Appendix Report : DIY & Hacking

Olaf Léon[1], Thomas Mellier[1†], Yann Rives-Hapiak[1†]

[1]UFR Sciences et Techniques, A.M.U, Pl. Victor Hugo, Marseille, 13003, France.

Contributing authors: olaf.leon.91@gmail.com; thomasmellier27@gmail.com; yann.riveshapiak@gmail.com;
[†]These authors contributed equally to this work.

Abstract

This additional document constitutes the appendix of the DIY & Hacking paper written by the same authors. It presents the authors and their roles. It also condenses schematics, PCBs, code functions as well as other informations and images that goes into the detail of the project.

Keywords: DIY, Hacking, Modular synthesis, Research-Creation

Appendix A Authors Presentation

A.1 Olaf Leon (olaf.leon.91@gmail.com)

Olaf is in the Engineering and Sound Design master's degree. He was in charge of the hardware development: PCB designs and engraving.

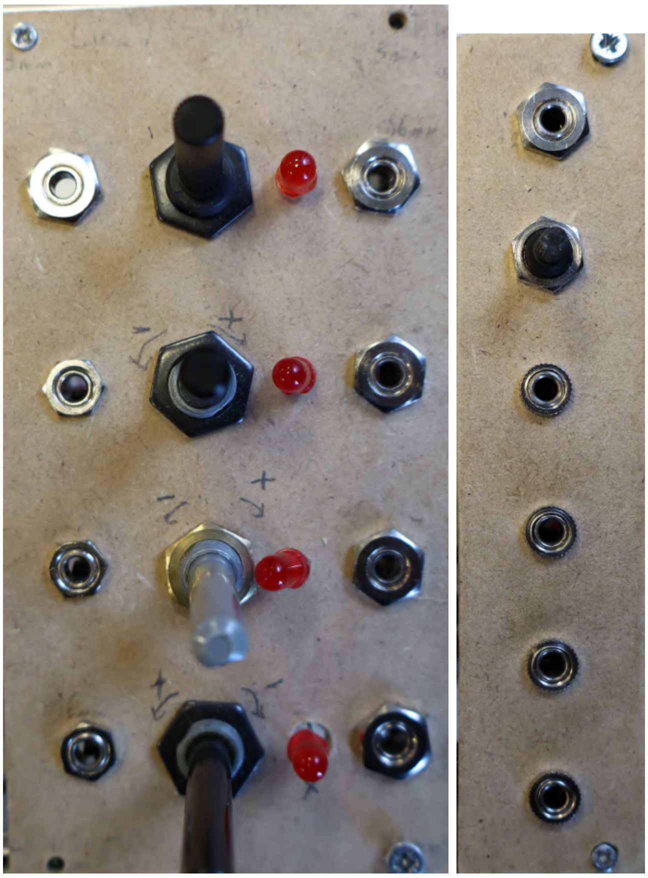
A.2 Thomas Mellier (thomasmellier27@gmail.com)

Thomas is in the Engineering and Sound Design master's degree. He was in charge of the software development and the interfacing with the STM32s.

A.3 Yann Rives-Hapiak (yann.riveshapiak@gmail.com)

Yann is in the Musicology master's degree. They were in charge of the circuit bending and the composition.

Appendix B Modules Panels



(a) Mixer IN (b) Mixer Out

Fig. B1: Mixers

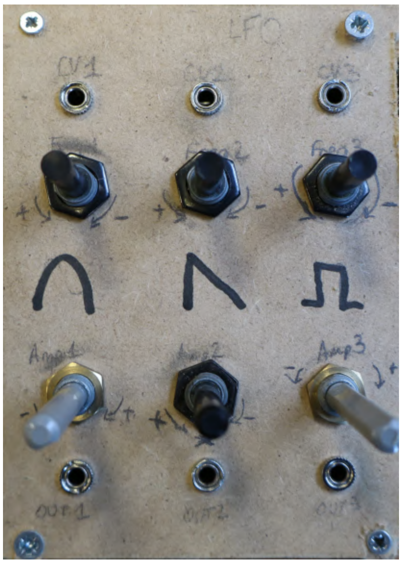


Fig. B2: LFO

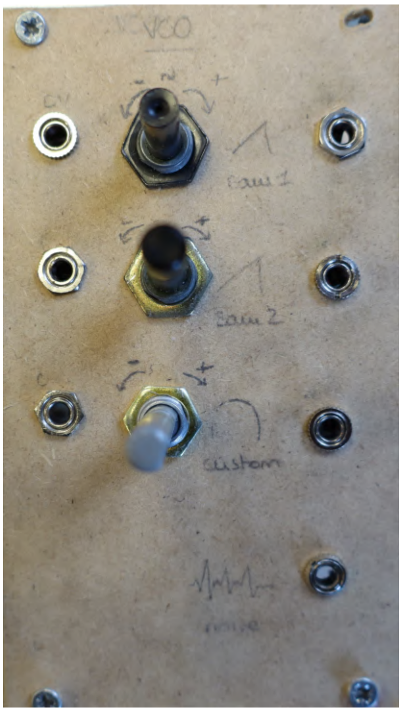


Fig. B3: VCO

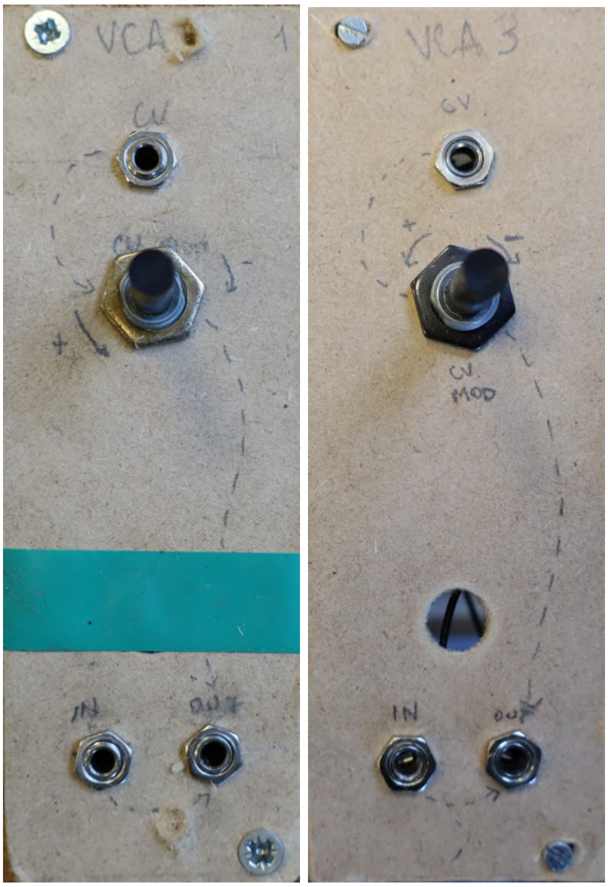


Fig. B4: VCAs

Fig. B5: VCFs

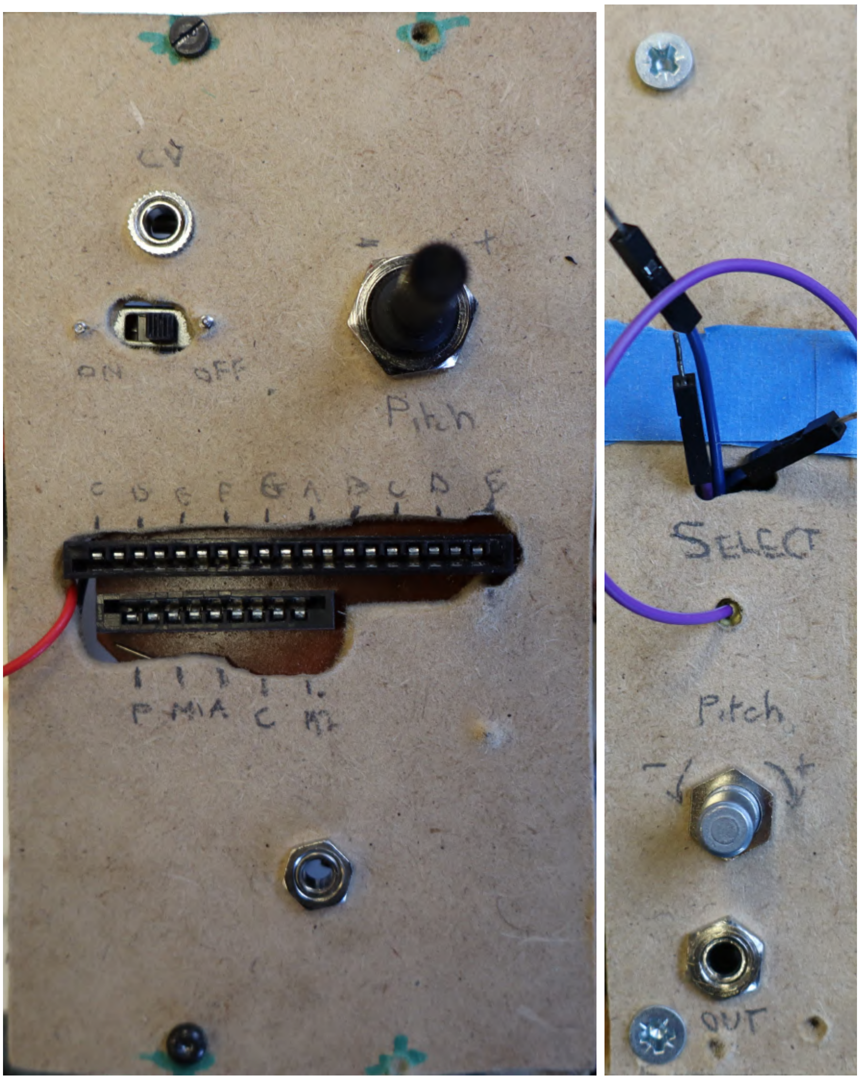


Fig. B6: Circuit bent toys

Appendix C Soundcloud playlist of sound exctrats

Following this link

Appendix D Electrical Schematics

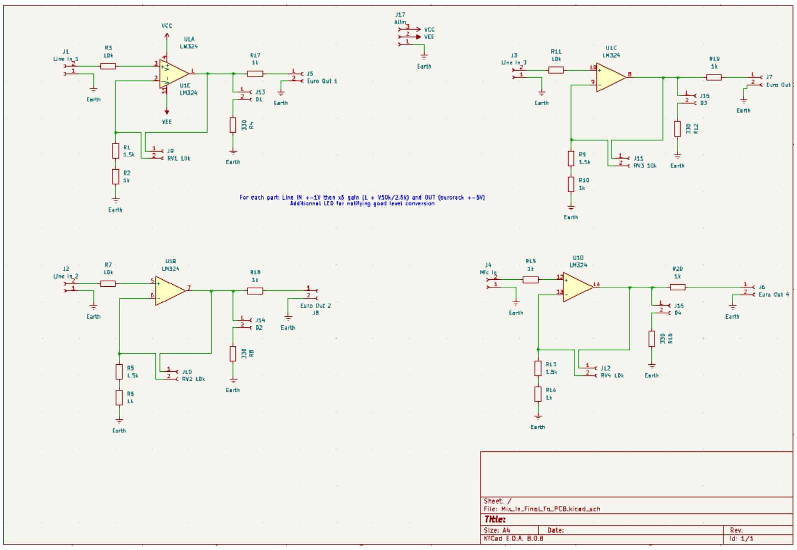


Fig. D7: MixIn schematic

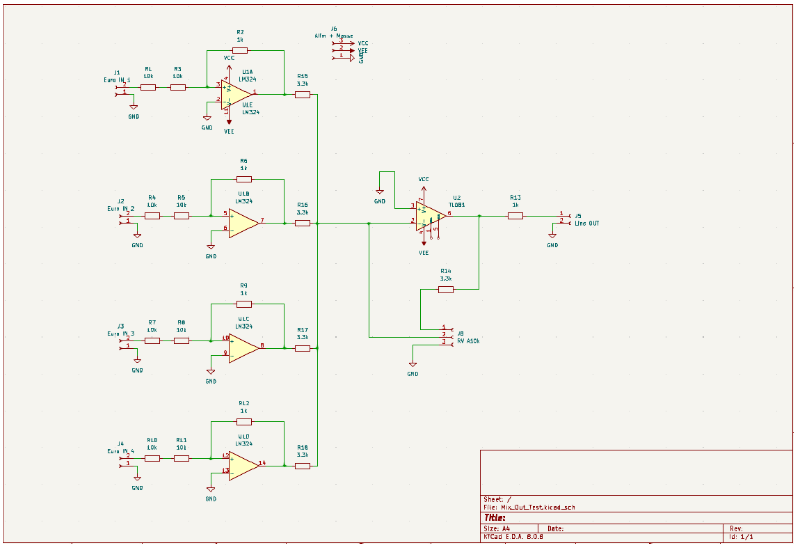


Fig. D8: MixOut schematic

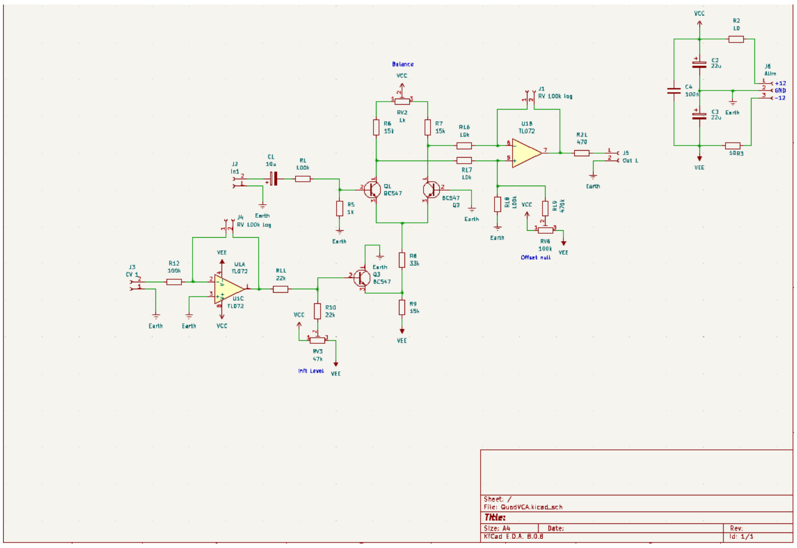


Fig. D9: VCA schematic

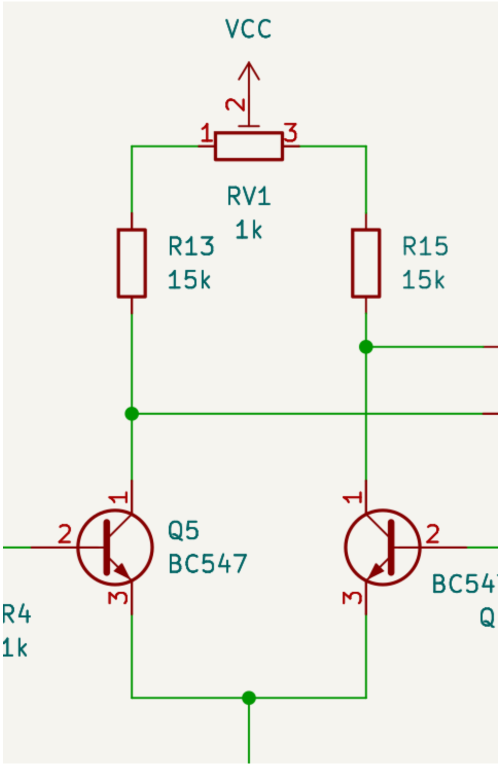


Fig. D10: VCA schematic - zoom on OTA

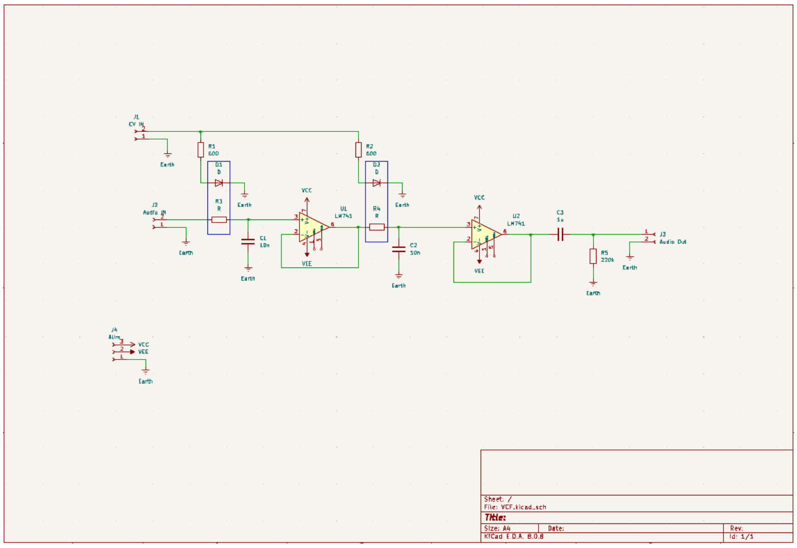


Fig. D11: VCF schematic

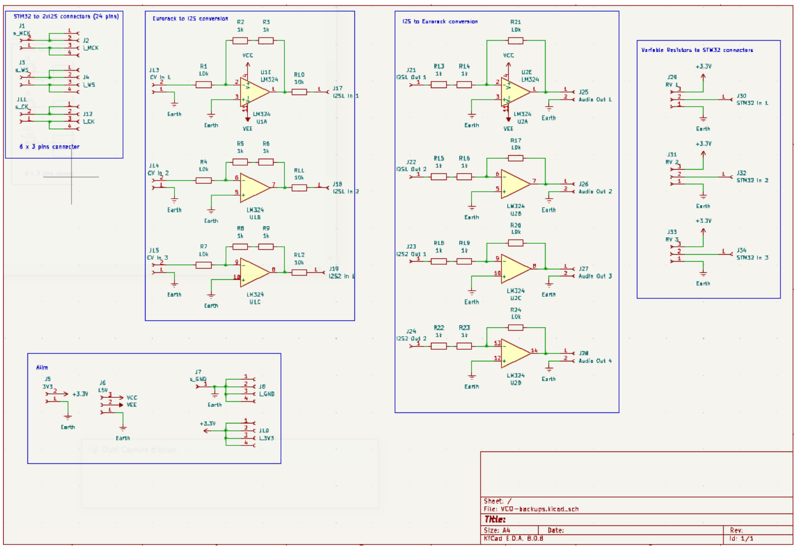


Fig. D12: VCO schematic

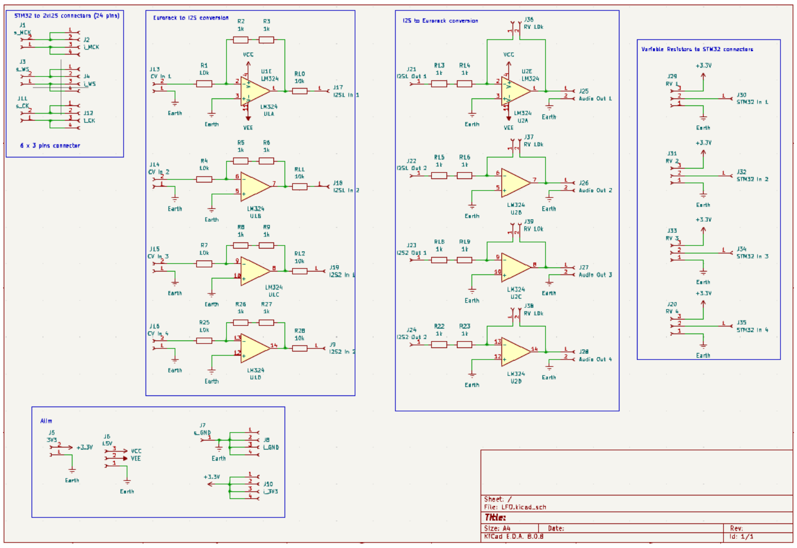


Fig. D13: LFO schematic

Appendix E Rastnests and PCB design

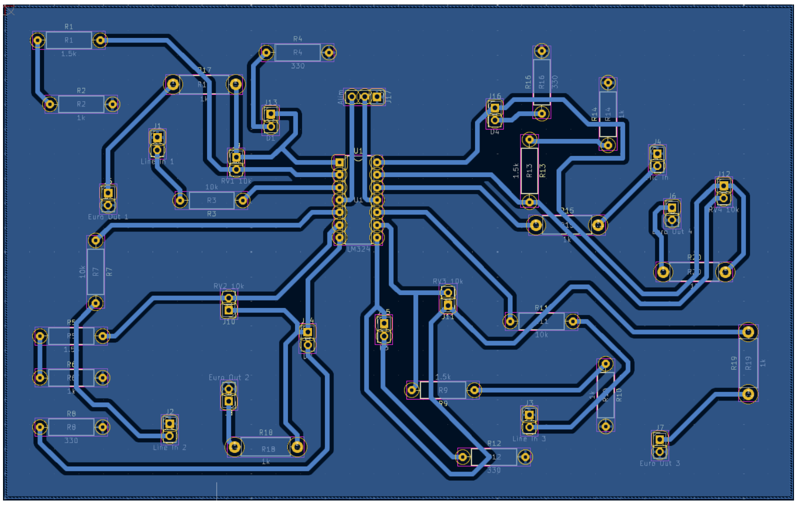


Fig. E14: MixIn PCB

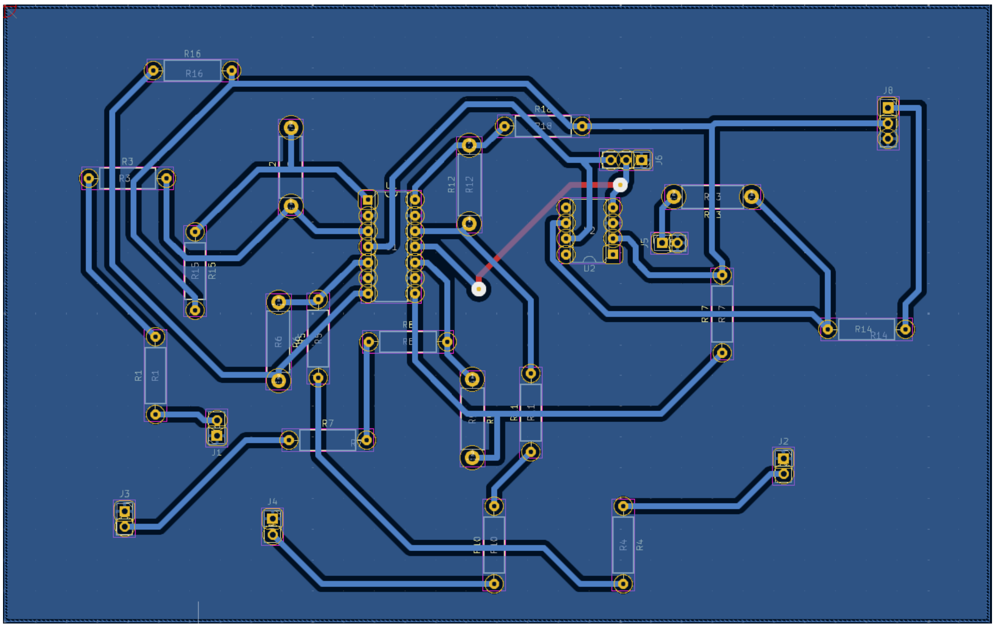


Fig. E15: MixOut PCB

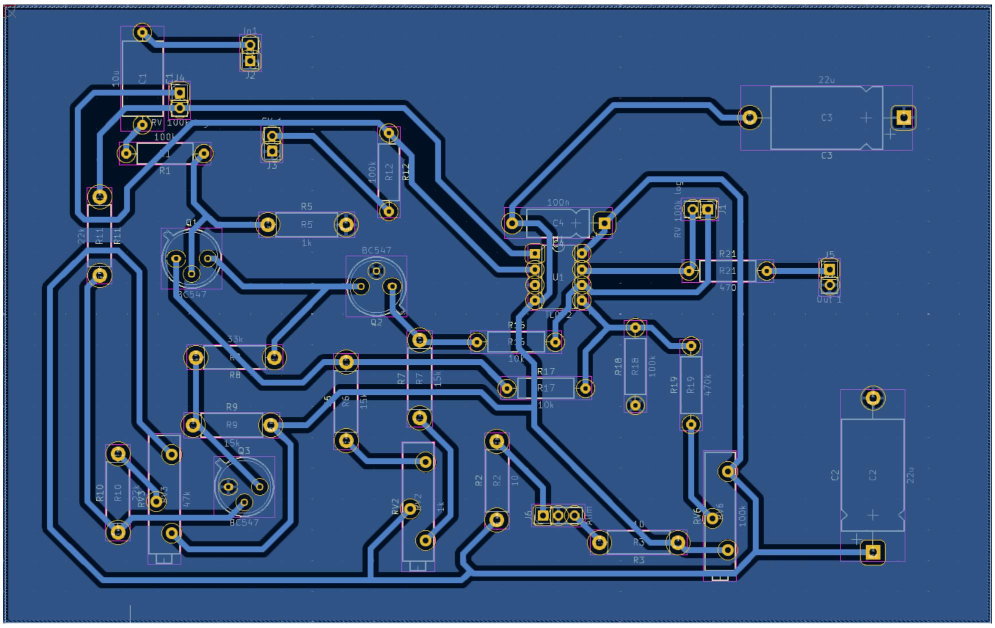


Fig. E16: VCA PCB

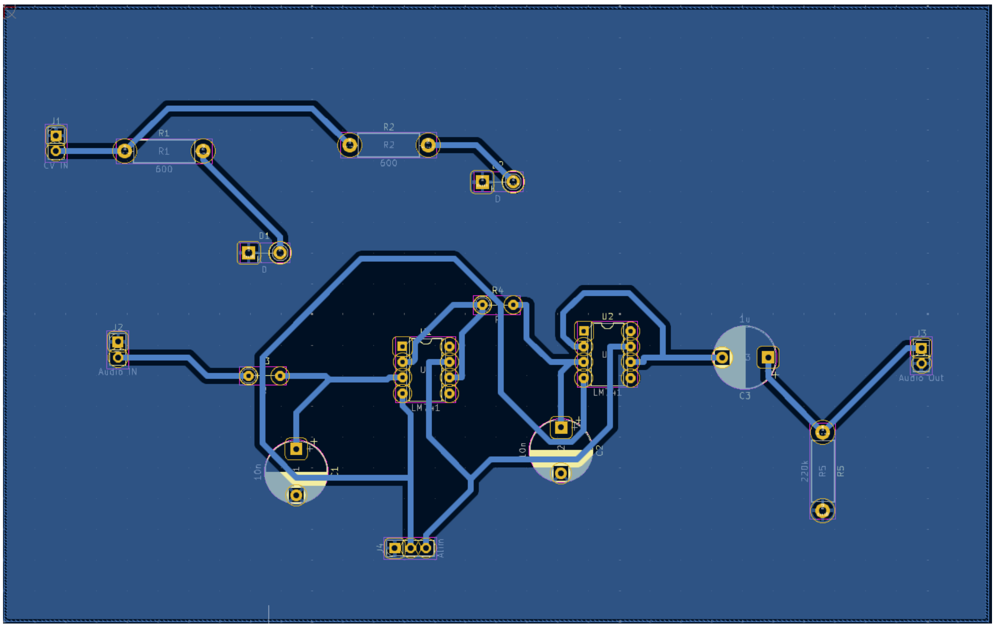


Fig. E17: VCF PCB

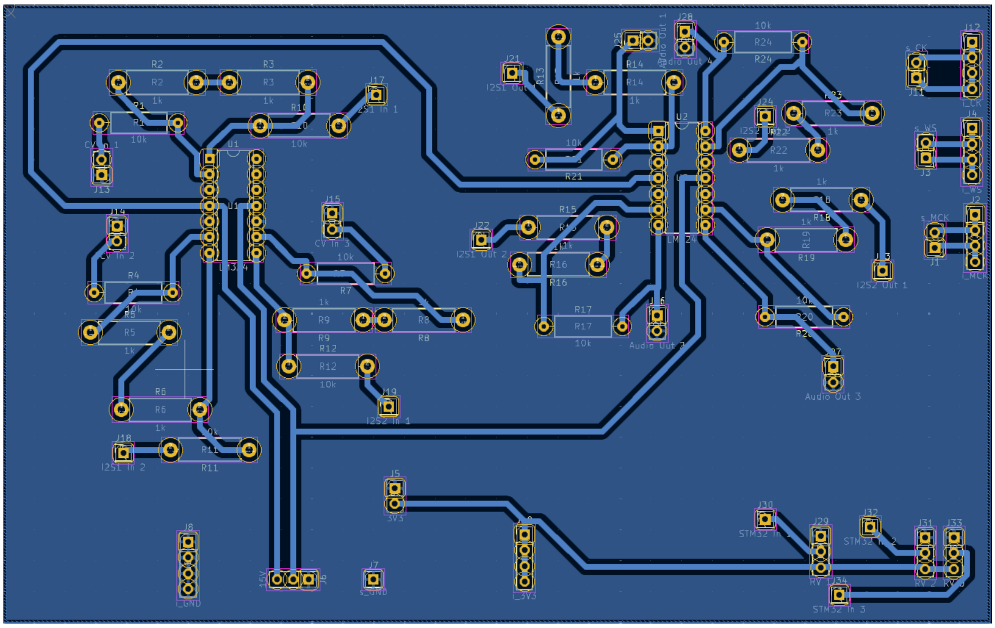


Fig. E18: VCO PCB

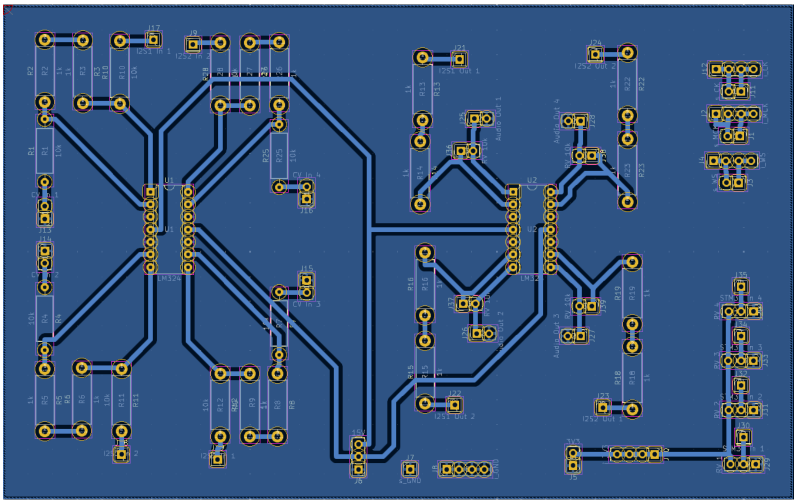


Fig. E19: LFO PCB

Appendix F Flatcam and CNC objects

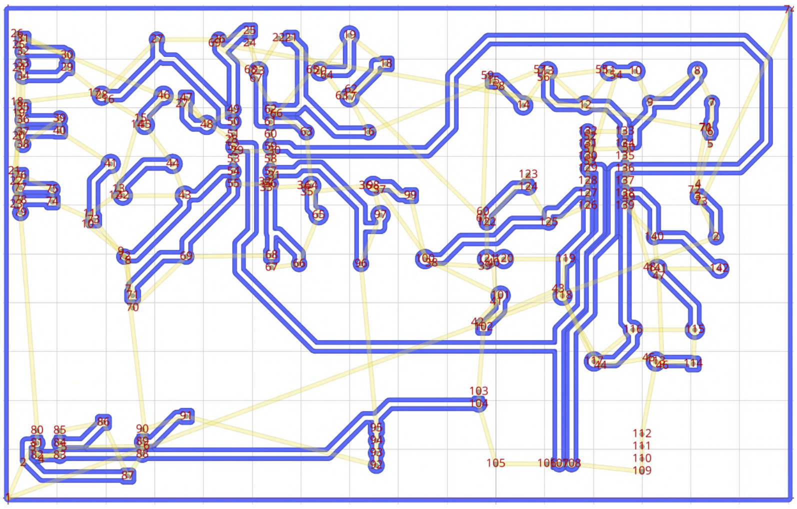


Fig. F20: VCO CNC

Appendix G   Code functions

```c
void HAL_I2SEx_TxRxHalfCpltCallback(I2S_HandleTypeDef *hi2s) //callback half I2S
{
    if (hi2s == &hi2s2)
    {
        inBufPtr = &adcData[0];
        outBufPtr = &dacData[0];
        h2 = true;
    }
    if (hi2s == &hi2s3)
    {
        inBufPtr2 = &adcData2[0];
        outBufPtr2 = &dacData2[0];
        h3 = true;
    }

    if (h2 == true && h3 == true)
    {
        HAL_ADC_Start_DMA(&hadc1, (uint16_t *)ADC_Value, 3);
        //processData();
        h2 = false;
        h3 = false;
    }
}
```

Fig. G21: Code ISR I2S

```c
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc) //callback full adc
{
    for (int i = 0; i < 3; i++)
    {
        ADC_Memory[i] = int12_to_float(ADC_Value[i]) * -1.0f; // voltage to float (-1, 1)
    }

    CV_processing();
    update(freq_CV_saw, freq_CV_saw2, freq_CV_custom);
    processData();
}
```

Fig. G22: Code ISR ADC

```c
float int12_to_float(int int12)
{
    return (float)(int12 - 2048.0f) / 2048.0f;
}
```

Fig. G23: Int12_to_float function

```c
void CV_processing()
{
    int unmux_saw = (((int)inBufPtr[0]<<16)|inBufPtr[1])>>8;
    int unmux_saw2 = (((int)inBufPtr[2]<<16)|inBufPtr[3])>>8;
    int unmux_custom = (((int)inBufPtr2[0]<<16)|inBufPtr2[1])>>8;

    float CV_saw = INT24_TO_FLOAT * unmux_saw * -1.0f; //CV input signal inversed because of the AOP montage
    float CV_saw2 = INT24_TO_FLOAT * unmux_saw2 * -1.0f;
    float CV_custom = INT24_TO_FLOAT * unmux_custom * -1.0f;

    CV_saw = clipper((CV_saw + ADC_Memory[0]), -1.0f, 1.0f) * 5.0f;// time 5 to get -+ 5V
    CV_saw2 = clipper((CV_saw2 + ADC_Memory[1]), -1.0f, 1.0f) * 5.0f;
    CV_custom = clipper((CV_custom + ADC_Memory[2]), -1.0f, 1.0f) * 5.0f;

    freq_CV_saw = Frequency_ref * powf(2, CV_saw); // frequency
    freq_CV_saw2 = Frequency_ref * powf(2, CV_saw2);
    freq_CV_custom = Frequency_ref * powf(2, CV_custom);
}
```

Fig. G24: Code CV_Processing

```c
void update(float input_saw, float input_saw2, float input_custom)
{
    target_saw2 = input_saw2;
    target_saw = input_saw;
    target_custom = input_custom;
}
```

Fig. G25: Code Update

```c
void smoothen()
{
    frequency_saw2 += (target_saw2 - frequency_saw2) * coeff;
    frequency_saw += (target_saw - frequency_saw) * coeff;
    frequency_custom += (target_custom - frequency_custom) * coeff;
}
```

Fig. G26: Code Smoothen

```c
void processData()
{
    float local_Out[4];

    for (uint8_t n = 0; n < (BUFFER_SIZE)-1; n += 4)
    {
        smoothen();

        T_saw = 2.0f / frequency_saw;
        T_saw2 = 2.0f / frequency_saw2;
        T_custom= 8.0f / frequency_custom;

        anti_aliasing(gen_saw(), gen_saw2(), gen_custom(), gen_noise(), local_Out);

        int leftOut_unmux_1 = (FLOAT_TO_INT24 * local_Out[0]);
        int rightOut_unmux_1 = (FLOAT_TO_INT24 * local_Out[1]);
        int leftOut_unmux_2 = (FLOAT_TO_INT24 * -local_Out[2]);
        int rightOut_unmux_2 = (FLOAT_TO_INT24 * local_Out[3]);

        outBufPtr[n] = (leftOut_unmux_1 >> 8) & 0xFFFF;
        outBufPtr[n + 1] = leftOut_unmux_1 & 0xFFFF;
        outBufPtr[n + 2] = (rightOut_unmux_1 >> 8) & 0xFFFF;
        outBufPtr[n + 3] = rightOut_unmux_1 & 0xFFFF;

        outBufPtr2[n] = (leftOut_unmux_2 >> 8) & 0xFFFF;
        outBufPtr2[n + 1] = leftOut_unmux_2 & 0xFFFF;
        outBufPtr2[n + 2] = (rightOut_unmux_2 >> 8) & 0xFFFF;
        outBufPtr2[n + 3] = rightOut_unmux_2 & 0xFFFF;
    }
}
```

Fig. G27: Code processData

```c
static inline float gen_noise()
{
    float sampleOut_noisef;
    sampleOut_noisef = ((float)rand() / RAND_MAX) * 2.0f - 1.0f;
    return sampleOut_noisef;
}
```

Fig. G28: Code gen_noise

```c
static inline float gen_saw()
{
    sampleOut_sawf += (2.0f * Ts / T_saw);
    if (sampleOut_sawf >= 1.0f) sampleOut_sawf -= 2.0f;
    return sampleOut_sawf;
}
```

Fig. G29: Code gen_saw

```c
static inline float gen_custom()
{
    float x = phase_custom * frequency_custom;
    phase_custom += Ts;
    if (phase_custom >= T_custom) phase_custom -= T_custom;
    return 8.0f * (x * (1.0f - x)) - 1.0f;;
}
```

Fig. G30: Code gen_custom

```c
void anti_aliasing(float saw, float saw2, float custom, float noise, float* buffer_Out)
{
    static float in[4][2];
    static float out[4][2];

    float a[2] = { 1.3629f, 0.5216f };
    float b[3] = { 0.7211f, 1.4423f, 0.7211f };
    float sampleIn[4] = { saw, saw2, custom, noise };

    for (int x = 0; x < 4; x++)
    {
        buffer_Out[x] = b[0] * sampleIn[x] + b[1] * in[x][0] + b[2] * in[x][1]
                                        - a[0] * out[x][0] - a[1] * out[x][1];
        in[x][1] = in[x][0];
        in[x][0] = sampleIn[x];
        out[x][1] = out[x][0];
        out[x][0] = buffer_Out[x];
    }
}
```

Fig. G31: Code anti aliasing

```c
void anti_aliasing(float saw, float saw2, float custom, float noise, float* sampleOut)
{
    static float in[4][2];
    static float out[4][4];

    float a[2] = { 1.3629f, 0.5216f };
    float b[3] = { 0.25, 0.5, 0.25 };

    float sampleIn[4] = { saw, saw2, custom, noise };
    float temp_out;

    for (int x = 0; x < 4; x++)
    {
        temp_out = b[0] * sampleIn[x] + b[1] * in[x][0] + b[2] * in[x][1] - a[0] * out[x][0] - a[1] * out[x][1];
        sampleOut[x] = b[0] * temp_out + b[1] * out[x][0] + b[2] * out[x][1] - a[0] * out[x][2] - a[1] * out[x][3];

        in[x][1] = in[x][0];
        in[x][0] = sampleIn[x];
        out[x][3] = out[x][2];
        out[x][2] = out[x][1];
        out[x][1] = out[x][0];
        out[x][0] = temp_out;
    }
}
```

Fig. G32: Code anti_aliasing 4th order

```c
static inline float gen_sqr()
{
    sampleOut_sqrf = (phase_sqr < T_sqr * 0.5f) ? attenuateur : -attenuateur;
    return sampleOut_sqrf;
}
```

Fig. G33: Code gen_sqr

```
void Write(float sampleInf, int sample_index)
{
    if (CV_record > THR) // on state
    {
        if (write_state == false) length = 0; // initializes size if writing starts
        if (length < SD_SIZE) length++; // size increments if possible
        if (length > max_length) max_length = length; // max size saved
        float_to_send_buffer(sampleInf, sample_index, send_buffer); // writes float in send_buffer
        write_index++;
        if (write_index >= SD_SIZE) write_index = 0; // write_index increments but it's a circular buffer
        write_state = true; //writing on
    }
    if (CV_record < THR) // off state
    {
        write_index = 0; /// index 0
        write_state = false; // writing off
    }
}
```

Fig. G34: Code Write

```
float Read()
{
    float sampleOutf = 0.0f; // returns 0 except if reading

    if (CV_start < THR && trig == true)
    {
        trig = false; // allows re-triggering of reading
    }
    if (CV_start > THR && max_length > 0 && trig == false)
    {
        trig = true; // reading on
        read = true; // reading authorized
        read_index = 0; //reading starts from beginning
        buffer_index = 0; // initializing buffer_index
    }
    if (read == true) // reading
    {
        sampleOutf = receive_buffer[read_index - RECEIVE_SIZE * buffer_index]; //
        read_index++;
        if (read_index >= ((buffer_index + 1) * RECEIVE_SIZE) - 1) buffer_index++;
        if (read_index >= max_length) // reading ended
        {
            trig_end = true; // allows CV generation
            read = false; // reading unauthorized
            trig = false; // reading off
            buffer_index = 0; // initializing buffer index
        }
    }
    return sampleOutf;
}
```

Fig. G35: Code Read

```
void float_to_send_buffer(float sampleIn, int sample_index, unsigned char* send_buffer)
{
    if (!send_buffer) return;
    memcpy(&send_buffer[sample_index], &sampleIn, sizeof(float));
}
```

Fig. G36: Code float_to_send_buffer

```c
void buffer_to_SD(unsigned char* send_buffer, FIL* fichier)
{
    if (fresult != FR_OK) return;
    fresult = f_write(fichier, send_buffer, RECEIVE_SIZE * sizeof(float), &bytesWritten);
}
```

Fig. G37: Code buffer_to_SD

```c
void SD_to_buffer(float* receive_buffer, FIL* fichier)
{
    if (!receive_buffer || fresult != FR_OK) return;
    int position = (buffer_index - 1) * sizeof(float) * RECEIVE_SIZE;
    if (position < 0) position = 0;
    fresult = f_lseek(fichier, position);
    fresult = f_read(fichier, receive_buffer, RECEIVE_SIZE * sizeof(float), &bytesRead);
}
```

Fig. G38: Code SD_to_buffer

```c
void processData()
{
    float AudioIn, AudioOut, CVOut;

    SD_to_buffer(receive_buffer, &file);

    for (int n = 0; n < BUFFER_SIZE - 1; n +=4)
    {
        int AudioIn_mux_i = (((int)inBufPtr[n] << 16) | inBufPtr[n + 1]) >> 8;
        AudioIn =  INT24_TO_FLOAT * AudioIn_mux_i;
        //Write(AudioIn, n);
        //AudioOut = Read();
        float_to_send_buffer(AudioIn, n, send_buffer);
        write_index++;
        if (write_index * sizeof(float) >= SD_SIZE) write_index = 0;
        AudioOut = receive_buffer[n/4];
        int AudioOut_unmux = (FLOAT_TO_INT24 * AudioOut);
        int AudioIn_unmux = (FLOAT_TO_INT24 * AudioIn);

        outBufPtr2[n] = (AudioIn_unmux >> 8) & 0xFFFF;
        outBufPtr2[n + 1] = AudioIn_unmux & 0xFFFF;
        outBufPtr2[n + 2] = (AudioOut_unmux >> 8) & 0xFFFF;
        outBufPtr2[n + 3] = AudioOut_unmux & 0xFFFF;
    }
    buffer_index +=1;
    if (buffer_index >= NBR_BUFFER_SD) buffer_index = 0; // incrementation of the buffer index

    fresult = f_lseek(&file, write_index * sizeof(float)); // sets cursor before writing
    buffer_to_SD(send_buffer, &file);
    fresult = f_sync(&file); //force writing on the SD card
}
```

Fig. G39: Code processData_looper

Appendix H Statement of intent

What Colors Do You See in the Sea?

Artistic Intention

Anamnesis - Return to the Past, Memory

At the age of 10, I had a board, wires, resistors, LEDs, and an Arduino in my hands. Since then, I've been attracted to any electronic gadget that comes my way, wondering how it works, what it's for, and how I could recreate it myself. I still remember making an LED Christmas ornament with various sequences and blinking patterns. I especially remember trying to do something, however clumsily, with a small buzzer whose function I didn't understand to try and make music. After several days of tinkering, I managed to create a third.

Thus, it's easy to imagine my enthusiasm when I discovered modular synthesizers. Having never had the opportunity to get my hands on one, I could only imagine what I could do with it.

One of the first things that fascinated me about these synthesizers was generative patches (like the Krell patch): an electronic circuit that plays on its own and is pleasant to listen to. However, it's an approach I quickly abandoned, as the compositional aspect was more limited and had a predictable, albeit unpredictable, structure.

Anticipation - Desire, Idea, Emerging Image

What I find particularly interesting in electronic composition is the focus, or even the central role, given to sound itself, its texture, and timbre, outside of any driving rhythm or melody. This, in my opinion, is what makes drone music particularly attractive, where the composer establishes a progressive interplay of the different parameters of the sound's timbre. This is the first aspect I wish to explore in this composition. The exploration of sound texture, even with a limited number of elements (sources, modulations).

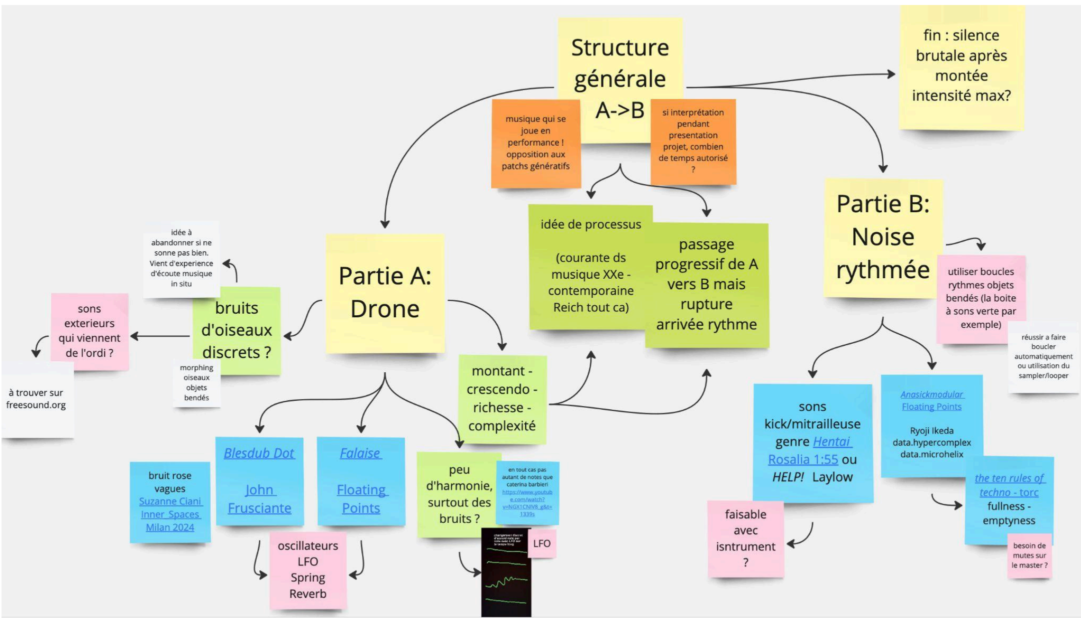
This contemplation of infinitesimal changes evokes a precise image for me: the way my paternal grandmother appreciated the sea in Brittany. Suffering from AMD, she could no longer really see it. She would ask: "What colors do you see in the sea?" You can contemplate it for hours. Every day. Endlessly. It never has the same color. It is full of nuances. Yet, it is always captivating. Moving. Emotional. This is the idea I would like to develop in the creation: composing a drone that is listened to as my grandmother contemplated the sea.

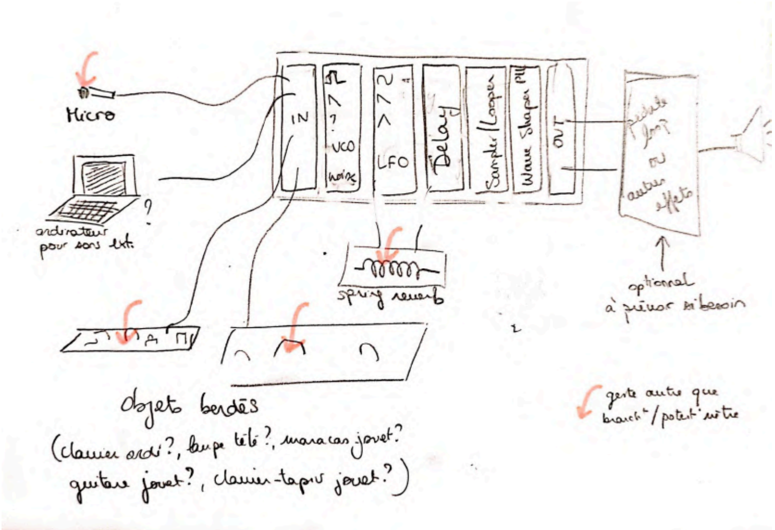
Intentions - Aesthetic, Artistic, etc...

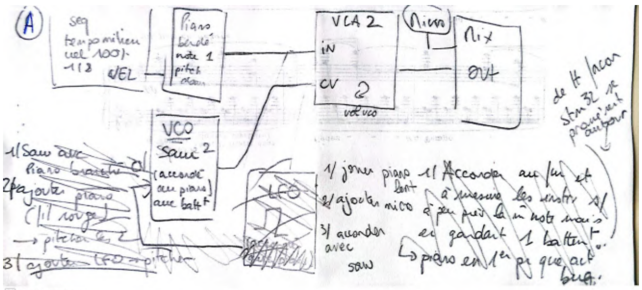
My artistic intention is divided into two main ideas. Firstly, to compose a piece that is played, interpreted, lived, and performed. In the movement of modular synthesis, this opposes the trend of generative patches (of which the Krell patch is one of the most famous examples). These patches, once set up, play music on their own, based on a pseudo-random pattern. The interpretation of the composition plays a crucial role in the work, especially since the instrument used has a certain degree of unpredictability. Indeed, the fact that the musician does not have absolute control over the textures, that the instrument "lives" and produces its own variations, is not an aspect to be avoided but rather an element from which the composition benefits. This loss of control puts the musician in a delicate position. The fact that one can hear the human striving to master their own machine breathes a living, even biological, dimension into the music, which can easily lack in this genre. The composition highlights the imperfections of both the human and the machine, thus offering a wavering way of creating art, tinkering with sounds and images, much like amateur science fiction films from the 1950s.

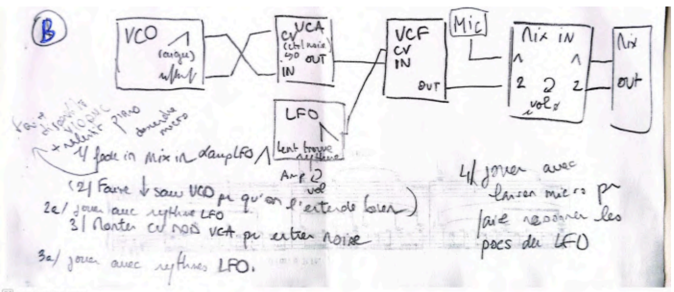
Secondly, to compose a drone that is listened to as my grandmother contemplated the sea, as explained in the previous section. To propose a different temporality from the daily routine. Taking the time to examine both the slow and broad evolutions (of the tides) and the smallest events (the foam bubbles)...

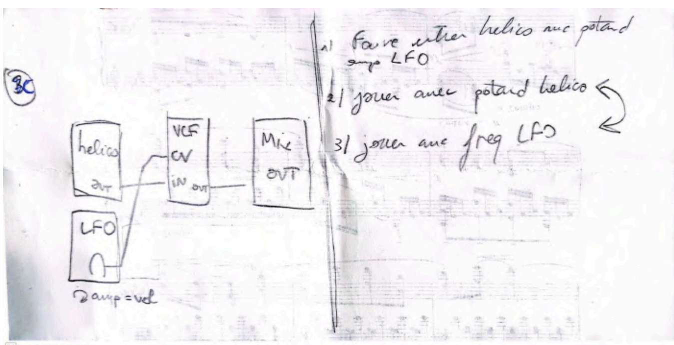
Planning (Sketches, Samples...)




Micro
IN
VCO
noise
LFO
Delay
Sampler / Looper
Wave shaper
OUT
pédale loop ou autres effets
ordinateur pour sons ext.
?
spring reverb
optionnel à prévoir si besoin
objets bendés
(clavier ordi?, lampe télé?, maracas jouet?
guitare jouet?, clavier-tapis jouet?)
geste autre que branch^t / potent^mètre


A
seq
Piano bendé
VCA 2
IN
CV
Mic
Mix
OUT
VCO
Saw
LFO
1/ jouer piano
2/ ajouter mico
3/ accorder avec saw


B
VCO
VCA
CV
OUT
IN
VCF
CV
IN
OUT
Mic
Mix IN
2
Mix
OUT
LFO
lent
Amp
1/ fade in Mix IN ↑ ampLFO
2/ Faire ↓ saw VCO pr qu'on l'entende bien
2a/ jouer avec rythme LFO
3/ Monter CV noise VCA pr entrer noise
3a/ jouer avec rythmes LFO.
4/ jouer avec
lancer micro pr
faire résonner les
pics du LFO


C
helico
OUT
VCF
CV
IN
OUT
Mix
OUT
LFO
amp = vol
1/ Faire entrer helico avec potard
amp LFO
2/ jouer avec potard helico
3/ jouer avec freq LFO

References

Methodology

Richard, M., Théberge, M.-P., & Majeau, C. (2017). LE DISPOSITIF DE CRÉATION/MÉDIATION AMALGAME : CROISER LES POSTURES ET TRANSGRESSER LES FRONTIÈRES. *Revue de recherches en littératie médiatique multimodale*, *6*, 1043752ar. https://doi.org/10.7202/1043752ar

Musical Inspirations

180 Fact. (2023, April 27). Patch notes: Caterina Barbieri [Video]. YouTube. https://www.youtube.com/watch?v=NGX1CNIV8_g

Floating Points. (2019, October 1). Floating Points - AnasickModular (Official Audio) [Video]. YouTube. https://www.youtube.com/watch?v=Md9gjJIqAxQ

Floating Points - Topic. (2019, October 17). Falaise [Video]. YouTube. https://www.youtube.com/watch?v=xJfpOgQcq9I

Release - Topic. (2023, February 2). Blesdub Dot [Video]. YouTube. https://www.youtube.com/watch?v=VC3mKDuFfdA

RosaliaVEVO. (2022, March 16). ROSALÍA - HENTAI (Official Video) [Video]. YouTube. https://www.youtube.com/watch?v=_6YCNd3ONUU

Ryoji Ikeda - Topic. (2023, May 14). data.microhelix [Video]. YouTube. https://www.youtube.com/watch?v=83gDTvwgm-E

San Fedele Musica. (2025, January 17). Suzanne Ciani Inner_Spaces Milan 2024 [Video]. YouTube. https://www.youtube.com/watch?v=kZpYkjook_I

Appendix I Miscellaneous

	Piano note	Melodies	Animal	SFX	Functions
C	Note C	/	/	/	/
D	Note D	Piano	Elephant	Scream	Vol +
E	Note E	Violon	Cow	Grunt 1	Vol -
F	Note F	Accordeon	Sheep	Eagle	Melody
G	Note G	Trumpet	Horse	Grunt 2	music random
A	Note A	Xylophone	Rooster	Grunt 3	music random
B	Note B	Flutes	Chicken	Kick	sound piano on
C	Note C	Synth	Duck	Drift	bip
D	Note D	Ocarina	Witch	Horn	on/off
E	Note E	/	/	/	/

Fig. I40: Samples available in chip of the bent piano module

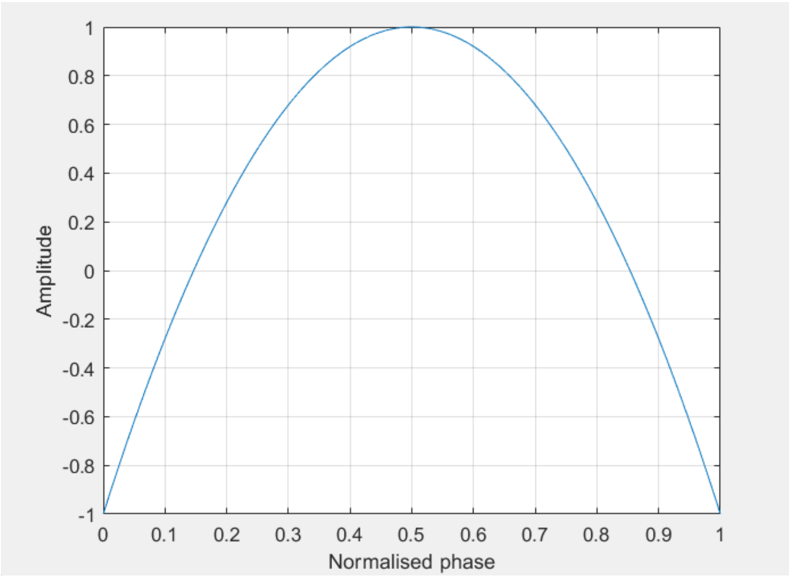


Fig. I41: Custom Waveform

Fig. I42: CNC Machine

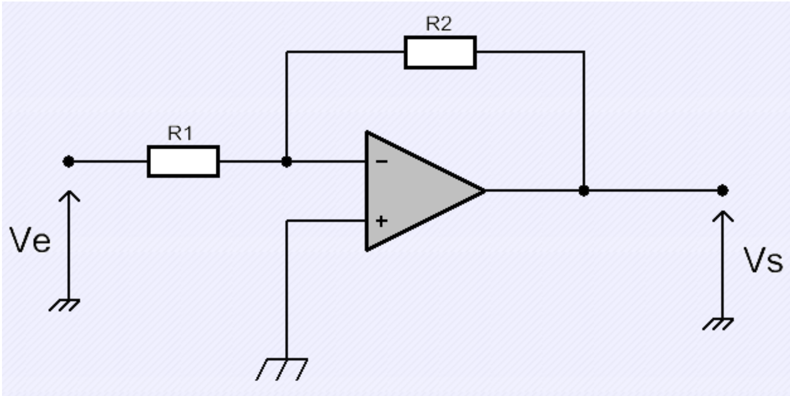


Fig. I43: Inverting gain circuit

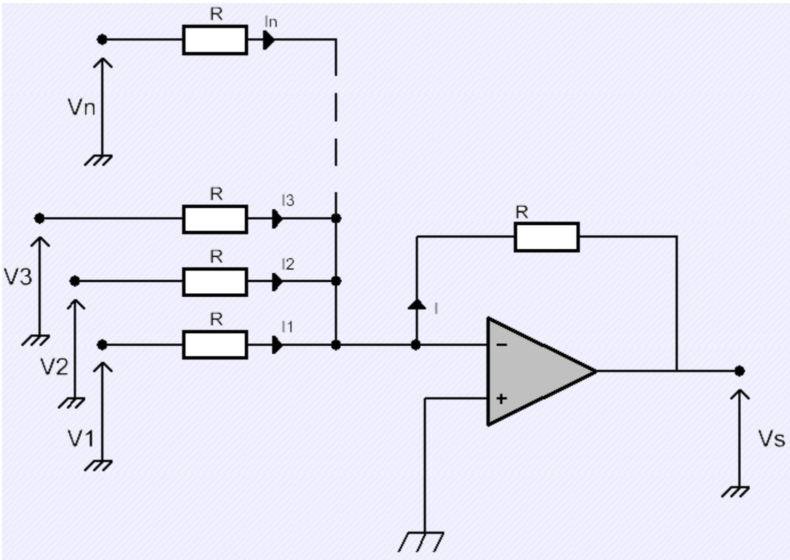


Fig. I44: Inverting summing circuit

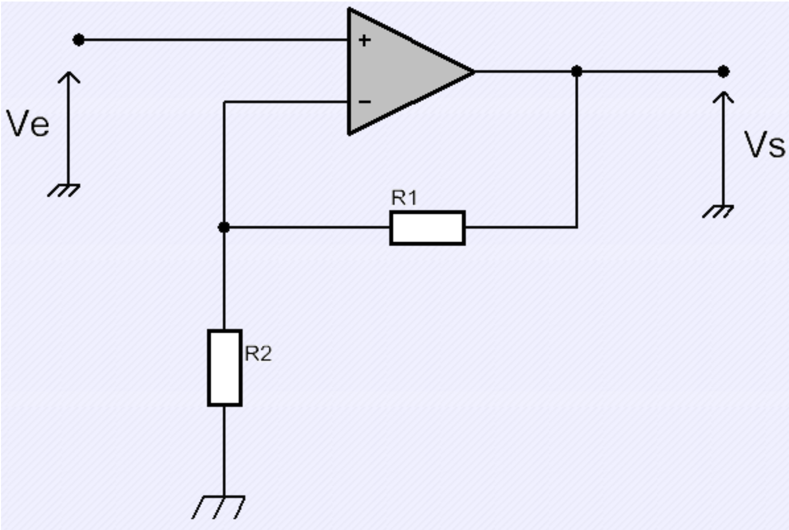


Fig. I45: Non-Inverting summing circuit