Stereo Audio Analyser

Introduction

This patcher has been created that can be used to generate and analyse audio from a number of sources. Shown in Figure 1 (below), the patcher utilises a range of Max objects that provide various functions to create a stereo audio analyser app, providing the user with a range of sound sources, a selection of audio effects and a set of visual displays of the sound in various forms.

Figure 1

Overview of Analyser Features

The analysers features fall in to three groups; audio source, audio effect and audio analysis. The interface has been designed to allocate the patchers features in to these groups, encouraging an ease of use within the patcher. The user is provided with a minimal but functional display, prioritising the controls over the control values to help the patch to be easy to navigate.

The patcher, shown in Figure 2 (below) has three sections; the signal source section, the effects section and the analysis section.

Figure 2

The patch interface begins at the Signal Source Section; the user operates this section to activate a sound source or audio signal, ranging from basic features such as;

- an audio input,
- 4 sine wave oscillators,
- a noise source,
- a ramp wave
- a sound file player.

Additional to the criteria of this assignment, to allow the user to control the audio signal, the Signal Source Section also includes;

- MIDI control of oscillator frequency
- an audio input and line recorder,
- a source mixer,
- a phase scope,
- channel pan control.

The sound from the Signal Source Selection then travels through the Effects Section, shown in Figure 2 (above). This section provides a range of stereo audio effects that modulate the signal in various ways. The features that are additional to the criteria include:

- a VST Insert,
- an equaliser,
- a bit crusher,
- a frequency shifter,
- a stereo delay,
- a filter.

After the signal from the Signal Source Section travels through the Effects Section, it reaches the Analysis Section, which provides the user with a trilogy of visual audio response displays as part of the basic criteria of this assignment. These displays include mono and stereo versions of;

- a dB meter,
- an oscilloscope,
- a spectrum analyser,
- a sonogram display.

As well as implementing the features given in the criteria for this assignment, additional touches are also found in the patch, including;

- oscillator level balance and frequency link,
- sample speed and time stretch,
- source mixer pan controls,
- filter, bit crusher and delay presets.

To help simplify the user interface and not to overwhelm the user with too many controls at one time, tabs are used in the Signal Source and Effects sections to display only a few controls at a time. Both sections have three tabs of controls, helping to simplify the interface at a glance. Since the patch has a lot of controls, a plain and monotone theme was prioritised to prevent the patcher from becoming too busy to look at.

How to use the Analyser

The patch begins with the On/Off button in the analysis section of the patcher at the top right corner, seen in Figure 3 (below). This button turns Max MSP's audio drivers on or off, activating the visual response displays and enabling the user to hear the audio sources.

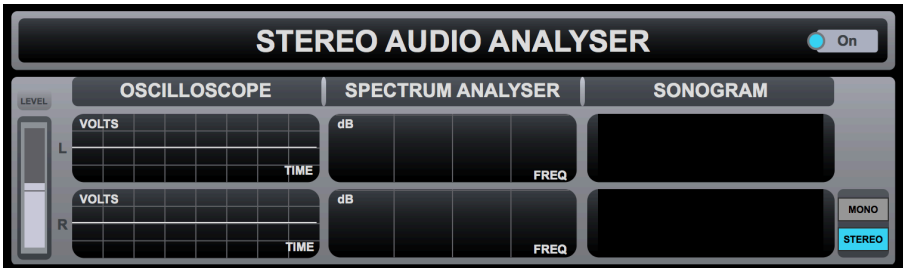

Figure 3

Once the audio driver is on, the user operates the Signal Source Section to generate a signal. The first tab of this section, shown in Figure 4 (below) offers the user a range of controls to generate sound from a series of oscillators. Each pitch-based oscillator has a frequency value control to set the pitch of the oscillator. This value can be set by operating the number boxes or by sending MIDI note values to the patcher.

Figure 4

Each oscillator also has a level control and a mute/unmute button to allow the user to control the amplitude of the oscillators. If the Harmonic Frequency control is set to 'Linked', the value of Sine oscillator 1 will be multiplied and linked with the other sine oscillators to link their frequency, creating an octave-based harmony throughout the sine wave oscillators.

The Sampler tab, shown in Figure 5 (below) offers the user controls to load samples and record line and input signal. The user can operate the sampler to load a file, clear the file, set the playback position of the file, play, stop and loop the file, change the speed and direction of the file, set the level of the sampler. The time stretch bar at the bottom of the sampler window also allows the user to time stretch the audio sample.

Figure 5

The user can operate the Recorder in Figure 5 (above) by activating the line signal to record the signal generated by Max from the oscillators and the sampler, or by activating the input signal to record the audio input routed in to Max MSP, such as a microphone. Much like the Sampler in Figure 5 (above), the Recorder allows the used to play, stop and loop the recording, adjust the level of the recording, adjust the speed and direction of the recording and clear the recording. The user can activate the Record Clip toggle to start recording, then deactivate the Record Clip toggle to stop recording, allowing the user to record a sample of any desired length.

The Mixer tab displays the source mixer, the audio menu and the phase scope, seen in Figure 6 (below). In this tab, the user can change the level and the pan settings of the oscillators, the sampler, the recorder and the main output. The Mixer tab also allows the user to mute/unmute each channel, set the audio input and output of the patcher and view the phase scope of the stereo audio output.

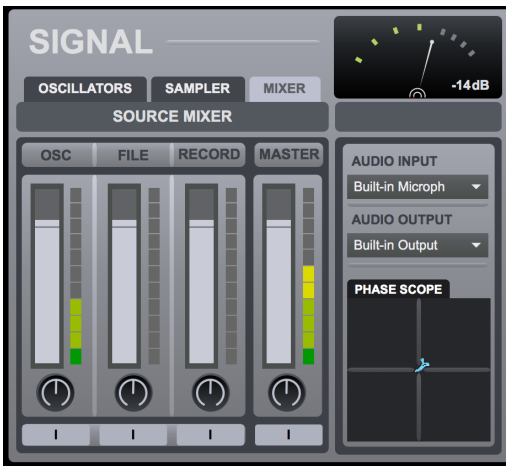

Figure 6

The effects tab in Figure 7 (below) presents the user with a Filter tab and two effects tabs. The Filter tab provides the user with a sample 4 band EQ and a filter. The user can set the level of each frequency band to adjust the EQ of the signal. The filter allows the user to set a filter type and adjust the frequency and resonance of the filter.

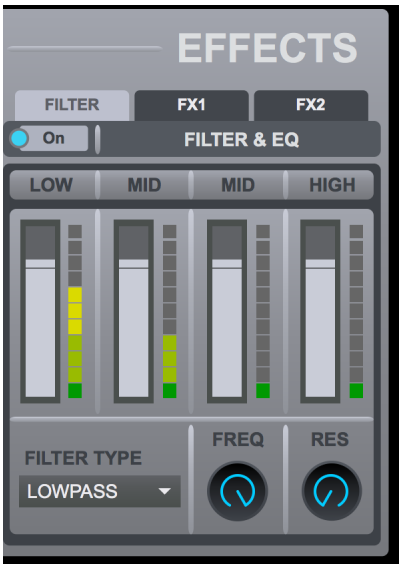

Figure 7

The FX1 tab in Figure 8 (below) opens the transpose and VST menu controls, allowing the user to change the frequency or pitch shift the audio signal. The VST menu allows the user to load and operate a VST in the signal chain.

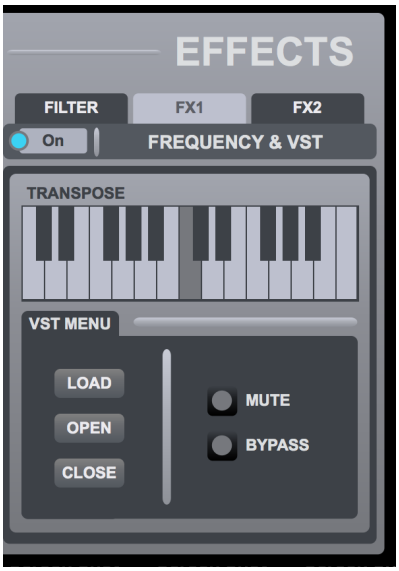

Figure 8

Once the signal has been processed through the effects chain, it reaches the system output and generates visual audio data for the user to observe. The Analyser Section in Figure 9 (below) allows the user to set the level of the analyser input and observe the oscilloscope, spectrum analyser and sonogram data in stereo or mono.

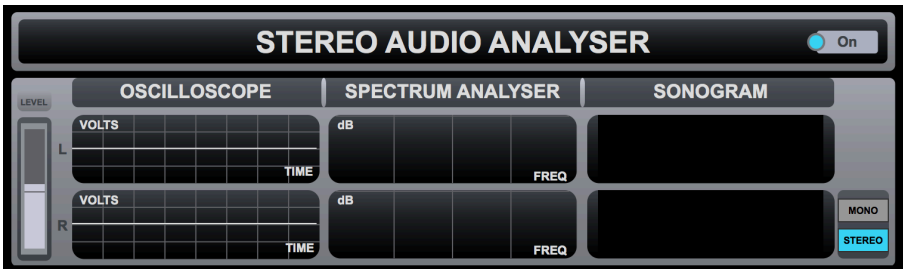
STEREO AUDIO ANALYSER
On
OSCILLOSCOPE
SPECTRUM ANALYSER
SONOGRAM
LEVEL
VOLTS
TIME
dB
FREQ
L
R
MONO
STEREO

Figure 9

How the Patcher Works

When the user opens the patcher, its default state will be set to have no audio sources playing and the audio drivers inactive.

To allow the user to activate and deactivate the audio driver, a `live.text` object is used, seen in Figure 10 (below), to trigger the audio driver in presentation mode, with messages set to indicate whether the driver is on or off.

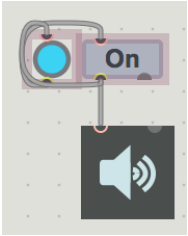

Figure 10

To provide a cleaner interface for the user to operate, each main section of the analyser is built in `bpatchers`. To show and hide these objects, `tab` objects are used, shown as item 1 in Figure 11 (below).

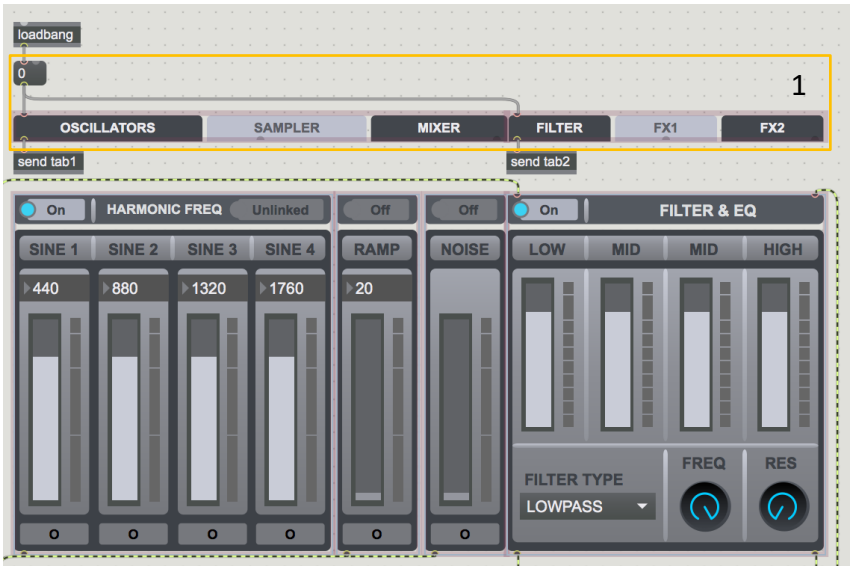

Figure 11

Each `tab` object sends a number value to a select object shown in Figure 12 (below), which selects a specific number and sends script hide and show messages to a `thispatcher` object, which hides and shows the relevant `bpatchers` for the user to operate. To ensure this feature was implemented efficiently, each `bpatcher` is given a formal Scripting Name for the program to refer to, such as Sine, Noise and Mixer. When a tab is selected, the appropriate `select` object outputs a bang, which will send script hide and script show messages to the `thispatcher` object, hiding some patchers from the presentation window but showing relevant patchers based upon the chosen tab.

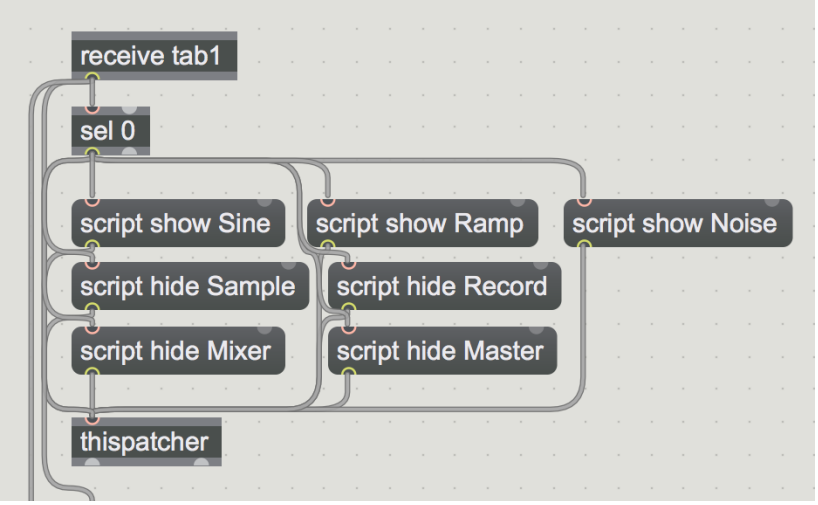

Figure 12

Figure 13 (below) shows an audio routing diagram that shows how the audio is routed from the audio sources to their audio destinations.

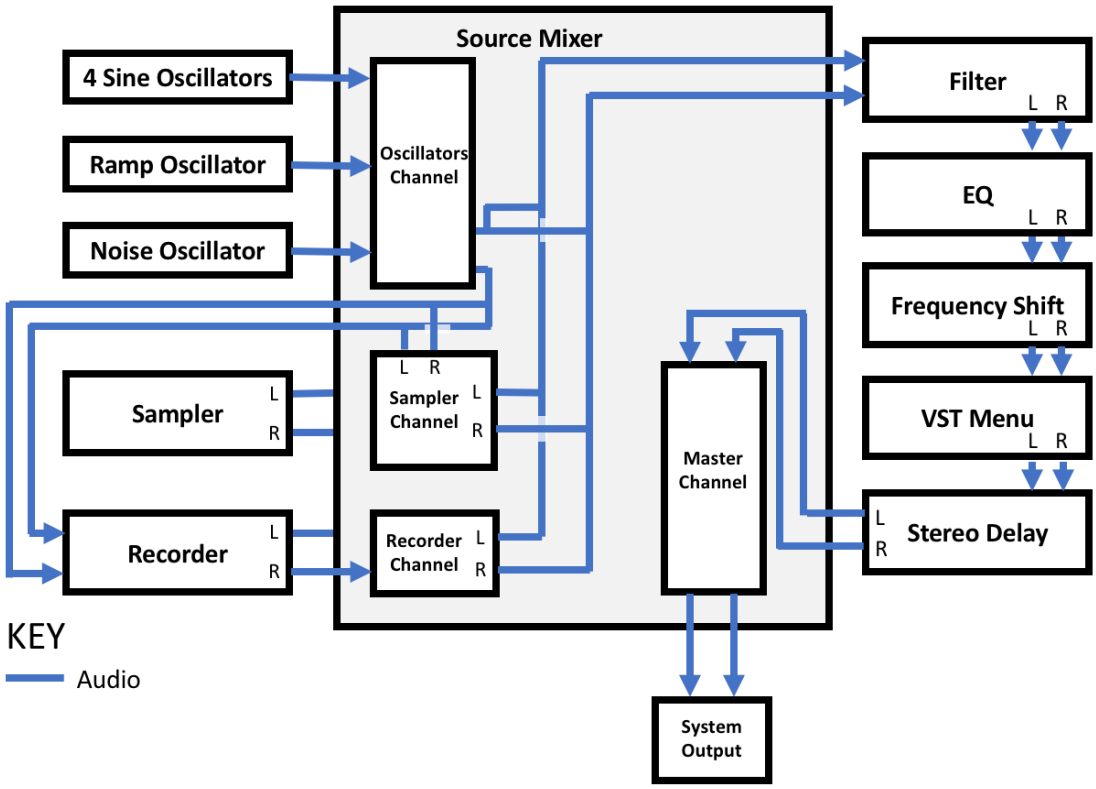

Figure 13

The first sound source comes from a series of sine wave oscillators, seen as item 1 in Figure 14 (below). Four `cycle~` objects are operated by individual `number` objects (item 2) to create sine waves. The first `cycle~` object is also controlled by a MIDI output by receiving MIDI data from a `notein` object (item 3), converting the MIDI data to frequency data using a `mtof` object (item 3) and sending it to the first oscillators `number` object. A `loadbang` object (item 4) is used to set the first oscillators default value to 440 Hz and activate the frequency link upon opening the patch.

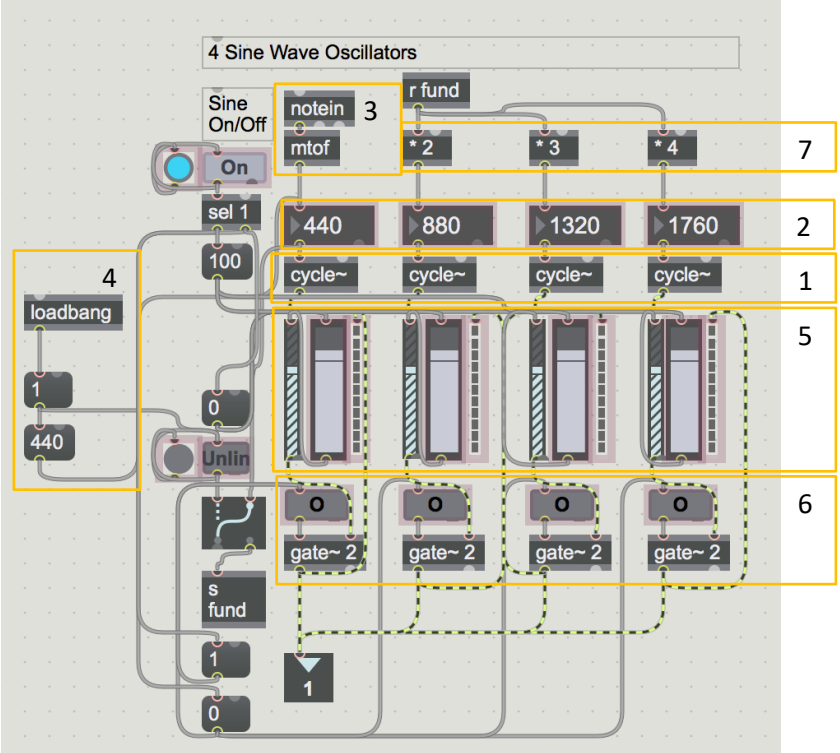

Figure 14

Each oscillator has its own amplitude control. Shown in Figure 14 (above), each oscillator travels through a `gain~` object, which is controlled by a `slider` object (item 5). The audio signal from the `gain~` object travels to a `meter~` object (item 5) to provide a visual response of the oscillators amplitude. Then, the signal travels through a `gate~ 2` object which is controlled by a `live.text` object (item 6) to provide a mute/unmute feature for the user. In order for the oscillators to harmonize, the frequency value from oscillator 1's `number` object travels through a trio of multiply (`*`) objects (item 7) to create a series of harmonic values for each oscillator.

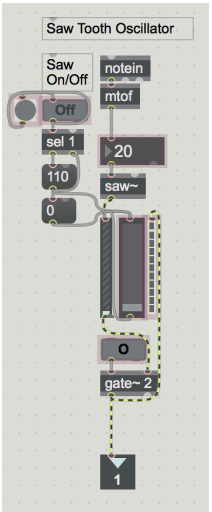

Figure 15

Similarly, the ramp wave oscillator in Figure 15 (above) and the noise generator in Figure 16 (below) are operated with `number` objects and have amplitude and mute controls from the same objects as the sine wave oscillators in Figure 14 (above).

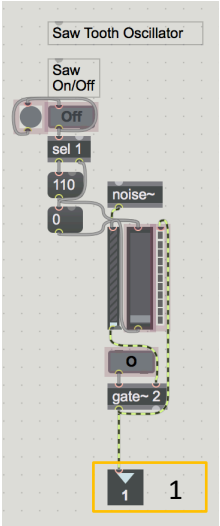

Figure 16

In order for the audio data to leave the oscillators `bpatcher`, the audio signal is routed to an `outlet` object, allowing the signal to leave the bpatcher (item 1, Figure 16) and travel through the main patcher.

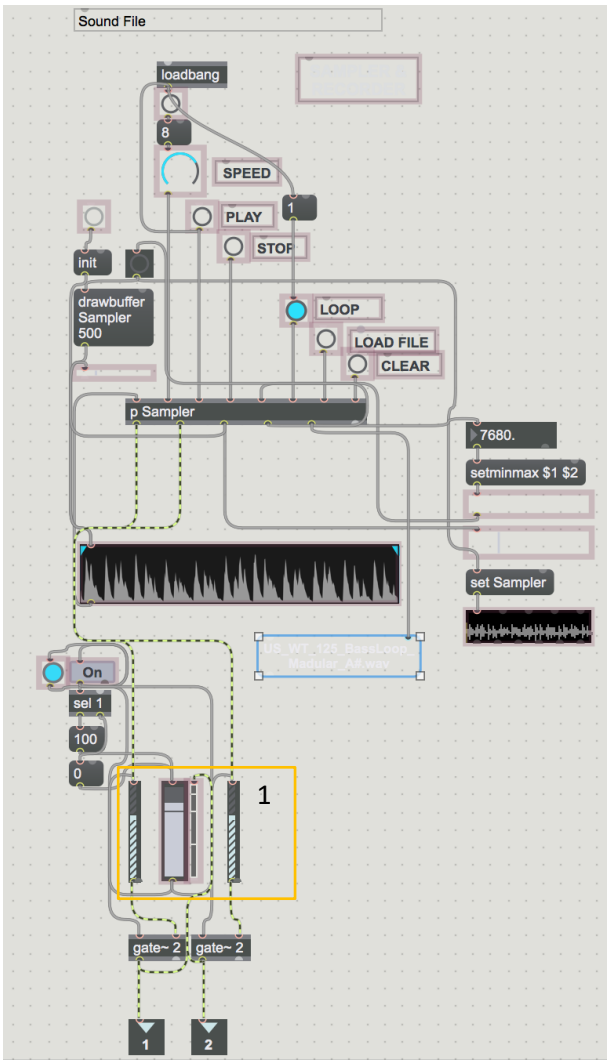

Figure 17

Figure 17 (above) shows the objects used in creating the sampler. A range of user interface objects, such as `button` and `led` objects are used to control the `groove~` (item 1) and `buffer~` (item 2) objects found in the `p Sampler` patcher in Figure 18 (below), which is linked with the bpatcher using `inlet` and `outlet` objects (item 3).

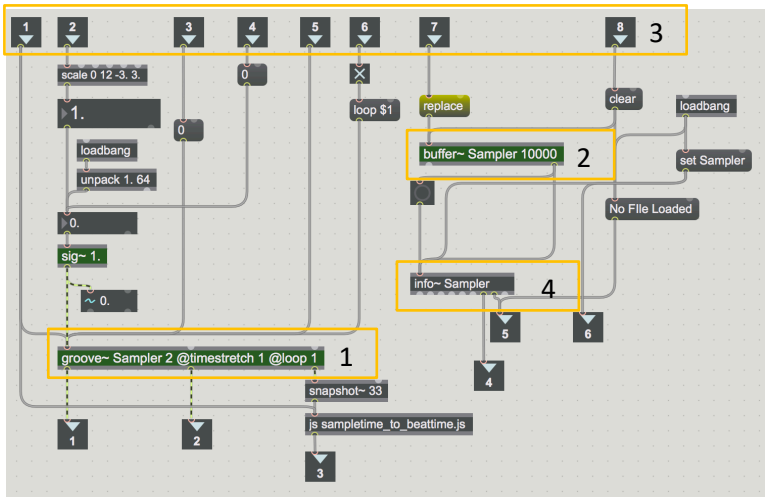

Figure 18

The `outlet` objects take data from the `info~ Sampler` object (item 4) and set Sampler `message` object to provide information about the sampler to the user, such as `waveform~` object information and file name information to the cold inlet of a `message` object. To ensure the sample is in stereo, the signal travels from both channels from the sampler's `groove~` object before travelling through two `gain~` objects controlled by a `slider` object (item 1), seen in Figure 18 (above).

The recorder system, shown in Figure 19 (below), takes its stereo input from an `ezadc~` object (item 1). After travelling through two amplifiers to reduce the amplitude of the signal (item 2) and a `gate~ 2` object controlled by the user interface from the 'Input' `live.text` object, the input signal reaches a `record~` object (item 3), used to record the input signal from the audio input. The audio signal from the oscillators and sampler also travels to the `record~` object from two inlets.

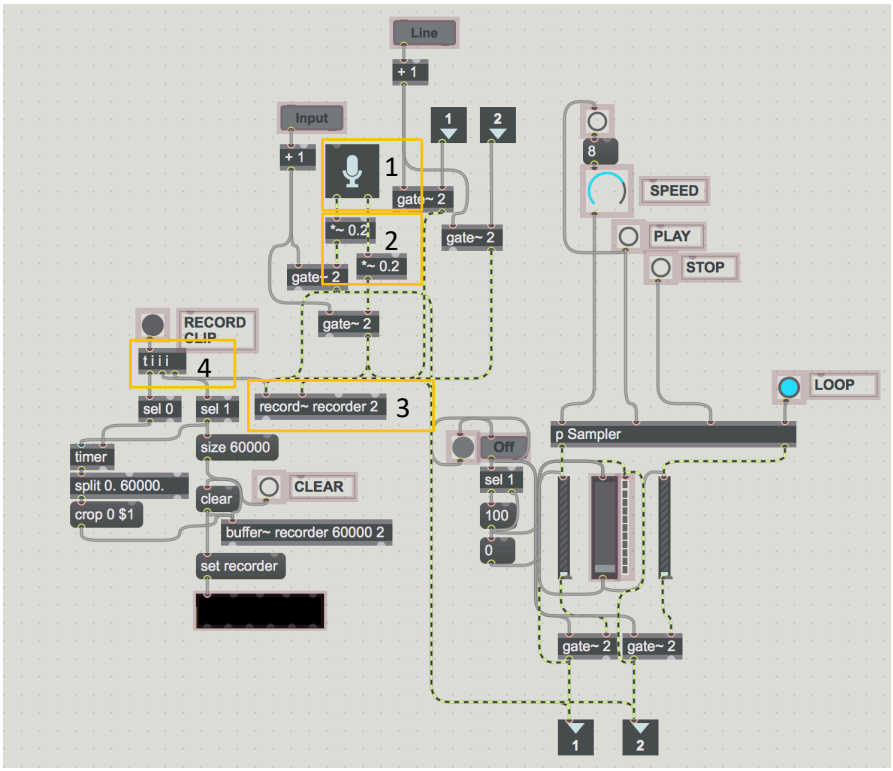

Figure 19

When the user operates the `led` object to record the clip, the signal is triggered in multiple places from a `t i i i` object (item 4), which initiates a sequence of objects that allows the user to record the sample for as long as desired. When the record clip button is deactivated, the signal stops the recording and defines the recorded clip length to the elapsed time, which ensures that the clip is recorded entirely and for as long as desired. Shown below in Figure 20, a `buffer~` object is used to store the clip and a `message` object is used to provide visual feedback of the recorded sample to the `waveform~` object. Similar to the `patcher` object in the sampler in Figure 19 (above), a `groove~` object is stored in a `patcher` object, shown in Figure 20 (below).

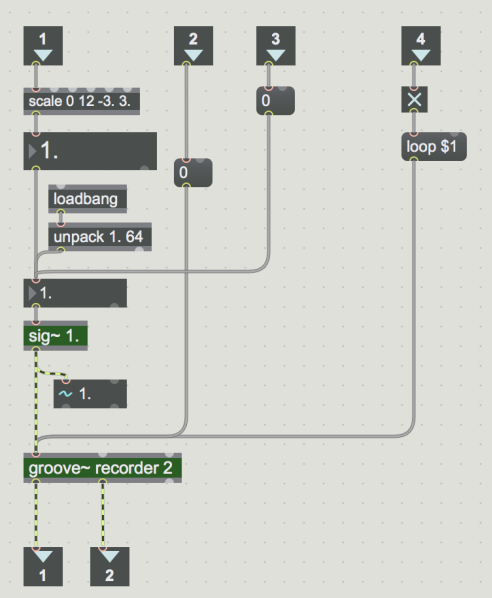

Figure 20

The signal leaves the `groove~` object in stereo and travels through two outlets to reach the recorder bpatcher, before travelling through two `gain~` objects and leaving the bpatcher in stereo.

The source mixer, shown in Figure 21 (below), takes input from the audio sources through `inlet` objects (item 1) before routing the signal through `gain~` objects controlled by `slider` objects, paired with `meter~` objects (item 2) to provide a visual response of each channels loudness.

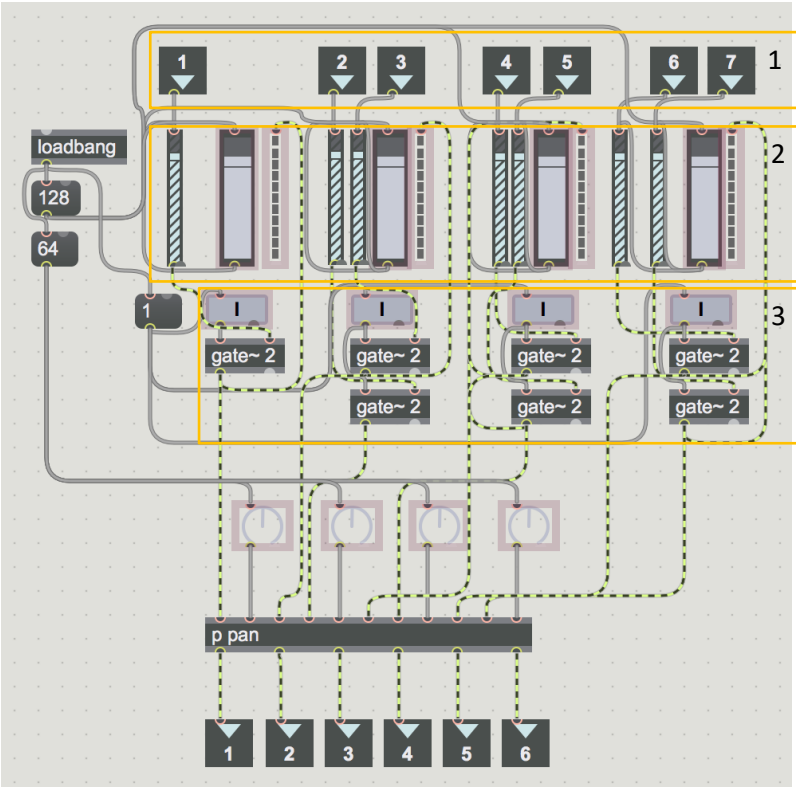

Figure 21

After the signal from each channel travels through `gate~ 2` objects controlled by `live.text` objects (item 3) to mute/unmute the channels signal, the audio is routed to a `patcher` object for the pan, shown in Figure 22 (below). The patcher takes its `inlet` data from each channels audio signal and a set of `live.dial` objects to control the pan from outside the patcher. The `live.dial` object signal controls the `gain~` objects (item 1) shown in Figure 22 (below), with a `scale` object (item 2) inverting the right signal of the `live.dial` object. By setting the Number of Steps of the `gain~` object to 64, the `live.dial` will operate the `gain~` objects as an effective pan. The audio from the pan is then routed to the output via the stereo pair `outlet` objects (item 3).

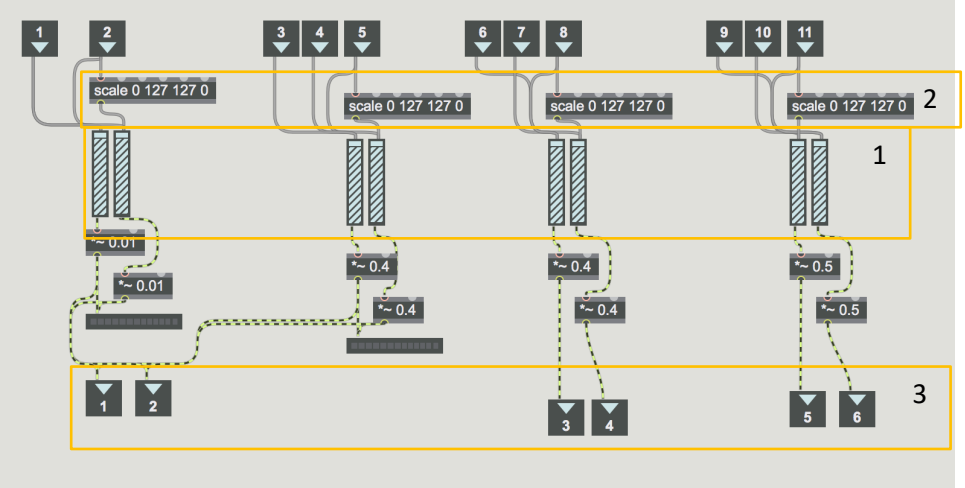

Figure 22

From here, the signal travels to the filter and EQ unit. Shown in Figure 23 (below), the stereo input travels through two `svf~` and `selector~ 4` objects (item 1), controlled by a pair of `live.dial` objects (item 2) to operate the filter cutoff and resonance controls. A `umenu` object (item 3), shown in Figure 23 (below) is used to send filter type messages to the `selector~ 4` objects to allow the user to operate a variation of filter types.

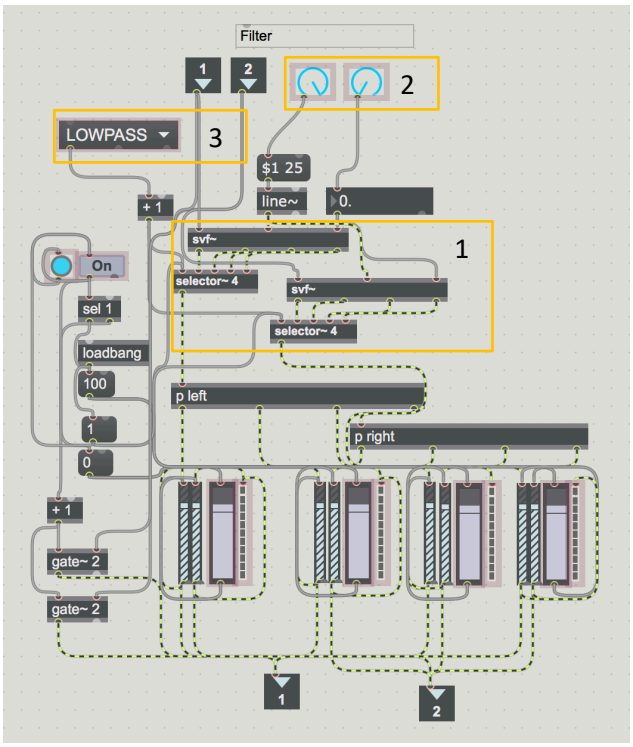

Figure 23

From the `selector~ 4` stereo outputs, the signal then travels in to two patchers, shown in Figure 24 (below). These patchers contain a string of `cross~` objects, which filter the audio at specified values, sending the lowpass filtered audio to an audio outlet and sending the remaining filtered data to the next filter in the series. This string of filter objects splits the signal in to four frequency bands for the user to control.

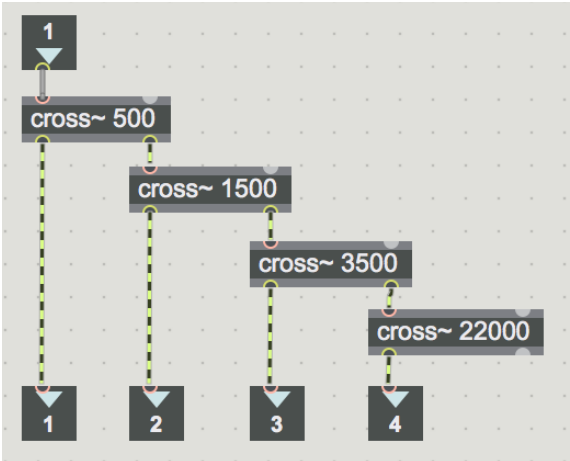

Figure 24

To ensure that the signal remain in stereo, two patcher objects are used with this string of filters to filter the left and right channels separately. From here, the output from the two patchers is routed in to pairs of `gain~` objects seen in Figure 24 (above), controlled by a `slider` object. The audio is routed in to `meter~` objects to give the user a visual response of each bands amplitude. Shown in Figure 24 (above), the audio is then routed through two outlet objects, capturing the left and right outputs of each frequency band.

The frequency shift was achieved with the use of a `pfft~` object (item 1) in Figure 25 (below). Although this object was not fully understood at the time of its instalment, it was implemented well and tested thoroughly. The stereo pair of `pfft~` objects receive data from the `transratio` object (item 2), which converts the frequency value of the `kslider` object (item 3) to note transposition data.

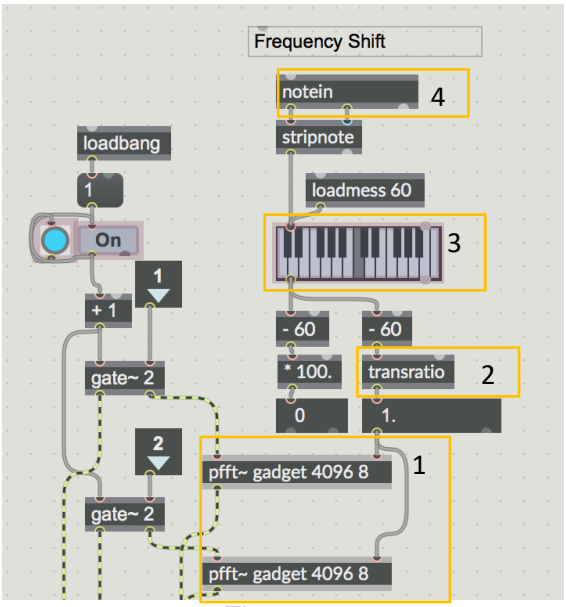

Figure 25

Whilst the user is able to operate the frequency shift with the `kslider` object in the interface of the patcher, the user can also send MIDI data to the patcher, which will be received by the `notein` object (item 4), allowing the user to control the pitch shift with a MIDI keyboard.

The VST menu, shown below in Figure 26, uses a `vst~` object (item 1) to implement a VST input in the effects chain. The `vst~` object takes the stereo audio signal whilst receiving data from a series of `message` objects that send commands to load, open and bypass VST inserts in the patcher.

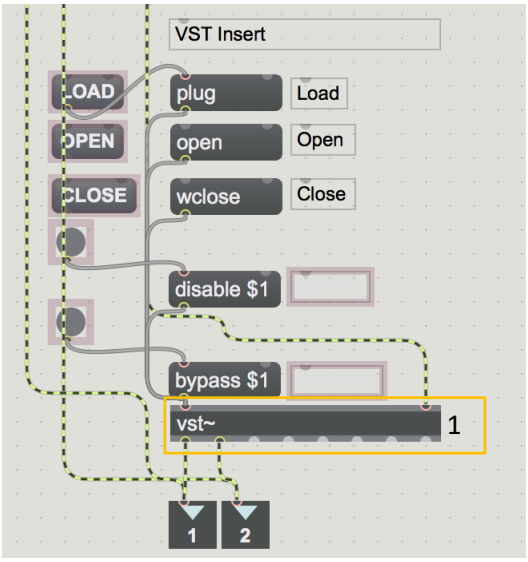

Figure 26

To ensure the user interface is more understandable, `message` and `led` objects are used in the interface to trigger the command `message` objects. This helps the patch to be easy to understand.

From the pitch shift and VST bpatcher, the stereo signal is routed in to the bitcrusher and delay bpatcher. The signal then reaches a pair of `tapin~` objects (item 1), shown in Figure 27 (below). The `tapin~` object acts as an input to a delay line. Paired with a `tapout~` object (item 2), the signal is delayed by a set value, controlled by the dials in the user interface. To implement feedback in the stereo delay, the output of the `tapout~` object (the delayed signal) is routed to an amplifier, shown in Figure 27 (below), with a `dial` object granted to the user in the interface of the patcher, which controls the amplitude of the amplifier. To ensure the signal does not get too loud, the amplifier is limited to multiply 0.85.

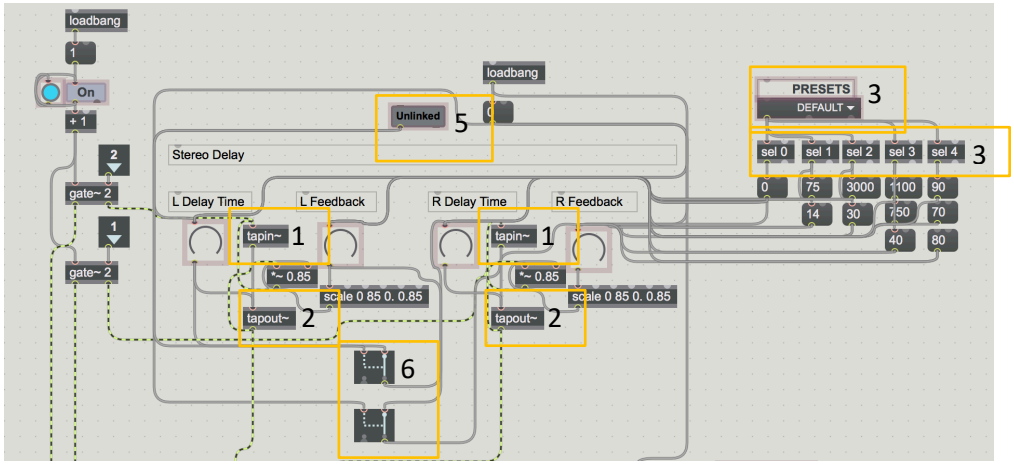

Figure 27

The user is provided with a range of preset parameters for the stereo delay unit. Shown in Figure 27, a `umenu` object (item 3) is implemented with a list of names relevant to the values parameters. When a `umenu` preset is recalled, a series of `sel` objects (item 4) triggers a bang, depending on the output value of the `umenu` object. Each `sel` object is followed by a sequence of `message` objects; when the `sel` object is triggered by the output value fo the `umenu` object, the following `message` objects send number values to the interface controls of the delay effect.

The user also has the option to link and unlink the stereo controls of the delay unit. A linked/unlinked `live.text` object (item 5) sends a trigger signal to a pair of `gswitch2` objects (item 6) in Figure 27 (above). If open, these switches send the data from the left stereo delay controls to the right stereo delay controls, linking the left controls with the right controls.

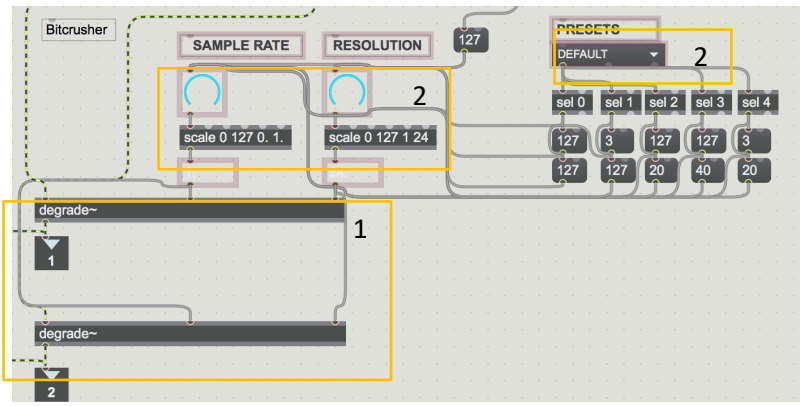

Figure 28

The bitcrusher is achieved with a pair of `degrade~` objects (item 1), shown in Figure 28 (above). Two `dial` objects (item 2) travel through `scale` objects to scale the dials to a suitable value for the `degrade~` object. Similar to the stereo delay in Figure 27 (above) the bitcrusher presents the user with a `umenu` (item 3) object with the ability to recall preset parameters for the bitcrusher controls.

Once the signal has travelled through the effects chain and returned to the source mixer, the signal is then sent to the Output `bpatcher`. This patcher, shown in Figure 29 (below), takes the stereo signal from two `inlet` objects and adds/subtracts the signal in to an amplifier (item 1), routing the signal in to a `scope~` object (item 2) to visualise the stereo phase of the audio signal.

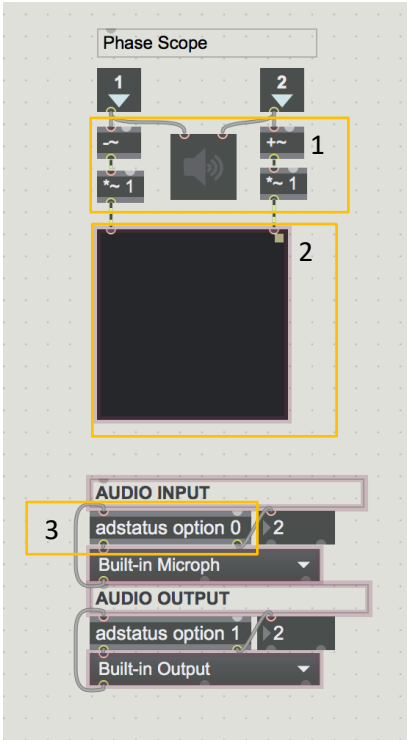

Figure 29

The Output patcher provides the user with `umenu` objects to set the audio input and output of the patcher. Once the patcher is opened, the `adstatus` object (item 3) reports the audio driver settings, listing them in to a `umenu` object. The same `umenu` object routes the audio driver settings in to the `adstatus` object, allowing the user to change the driver's settings.

Figure 30 (below) shows the master output from the source mixer routed to a stereo pair of `gain~` objects controlled by a `slider` object (item 1), which control the level of the signal routed to the visual response displays, such as the oscilloscope and the sonogram. The dB meter (item 2) in Figure 30 (below) has been added to the interface of the patch along with a message box (item 3) to display the number value of the dB meter.

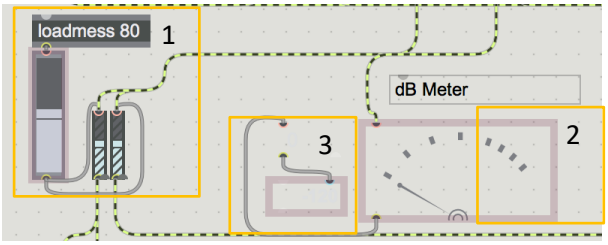

Figure 30

The mono and stereo analyser `bpatcher` objects in Figure 31 (below) uses `inlet` objects to receive the stereo audio signal of the patcher before routing the signal to the varied visual response displays. The oscilloscope is achieved by using a `scope~` object, the spectrum analyser is achieved by using a `spectroscope~` object and the sonogram is achieved by using a `spectroscope~` with the objects Display Mode set to Sonogram. The user is able to display either the mono or stereo version of this `bpatcher` object.

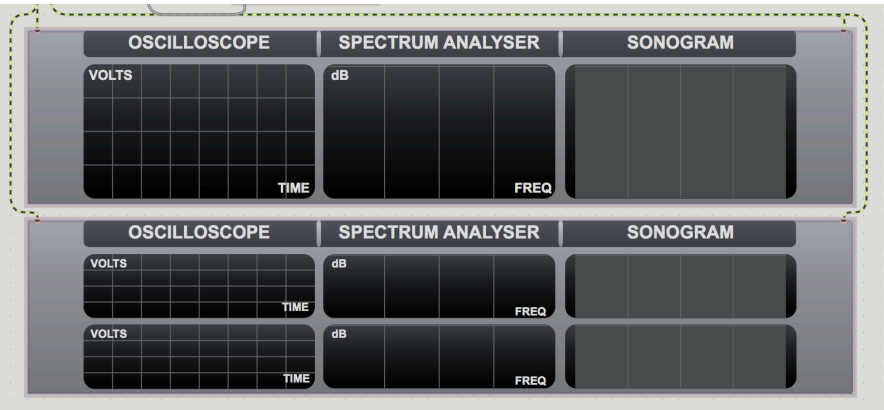

Figure 31