POLITECNICO
MILANO 1863

CMLS PROJECT

Prof. Fabio Antonacci

A.Y. 2023/2024

Andrea Arosio
Codice Persona: 10741332

Marco Roncoroni
Codice Persona: 10622517

Stefano Morano
Codice Persona: 10737463

Enrico Torres
Codice Persona: 10712642

Contents

1 Introduction

The aim of our project is to create a complete synthesizer that could be used for live performance. The main system is characterized by three oscillators and for each of them the user can select the type of waveform between: sine, square and saw.

For each waveform there are several knobs that modify different parameters: the waveform (*octave, phase, tuning*), the envelope (*attack, release, sustain*) and there is also an LFO that can be controlled adjusting the frequency and depth. The user could decide to use only one waveform at a time or all together: the sound is processed and it is possible to control a low pass filter, by choosing the frequency and the resonance, the pan and the level. Another tool that we implemented is the sequencer and it can be activated at any time.

We used SuperCollider to generate and control all the sounds, Juce to create a GUI and to define the knobs and their behaviour. Processing is implemented to visualize the animation of the three waves. We used also Arduino in order to control the interaction of the parameters by using analog knobs. The interaction between the different programs is:

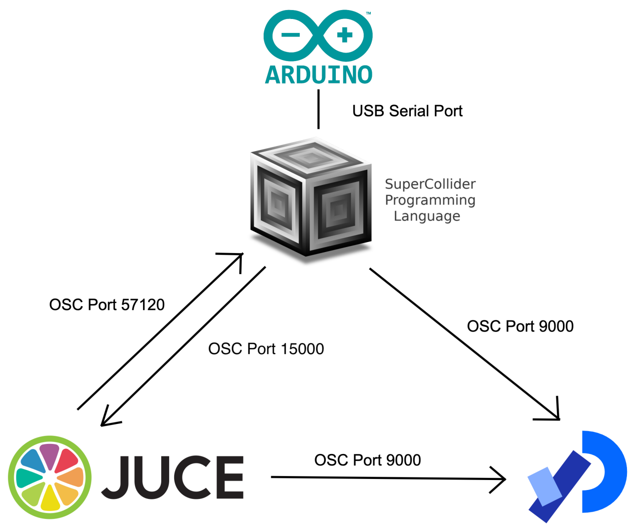

Figure 1: Connection and interaction between all the programs used

2 Juce

In Juce we have two main parts: each of them is composed by a header, where we declare our classes, methods and useful functions, and an implementation file, where we describe the behaviour of each methods and functions. In particular we have:

- an **editor**, that refers to the graphical user interface (GUI) component of an audio plugin and it is associated with the *AudioProcessorEditor* class where we defined all the buttons, sliders and knobs
- a **processor** that involves handling the audio data, applying effects, and managing plugin parameters and it is associated with the *AudioProcessor* class

We initialised all the graphical elements and positioned them in a suitable way, we have to implement a listener function that recognizes which element was modified and calls back the equivalent and unique function described in the processor.

In depth, we define the user interface where we initialize the various UI components. They are organized into sets of knobs for different sound parameters like phase, detune, attack, sustain, release, frequency, and depth. Each set corresponds to a different part of the synthesizer's functionality. The sliders are styled as rotary knobs and have their value ranges and text box styles set accordingly. The colors of the sliders' thumbs are set to orange, and listeners are added to handle user interactions.

Then we have a toggleable button to activate low-frequency oscillators and changing octaves, with the octave buttons updating a label to display the current octave. There are also controls for the filter section with cutoff and resonance parameters, general controls for pan and level, and sequencer controls like record, play/pause, speed, and time.

The resized method sets the bounds for each UI component, ensuring they are positioned correctly within the plugin editor window. The *sliderValueChanged* and *buttonClicked* methods contain the logic to update the audio processor's

state based on user interactions with the UI components.
While in the *OrangeJuceAudioProcessor* class is defined with several member functions that handle various aspects of the audio processing lifecycle. The constructor initializes the plugin with input and output buses, while the destructor cleans up any resources.
The *processBlock* method is where the main audio processing occurs. The loop also processes MIDI messages, handling note on and note off events, and sending frequency and amplitude data to a specified IP address and port using the OSC protocol. There are also methods to set various parameters such as phase, attack, release, sustain, detune, and frequency, which affect the sound synthesis. All the informations collected in input by the MIDI is sent to Supercollider (section 3).

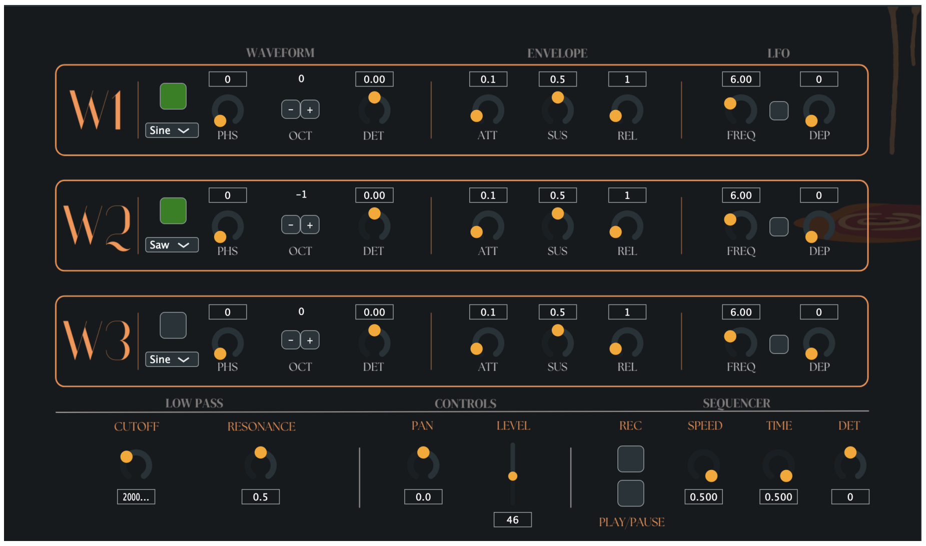

Figure 2: Interface of the VST created in Juce

3 SuperCollider

In SuperCollider we have more control in creating and modifying sounds. We have two main functions:

- a **GenerateWave** that creates the soundwave from the parameters that receives and modifies it if the lfo is activated with its parameters
- a **MiXSignal** is a function that collects all the three oscillators and returns as output the sum of the ones selected by the user

We initialize an audio server, set the sample rate, and specify the output device. Then we define a function to generate waveforms based on various parameters like frequency, wave type, detune, phase, octave, and low-frequency oscillation (LFO). It allows for dynamic waveform generation by linking the LFO to the waveform's frequency, creating modulated sounds.

The script also includes a *SynthDef* (a blueprint for a sound) named 'mySynth,' which uses the waveform generation function to create three oscillators with individual envelopes and parameters. These oscillators can be mixed together in different combinations using a mixing function, and the resulting signal is processed through an envelope generator, panning, and a low-pass filter to shape the final output.

Additionally, the script employs Open Sound Control (OSC) definitions to receive real-time control messages for various parameters like frequency, amplitude, attack, release, sustain, wave type, detune, button status, octave, panning, and sequencer settings. This allows for interactive control of the synth's parameters, enabling live performance or responsive sound installations.

The serial input, that arrives from Arduino (section 5), is converted into a string of ASCII. At this point the function ∼*readSerialValue* is called to evaluate the extracted value and it modifies the associated parameter by sending, at the same time, a feedback OSC to Juce.

Using the button *rec, play* and *pause* it is possible to record a sequence of notes and play them in loop. By modifying a few parameters it is possible to modify how the sequence is played changing *e.g.* the speed of execution of the loop.

4 Processing

The Processing code is used to visualize the three different waveforms at the same time.
It integrates the *oscP5* library to handle OSC (Open Sound Control) messages, allowing for real-time audio control. The sketch defines variables for frequency, amplitude, and phase of the waveforms, as well as visual elements like colors and fonts. The *setup()* function initializes the OSC communication and sets up the visual environment, while the *draw()* function continuously renders the waveforms and interface elements on the screen.
The sketch supports three types of waveforms: sinusoidal, square, and sawtooth, which are drawn based on the $waveform\_type$ array. Users can interact with the sketch to start and stop the sequencer, change waveform types, and adjust the frequency and phase of the waveforms through OSC messages. The *updateSequencer()* function manages the timing and playback of a sequence of notes, and the *oscEvent()* function processes incoming OSC messages to update the sketch's parameters accordingly.

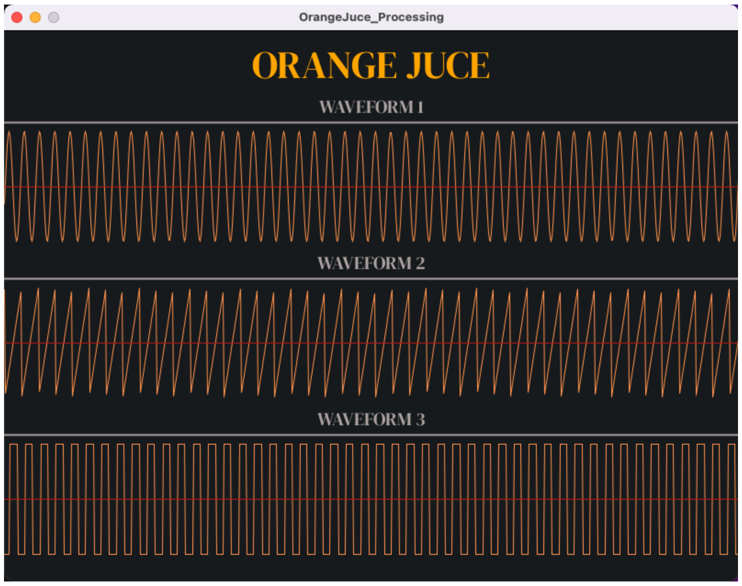

Figure 3: Processing interface with three waveforms

5 Arduino

The Arduino code is used to control a set of knobs that modify some parameters of the sound processed by SuperCollider (section 3) and control also some LEDs. It begins with the declaration of arrays to store the pin numbers for the knobs and LEDs, as well as an array to keep track of the last read values from the knobs. In the *setup()* function, the serial communication is initiated, and the LED pins are set as outputs with the first LED turned on. The initial values of the knobs are read and stored.

In the *loop()* function, which runs continuously, the code reads the value from each knob and compares it to the last stored value. If there is a significant change, the new value is stored, and depending on the knob's position, different act ions are taken. For the eighth knob, the value is inverted and mapped to a range of 0 to 100. Depending on this value, one of the three LEDs is turned on, and a corresponding value is sent via serial communication, indicating a change in the knob's position. For the other knobs, if there is a change, the value is adjusted to fit within a 0 to 100 range and sent via serial communication. The 'a' character is sent after each value as a delimiter.

The output is a series of alphanumeric characters that assume a numerical value when a parameter is modified. The value is obtained by using:

$$(potentiometer\_number) * 100 + (potentiometer\_value)$$

The last potentiometer allows as to select the chosen waveform and a LED shows which wave is selected.

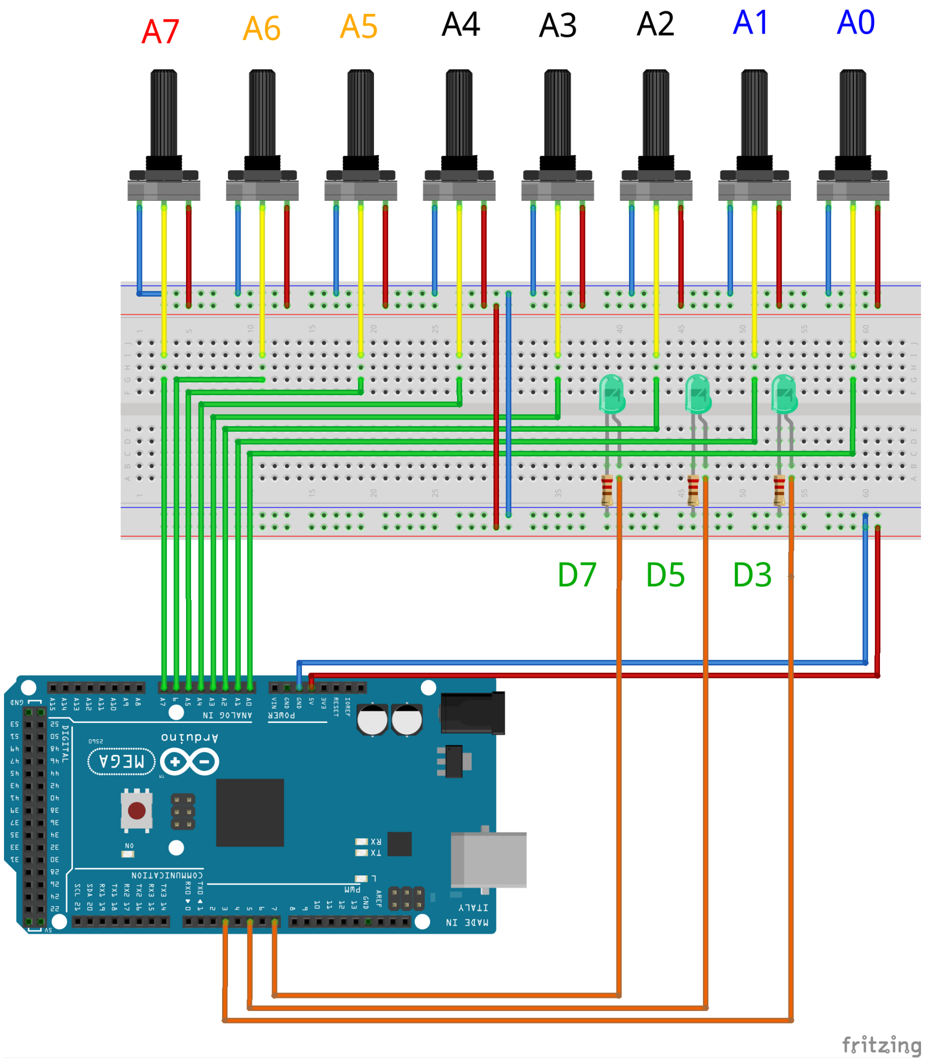

Figure 4: Arduino scheme