POLITECNICO MILANO 1863

CMLS 2022/23

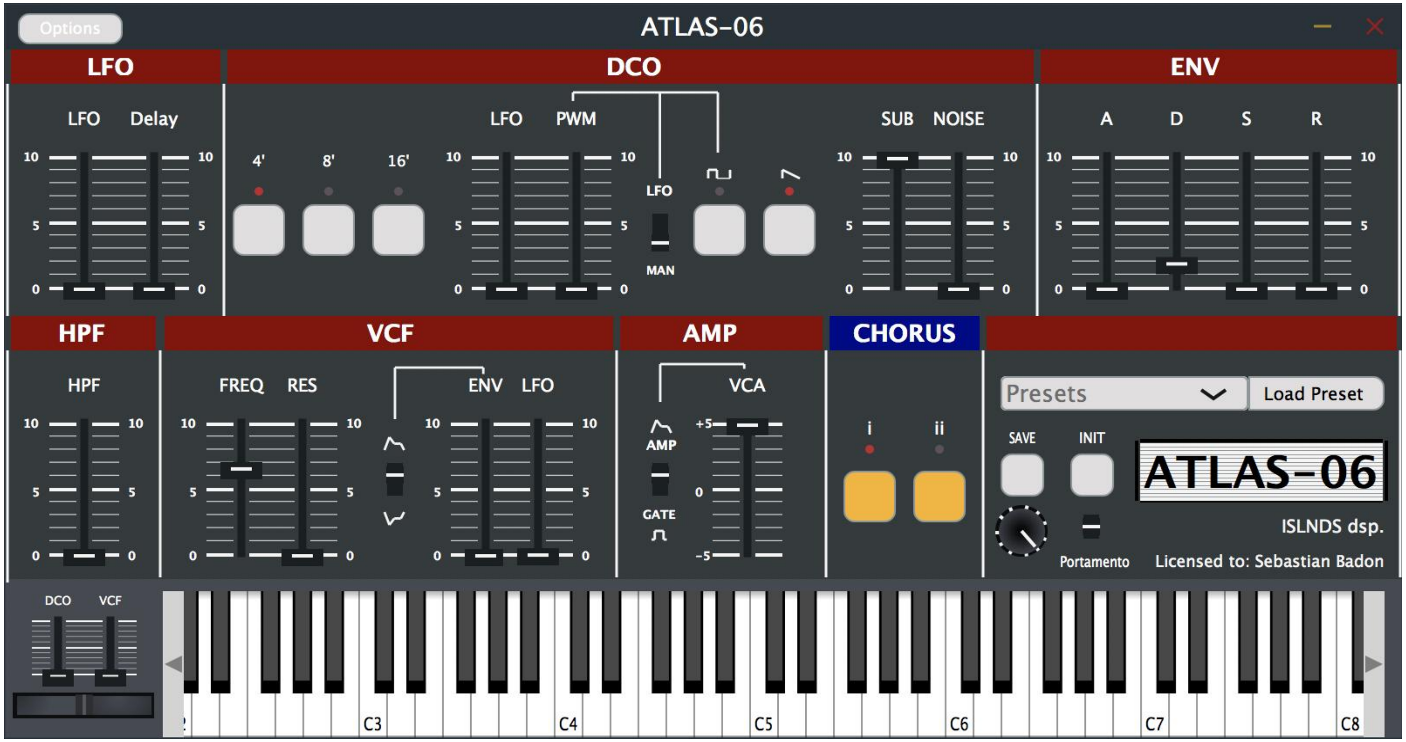
Analysis of ATLAS-06-Synthesizer

Group 12: 0.5 Musicians

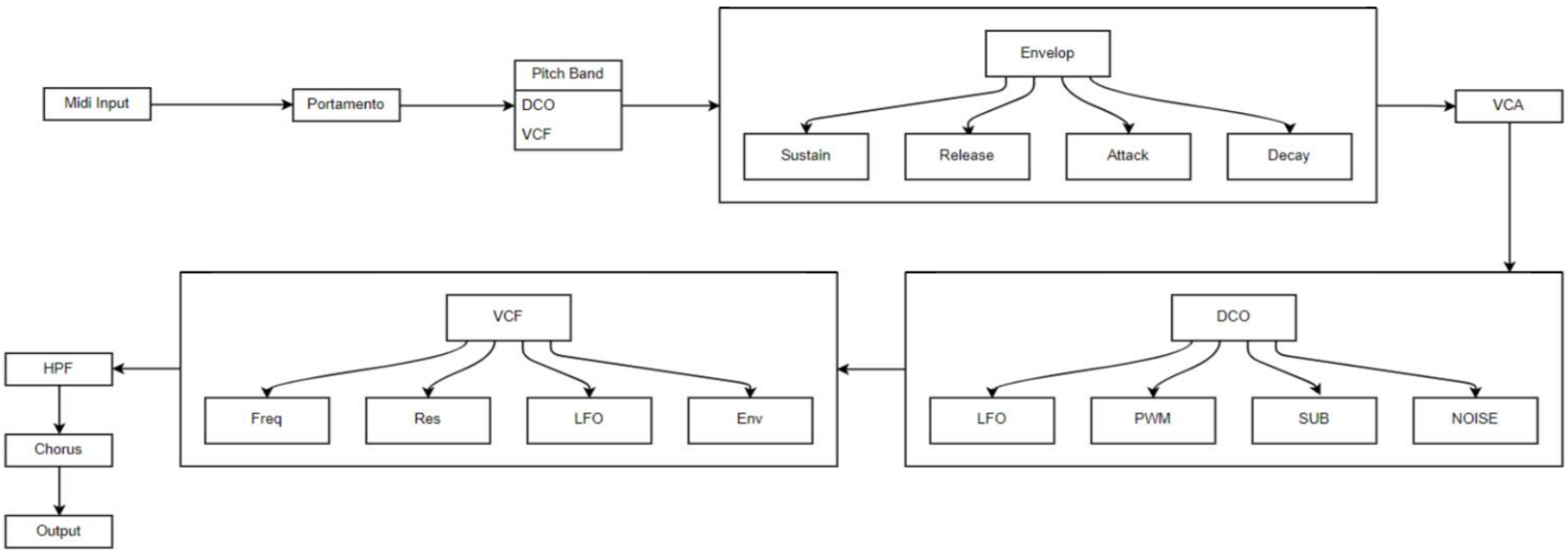
Stepanov Maksim
Jian Zhou
Baichen Li
Yueqian Wu
Wenjia Luo

Introduction of ATLAS-06-Synthesizer

This plug-in is designed to emulate a typical synthesizer inspired by 80s synths. It offers a graphical user interface that allows the users to generate different sounds using different terms.



Project Structure



The structure can be divided into seven modules: **LFO, DCO, ENV, HPF, VCA, VCF and Chorus.** The block diagram illustrates the audio chain of the whole plug-in.

Sound Production Logic

- **The sound synthesis functionality primarily relies on the ‘Maximilian’ library for implementation.**
    - ➢ An open-source audio synthesis and digital signal processing library written in C++

- **A class ‘SynthVoice’ is defined to implement the sound production logic. It extends the ‘SynthesiserVoice’ class from the JUCE framework .**
    - ➢ A set of variables and methods to manipulate the sound properties of a synthesiser voice
    - ➢ Using the maxiOsc and maxiEnv classes from the Maximilian library for creating oscillators and envelopes.

Modules-LFO

- **Working Principle**

The main principle behind LFOs is to create a repeating waveform that can be used to modulate other audio signals, adding a sense of movement and animation to the sound.

- **Implementation**

The ***LFO*** class:

➢ ***'Slider'*** objects for controlling the LFO rate and delay parameters

➢ ***'AudioProcessorValueTreeState::SliderAttachment'*** objects for linking the values of the sliders to a value tree

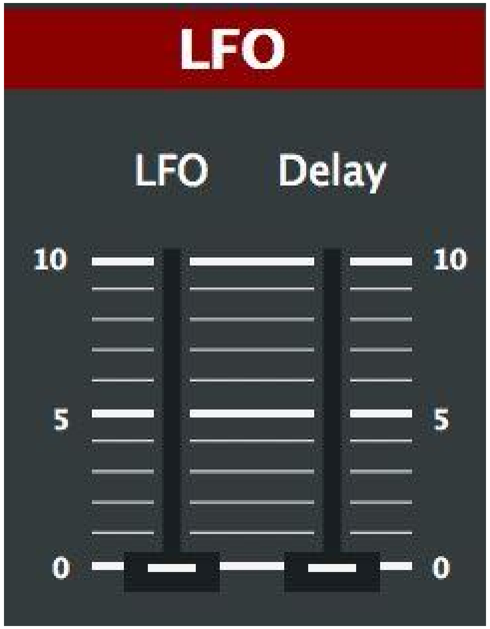
The ***SynthVoice*** class:

➢ A maxiOsc object from the 'Maximilian' library, named 'LFO'

```
double maxiOsc::sinewave(double frequency) {
    //This is a sinewave oscillator
    output=sin (phase*(TWOPI));
    if ( phase >= 1.0 ) phase -= 1.0;
    phase += (1./(maxiSettings::sampleRate/(frequency)));
    return(output);
```

Modules-LFO

- **GUI**

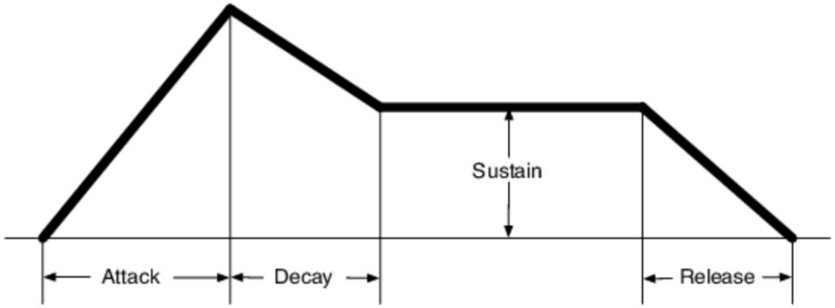


- ***'lfoRate'*** controls the frequency or rate of the LFO waveform, which determines how fast it oscillates or modulates the audio signal.

- ***'lfoDelay'*** controls the amount of time before the LFO modulation starts, which can be used to create a gradual or phased-in effect. In this case.

Modules-ENV

- **Working Principle**
  - **Attack:** the time taken for the volume to reach its maximum level after the key is pressed or the gate signal is received.
  - **Decay:** the time taken for the volume to drop from its maximum level to the level set by the Sustain parameter.
  - **Sustain:** the level at which the volume remains as long as the key or gate signal is held down.
  - **Release:** the time taken for the volume to return to zero after the key is released or the gate signal ends.



Modules-ENV

- **Implementation**

The ***ENV*** class:

➢ ***‘Slider’*** objects for controlling the attack, decay, sustain, and release parameters of the envelope***‘AudioProcessorValueTreeState::SliderAttachment’*** objects for linking the values of the sliders to a value tree

The ***SynthVoice*** class:

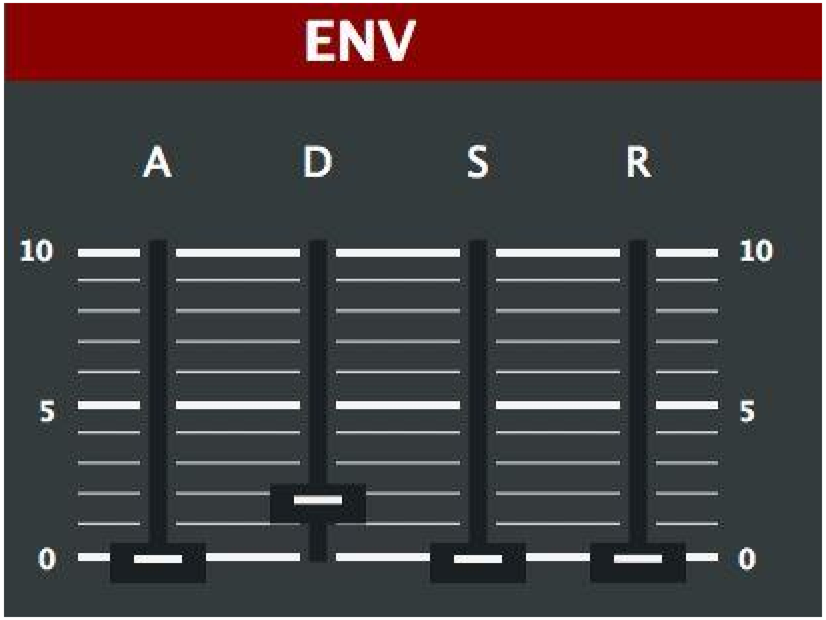
➢ The ***‘maxiEnv ’*** objects ***‘lfoEnv’*** are used to create instances of the envelope generator

```
double maxiEnv::adsr(double input, double attack, double decay, double sustain, double release, long holdtime, int trigger) { ... }
double maxiEnv::adsr(double input, int trigger) { ... }
```

➢ The ***‘getEnvelopeParams’*** method sets the parameters

Modules-ENV

- **GUI**



- ‘**AttackSlider**’ is the graphical slider that allows the user to adjust the attack value.
- ‘**DecaySlider**’ is a graphical slider that allows the user to adjust the decay value .
- ‘**SustainSlider**’ is the graphical slider that allows the user to adjust the sustain value.
- ‘**ReleaseSlider**’ is the graphical slider that allows the user to adjust the release value .

Modules-HPF

- **Working Principle**

  The HPF works by allowing signals above a certain cutoff frequency to pass through while attenuating signals below this frequency.

- **Implementation**

  The ***HPF*** class:

  - ***‘Slider’*** objects for controlling the cutoff frequency
  - ***‘AudioProcessorValueTreeState::SliderAttachment’*** objects for linking the values of the sliders to a value tree

  The ***SynthVoice*** class:

  - The ***‘maxiFilter’*** object named ***‘filter2’*** is used to implement the HPF

```
double maxiFilter::hipass(double input, double cutoff) {
    output=input-(outputs[0] + cutoff*(input-outputs[0]));
    outputs[0]=output;
    return(output);
}
```

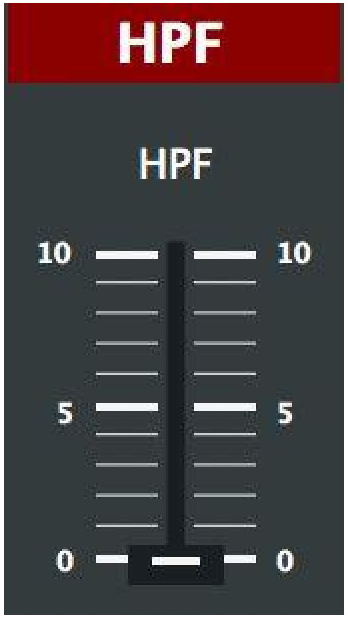
$$output = input\text{-}ouputs[0]\text{-}cutoff*(input\text{-}outputs[0]);$$

$$output = (1\text{-}cutoff)*(input\text{-}outputs[0]);$$

  - The ***‘setHpfSetting’*** sets the parameter

Modules-HPF

- **GUI**



- ‘HpfSlider’ is the graphical slider that allows the user to adjust the cutoff amount.

Modules-DCO

- **Working Principle**

DCO (Digitally Controlled Oscillator) is a type of oscillator that uses digital circuitry. They were designed to replace earlier VCO (Voltage Controlled Oscillator) designs. VCO typically consists of a core oscillator circuit that generates a waveform, such as a sine wave, triangle wave, etc. The frequency of the waveform is determined by a control voltage in the oscillator circuit.

- **Implementation**

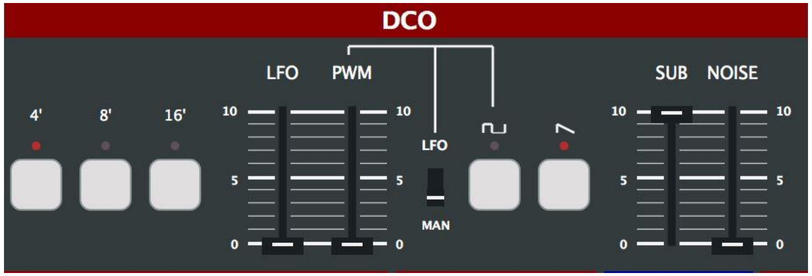
```
double maxiFilter::hipass(double input, double cutoff) {
    output=input-(outputs[0] + cutoff*(input-outputs[0]));
    outputs[0]=output;
    return(output);
}
```

➢ noiseValue: use rand() to generate a number between 0 to 1 and map it between -1 to 1, and multiply it by the value read by the noise slider.

➢ getSawOsc() and getSquareOsc() return the Sawtooth generator and square wave generator from the external library.

Modules-DCO

- **GUI**



- 4’ 8’ 16’ These buttons will divide the notes’ pitch frequency by 1, 2, and 4 respectively.
- LFO changes the amount of the modulation.
- PWMcontrols the amount of Pulse Width Modulation (PWM) applied to the DCO's pulse wave. By modulating the pulse width, a wide range of timbres and effects can be created.
- The next two buttons are the switches of the square and saw oscillator.
- SUB adds a square oscillator with a pitch 2 octaves lower.
- Noise adds white noise.

Modules-VCF

- **Working Principle**

VCF stands for "Voltage-Controlled Filter". VCF is an audio signal processor typically used for changing the richness and color of timbre and creating a variety of sound effects. It can manipulate the cutoff frequency and harmonic content of a filter based on a control voltage.

- **Implementation**

The ***std::unique_ptr<AudioProcessorValueTreeState::SliderAttachment>*** variable declaration creates 2 new SliderAttachment objects.

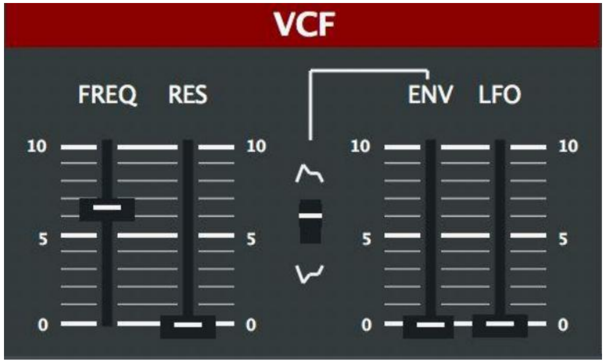
For Cut-off Frequency, the range of the resonanceVal will be between 30 - 4000 Hz, and 4000 Hz is the default value, which is calculated in the following code.

```
cutoffValue += getLfoValue()*lfoFilterEnvelopeSetting+vcfSliderPitchBendSetting*pitchBendPosition;
```

For the Resonance, the range of resonanceVal will be between 1-20, and 1 is the default value.

Modules-VCF

- **GUI**



The SliderAttachment object created in the function is assigned to the filterVal pointer for Cutoff-Frequency and resonanceVal pointer for Resonance, which represents a slider GUI component controlling the resonance parameter of the VCF. The GUI element is created in another part of the code and passed as a reference to the SliderAttachment object.

Modules-VCA

- **Working Principle**

An amplifier (AMP) is usually used to achieve amplification, which is the process of increasing the amplitude or strength of a signal.

- **Implementation**

'VCA' class constructor initializes includes a slider to control the VCA gain

'***ampModeSlider*** 'ensures that its value is either 0 or 1, corresponding to the two available modes of operation. When it modified, the mode will be change.

The values of these components are linked to '***AudioProcessorValueTreeState'***

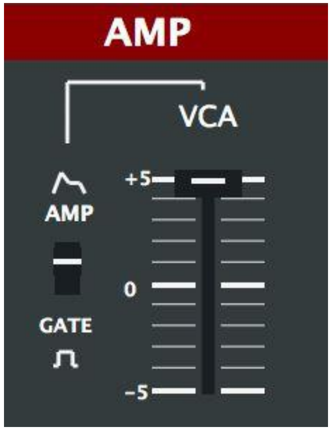
Modules-VCA

- **Implementation**

Within the 'SynthVoice' class, there exists a variable, 'vcasetting', designated for storing the value of 'vca gain'. Concurrently, a object named 'Env1' from the Maximilian library assists in implementing the audio amplification function.

```
double myCurrentVolume = env1.adsr(1., env1.trigger) * level * vcaSetting;
```

- **GUI**



- ***' vcaSlider'*** is the graphical slider that allows the user to adjust to control the VCA gain.
- ***'ModeSlider'*** is the graphical slider that allows the user to toggle between two modes of operation

Modules-Chorus

- **Working Principle**

Chorus effect is an audio effect that enriches a sound by creating a time-varying delay effect. It works by adding a slightly delayed and modulated copy of the original audio signal to the original signal.

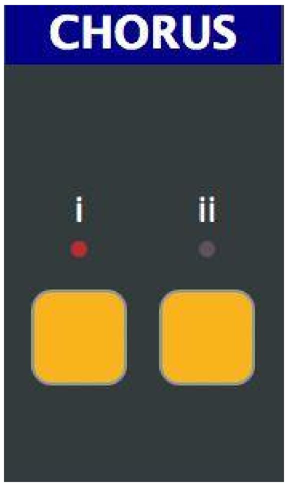
- **Implementation**

Chorus class: responsible for the user interface and interaction
ChorusEffect class: implements the core functionality of the chorus effect.

```
double ChorusEffect::processSignal(float sample, double lfoRate){
    double delayedSample = 0;
    delayedSample = delay.dl(sample, 1 + 175*( 1 + lfo.sinebuf4(lfoRate)), 0);
    return sample + delayedSample;
}
```

Modules-Chorus

- **GUI**



'chorusButton' contains 2 buttons which can control the chorus effect on and off. When the buttons clicked, the rates of the chorus effect changed.

```
double getChorusRate(){
    double chorusRate = 0;
    if(chorus1Setting > 0){ chorusRate += 0.1; }
    if(chorus2Setting > 0){ chorusRate += 0.2; }
    return chorusRate;
}
```

Thank you for your attention!

Analysis of ATLAS-06-Synthesizer

Group 12: 0.5 Musicians
Stepanov Maksim
Jian Zhou
Baichen Li
Yueqian Wu
Wenjia Luo