its work and especially make the visualization of these levels more readable, the signals must be enveloped. That is to say, smoothed when they are in a descending phase. For this, we use an envelope circuit:

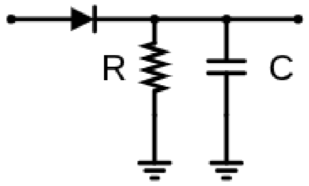

Figure 14: Envelope circuit schematic
black

To size these envelopes we can say that they act as filters with a small peculiarity. They possess a diode that allows only positive half-cycles to pass; clearly, when the signal is on a rising edge, the capacitor charges, then when the signal passes to a falling edge the diode acts as a barrier. The capacitor can then discharge towards the microcontroller which smoothes the signal until the next rising edge exceeding the current voltage in the circuit. After consultation with Mr. Pirog, this response time needs to be around 20ms. We will then use and rearrange the cutoff frequency formula noted:

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

By rearranging it we obtain:

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

We just need to do the dimensional analysis to better understand where this time comes from:

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

$$2\pi = 1$$

therefore:

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

By doing this analysis, we verify that RC will give seconds.

We can therefore solve the equation $RC = 20 * 10^{-3}$. We need a high enough resistance so that the capacitor is small and takes little time to charge. A value of 100K Ohm will do perfectly. With E6 values we find a capacitor at 220n Farad which gives a response time of 22ms which is widely acceptable.

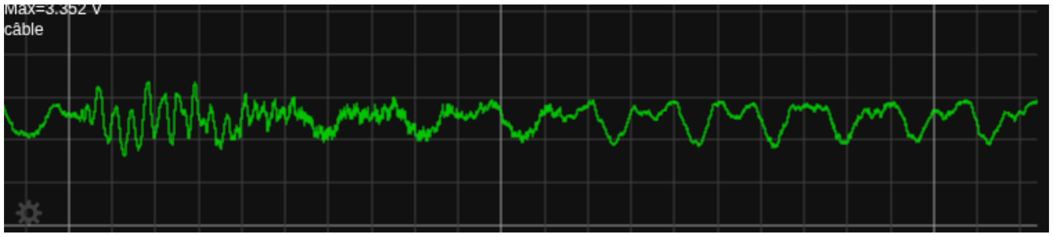

Figure 15: Example before envelope

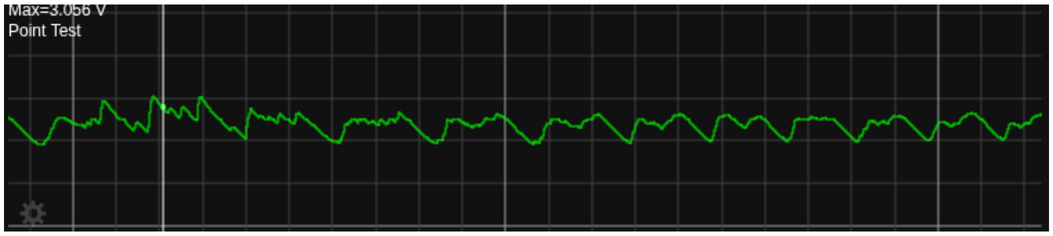

Figure 16: Example after envelope

4 Algorithmics and Programming

To program the pickit, two codes must be made: one in C language for analog-to-digital conversion and data processing, and one in assembly for the LED control part.
Here are the two codes:

4.1 Main.c

```
    /* ----------------------------------------------------------------
 * File        :  main.c
 * Author(s)   :
 * Description :
 * ---------------------------------------------------------------- */

#include <xc.h>

// Hardware Configuration of PIC :
#pragma config FEXTOSC = OFF            // No external clock source
#pragma config RSTOSC = HFINTOSC_64MHZ // Internal 64 MHz clock
#pragma config WDTE = OFF              // Disable watchdog

#define _XTAL_FREQ 64000000 // Clock frequency - necessary for delay macros (_delay(N) ; __d

// Definition of masks, macros, etc. :
// TODO


// Declaration of global functions and variables allowing C and asm code to share them
// A same function or variable on the asm side is prefixed by an underscore, and is not on t
// With this formalism, they are usable interchangeably and transparently:
// | ---- asm ----- | ------------- C ----------------- |
// | _TX_64LEDS  <--|--> void TX_64LEDS(void)           |
// | _pC         <--|--> volatile const char * pC       |
// | _LED_MATRIX <--|--> volatile char LED_MATRIX [256] |

// Definition of functions relative to the LED matrix:
extern void TX_64LEDS(void); // Function defined in tx.asm; Function allowing to send the co

// Definition of constants / variables relative to the LED matrix:
volatile char LED_MATRIX [256] ; // Definition of a 64 x 4 bytes matrix containing R/G/B/W c
volatile const char * pC = LED_MATRIX; // Pointer to LED_MATRIX


char ad_read() {
    ADCON0bits.ADGO = 1;              // Start conversion
    while(ADCON0bits.ADGO == 1) {}    // Wait for end
    return ADRESH;                    // Return the 8 MSB bits (value between 0 and 255)
}


void get_color(unsigned char n, unsigned char color[4]) {
    // Returns a different color depending on the value of ‘n‘
    if (n <= 3) {
        color[0] = 0;
        color[1] = 16;   // Green
        color[2] = 0;
        color[3] = 0;
    } else if (n <= 5) {
        color[0] = 16;   // Orange (red + a little green)
        color[1] = 8;
        color[2] = 0;
        color[3] = 0;
    } else {
        color[0] = 16;   // Red
        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; // Reset entire matrix
    }

    for (int level = 0; level < 4; level++) {
        unsigned char l = levels[level];
        unsigned char color[4];
        get_color(l, color); // Determine color for this level

        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];           // Main LED
                LED_MATRIX[led_index_adjacent + j * level] = color[j]; // Adjacent LED
            }
        }
    }
}
```

```
// - Main Function ---------------------------------------------------------------
void main(void) {
    // --- Configuration of analog inputs (RA2 to RA5) ---
    TRISAbits.TRISA2 = 1; // Input
    TRISAbits.TRISA3 = 1;
    TRISAbits.TRISA4 = 1;
    TRISAbits.TRISA5 = 1;

    ANSELAbits.ANSELA2 = 1; // Analog Input
    ANSELAbits.ANSELA3 = 1;
    ANSELAbits.ANSELA4 = 1;
    ANSELAbits.ANSELA5 = 1;

    // --- ADC Configuration ---
    ADPCH = 0b000010;        // Selection of first input (RA2)
    ADCON0bits.ADON  = 1;    // Activation of ADC module
    ADCON0bits.ADCONT = 0;   // Single conversion
    ADCON0bits.ADCS = 1;     // ADC Clock
    ADCON0bits.ADFM = 0;     // Result left justified
    ADCON0bits.ADGO = 0;     // Conversion inactive

    // --- Output ports configuration ---
    TRISC &= 0x00;           // PORTC as output
    LATC = 0x00;

    TRISBbits.TRISB4 = 0;    // Output to trigger TX
    TRISBbits.TRISB5 = 0;

    // --- Measurement variables ---
    char bas, bande_bas, bande_haut, haut;

    // --- Main Loop ---
    while(1) {
        LATBbits.LATB4 = 1; // Signaling or triggering
        //(to be documented according to hardware schematic)

        // Reading the 4 analog sensors
        bande_bas = ad_read();
        ADPCH = 0b000011;
        bande_haut = ad_read();
        ADPCH = 0b000100;
        haut = ad_read();
        ADPCH = 0b000101;
        bas = ad_read();
        ADPCH = 0b000010;
        // Level calculation on 4 bands (0 to 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;

        // Call to function
        FILL_LED_MATRIX(levels);

        // Sending data to LEDs
        TX_64LEDS();

        __delay_ms(50); // Small pause to avoid
        //refreshing too fast
    }
}
```

4.2 Tx.asm

```
    #include <xc.inc>

; When assembly code is placed in a psect,
//it can be manipulated as a
; whole by the linker and placed in memory.
psect   txfunc,local,class=CODE,reloc=2 ;
//PIC18's should have a reloc (alignment) flag of
;2 for any psect which contains executable code.

; -----------------------------------------------------------------
; GLOBALS
;
; Declaration of global functions and variables
;allowing C and asm code to share them
; A same function or variable on the asm side is prefixed
;by an underscore, and is not on the C side
; With this formalism, they are usable interchangeably
;and transparently :
; | ---- asm ----- | ------------- C ----------------- |
; | _TX_64LEDS  <--|--> void TX_64LEDS(void)           |
; | _pC         <--|--> volatile const char * pC       |
; | _LED_MATRIX <--|--> volatile char LED_MATRIX [256] |

; Global functions
global _TX_64LEDS ; Function defined in tx.asm; Function allowing
```

```
;to send the command to drive the 64 LEDs,
;as described in LED_MATRIX

; Global Constants/Variables
global _pC         ; Constant defined in main.c
; Pointer to LED_MATRIX
global _LED_MATRIX ; Variable defined in main.c
; Array (256 bytes = 64 x 4)
;of RGBW components of the LED matrix (1 byte/color/LED)


; -----------------------------------------------------------------------------
; Bit sending macros for communication with addressable LEDs
; -----------------------------------------------------------------------------
; Sends a '1' bit:
; Time at high state ? 750 ns (12 NOPs)
; Time at low state  ? 250 ns (4 NOPs)
SEND_1 MACRO
    BSF LATB, 5        ; Sets pin RB5 to 1 (HIGH)
    NOP                ; Wait (~62.5 ns at 64 MHz)
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    BCF LATB, 5        ; Sets pin RB5 to 0 (LOW)
    NOP
    NOP
    NOP
    NOP
ENDM

; Sends a '0' bit:
; Time at high state ? 250 ns (4 NOPs)
; Time at low state  ? 560 ns (9 NOPs)
SEND_0 MACRO
    BSF LATB, 5        ; Sets pin RB5 to 1 (HIGH)
    NOP
    NOP
    NOP
    NOP
    BCF LATB, 5        ; Sets pin RB5 to 0 (LOW)
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
ENDM

; ---------------------------------------------------------------------------
; SEND_8 Routine: Sends 8 bits (1 byte) read from LED_MATRIX via FSR0
; ---------------------------------------------------------------------------
SEND_8:
    MOVF POSTINC0, 0, 0   ; Retrieves the byte pointed by FSR0, then increments
    ;the pointer

    ; Bit 7 (MSB)
    BTFSC WREG, 7         ; Tests if bit 7 is at 1
    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                ; Return to caller
```

```
; -------------------------------------------------------------------------------
; Routine _TX_64LEDS : sends the 256 bytes of LED_MATRIX to the LEDs
; -------------------------------------------------------------------------------
_TX_64LEDS:
    ; Loading the pointer to LED_MATRIX into FSR0
    MOVFF _pC + 0, WREG  ; Loads low address (LSB) of LED_MATRIX
    MOVWF FSR0L, 0       ; Assigns to FSR0L (indirect address)
    MOVFF _pC + 1, WREG  ; Loads high address (MSB)
    MOVWF FSR0H, 0       ; Assigns to FSR0H

    BANKSEL LATB         ; Selection of LATB register (correct bank)

    ; Loop to send the 256 bytes of LED_MATRIX (64 LEDs × 4 bytes)
    REPT 256
        CALL SEND_8      ; Sends 1 byte via SEND_8 routine
    ENDM
```

5 Conclusion and Tests

Due to a lack of time and organization, no photos of the oscillograms were taken; a part of the tests is therefore based on deduction and theory.

After the sizing and programming were done, it was time to test; unfortunately, no conclusive test was performed. What we can say is that the sizing was achieved successfully; the team was able to display the sound level of one of the 4 components on the board's LEDs. However, when switching to the LED matrix, it did not work; it seems that this was a problem at the assembly code level.

This project allowed the team to better understand the stakes of proper sizing, without which the finished project could be uncertain or not function completely. With a minimum of extra time or possible access to the electronic board, the group would be delighted to continue working on it to see the possibilities that the project offers them.

5.1 Bonus

As a small bonus and to verify via simulation the veracity of our results, here is the simulation of the complete circuit. Just double click on No File and insert an MP3 file. Circuit Simulation

And here is the cut-out circuit with the values that can be copied/pasted into Falstad to check the Bode plots: Cut-out Circuit

5.2 Credits

PAUL-LOUP VEREBELY : REPORT/SIZING/SOLDERING

HUGO DUPRAT : PROGRAMMING

NOE DIBLASI : SIZING/PROGRAMMING