OB-Xd Synth Analysis

Group ID: 6 - FabFour

Amico Stefano Antonio, 10937333

Anand Abhijeet, 10859656

Pletti Manfredi, 10493741

Portentoso Alice, 10664207

May 5, 2023

1 Introduction

The Oberheim OB-X was the first of Oberheim's OB-series polyphonic analog subtractive synthesizers. First commercially available in June 1979, the OB-X was introduced to compete with the Sequential Circuits Prophet-5, which had been successfully introduced the year before.

Figure 1: Oberheim OB-X

The OB-X was the first Oberheim synthesizer based on a single printed circuit board called a "voice card" (still using mostly discrete components) rather than the earlier SEM (Synthesizer Expander Module) used in Oberheim semi-modular systems, which had required multiple modules to achieve polyphony. The OB-X's memory held 32 user-programmable presets. The synthesizer's built-in Z-80 microprocessor also automated the tuning process. This made the OB-X less laborious to program, more functional for live performance, and more portable than its ancestors.

The OB-Xd:

https://github.com/reales/OB-Xd

is based on the Oberheim OB-X. It attempts to recreate its sound and behaviors, but as the original was very limited in some important ways a number of things were added or altered to the original design. The OB-Xd is written in C++ and utilizes the Juce Framework (v. 7.0.3 suggested to compile).

2 Key components and functionality

Figure 2: Ob-xd UI

Initialization: The ObxdAudioProcessor main class initializes the synthesizer's components, such as the voice manager, LFOs, filters, and oscillators.

User Interface: The ObxdAudioProcessorEditor class handles user interaction with the synthesizer's controls, such as knobs and buttons. The Juce framework processes the user input and updates the synthesizer parameters accordingly.

MIDI Input: The MidiMap class handles MIDI events, such as note-on and note-off messages, and interacts with the voice manager to handle these events.

Voice Management: The ObxdVoice class handles voice allocation and updates for each voice. It processes the output of individual voices, including envelopes, oscillators, filters, and LFO modulation.

Audio Output: The mixed output of all active voices is sent to the audio output provided by the Juce framework in the processBlock() function of PluginProcessor.cpp.

Patch Management: The Juce framework handles patch saving and loading. Users can save and load patches using the host DAW's preset management features.

The following flow graph respresent the main blocks of OB-Xd:

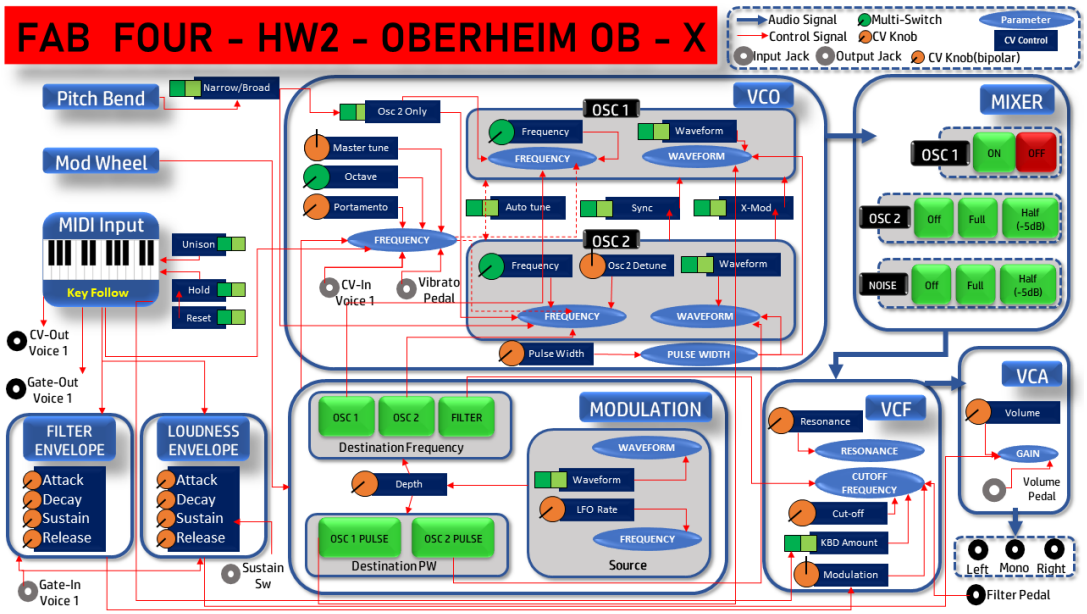


Figure 3: Flow-chart of Ob-xd - see end of report for bigger picture

3 Innovation - Continuous blendable multimode filter

As an innovation to the physical hardware synth, OB-Xd implement a continuous blendable multimode filter. It is a type of audio filter that can smoothly transition between different filter types.

In this case, the filter types are High Pass (HP), Notch, Band Pass (BP), and Low Pass (LP) filters; the term "multimode" refers to the ability to switch between these different modes.

The filter provides two configurations: 12 dB per octave and 24 dB per octave. In the 12 dB mode, it offers HP-Notch(BP)-HP filters, while in the 24 dB mode, it uses a 4-pole LP filter that can transition to a 1-pole HP filter.

The 12 dB mode uses a second-order filter (two poles), and the 24 dB mode uses a fourth-order filter (four poles). The "dB per octave" refers to the rate at which the signal is attenuated once it goes beyond the cutoff frequency of the filter. In a 12 dB per octave filter, the signal level drops by 12 dB for every octave the frequency moves away from the cutoff frequency.

To understand the math behind these filters, we need to look at the transfer function of each filter type:

High Pass filter:

$$H_{hp}(s) = \frac{s}{s + \omega_c}$$

where s is the complex frequency and w_c is the cutoff frequency

Low Pass filter:

$$H_{lp}(s) = \frac{\omega_c}{s + \omega_c}$$

Band Pass filter:

$$H_{bp}(s) = \frac{s}{s^2 + \frac{s \cdot \omega_c}{Q} + \omega_c^2}$$

where Q is the quality factor of the filter.

Notch filter:

$$H_{notch}(s) = \frac{s^2 + \omega_c^2}{s^2 + \frac{s \cdot \omega_c}{Q} + \omega_c^2}$$

The continuous blendable multimode filter essentially takes a weighted average of these filters, allowing for smooth transitions between them.

The original code is handled in the Filter class (Engine/Filter.h). The Filter class is responsible for morphing between different filter types, here are the key methods and variables of the class:

Instance variables:

s1, s2, s3, s4: These variables store the internal states of the filter, used in the processing of the input signal.

R, R24: Resonance variables for the 2-pole and 4-pole filters, respectively.

rcor, rcorInv: Variables for compensating the resonance change when the sample rate is adjusted.

rcor24, rcor24Inv: Similar to rcor and rcorInv, but for the 4-pole filter.

mm, mmt, mmch: Variables for handling the multimode blending.

SampleRate, sampleRateInv: The sample rate of the filter and its inverse.

bandPassSw: A boolean flag indicating whether the band-pass switch is enabled.

selfOscPush: A boolean flag indicating whether self-oscillation is pushed.

Filter() constructor: Initializes the filter with default values.

setMultimode(float m): Sets the blending factor (m) for the multimode filter. It calculates the integer part of mm * 3 as mmch, and the fractional part as mmt.

setSampleRate(float sr): Sets the sample rate (sr) for the filter and calculates the sampleRateInv. It also adjusts the rcor and rcor24 variables to compensate for the change in the sample rate.

setResonance(float res): Sets the resonance (res) for the filter. The variable R is set to 1 - res, while R24 is set to 3.5 * res.

diodePairResistanceApprox(float x): This is an approximation function used to model the non-linear behavior of a diode pair in the filter. It returns an approximate value based on a Taylor series.

NR(float sample, float g): This function calculates the non-linear resonance for the 2-pole filter. It takes an input sample and the cutoff frequency parameter (g) and computes the feedback using the diode pair approximation.

Apply(float sample, float g): This function applies the 2-pole filter to the input sample using the

cutoff frequency parameter **(g)**. It first calculates the nonlinear resonance **(NR)** and updates the internal states **s1** and **s2**. It then computes the multimode output based on the mm variable and the states.

NR24(float sample, float g, float lpc): This function calculates the non-linear resonance for the 4-pole filter, similar to NR for the 2-pole filter. It computes the output using the internal states and the input sample.

Apply4Pole(float sample, float g): This function applies the 4-pole filter to the input sample using the cutoff frequency parameter (g). It calculates the nonlinear resonance (NR24) and updates the internal states s1, s2, s3, and s4. It then computes the multimode output based on the mmch and mmt variables and the states.

The most important innovation is the blendable control as figured in Filter class.

The **setMultimode** method sets the **mm** value and calculates **mmch** and **mmt**. The **mm** value determines how the output is combined from different stages of the filter.

```
void setMultimode(float m)
{
    mm = m;
    mmch = (int)(mm * 3);
    mmt = mm*3-mmch;
}
```

Listing 1: Blending

The **bandPassSw** variable is a boolean flag that, when set to true, enables the band-pass filter mode. By default, it is set to false, which means the filter operates in low-pass/high-pass mode. The output of the filter is determined in the Apply function based on the **bandPassSw** variable and the **mm** value:

```
float mc;
if (!bandPassSw)
{
    mc = (1 - mm) * y2 + (mm) * v;
}
else
{
    mc = 2 * (mm < 0.5 ?
        ((0.5 - mm) * y2 + (mm) * y1) :
        ((1 - mm) * y1 + (mm - 0.5) * v)
    );
}
```

Listing 2: Blending

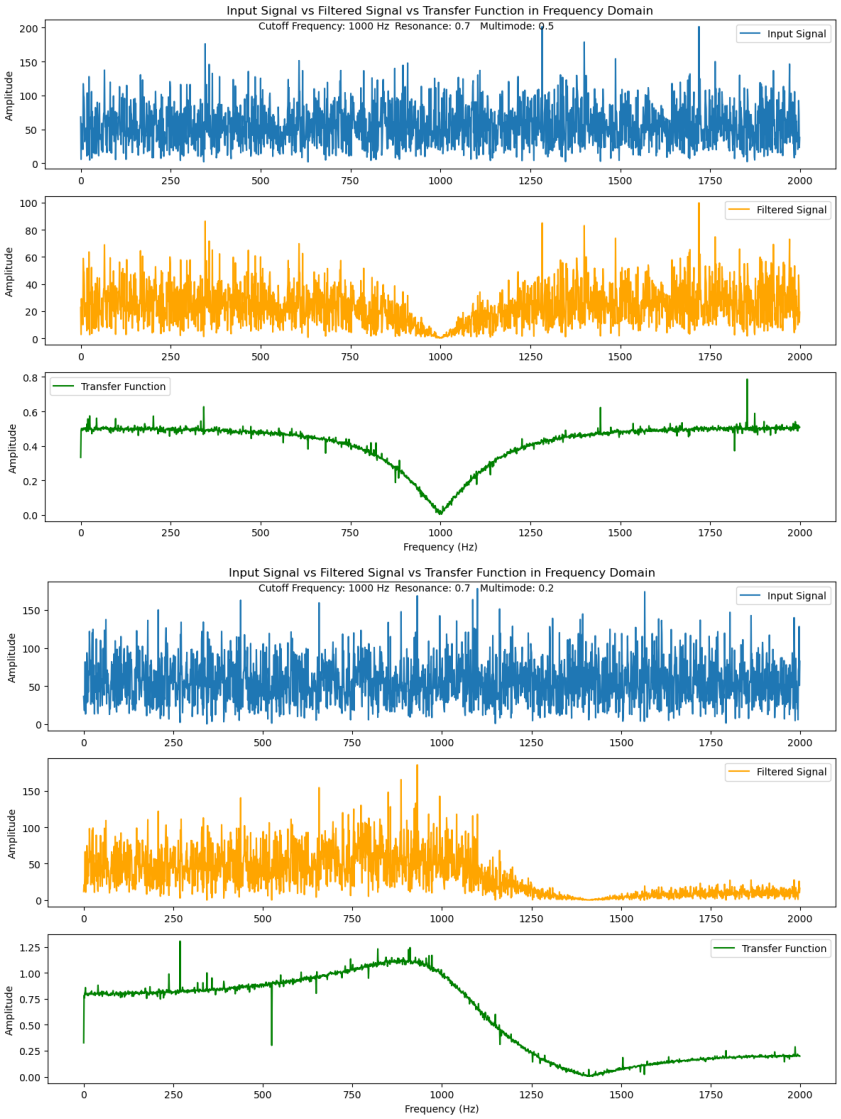
If **bandPassSw** is false, the output is a combination of high-pass (HP) and low-pass (LP) filter outputs, with mm controlling the blend between them. When mm is 0, the output is a low-pass filter,

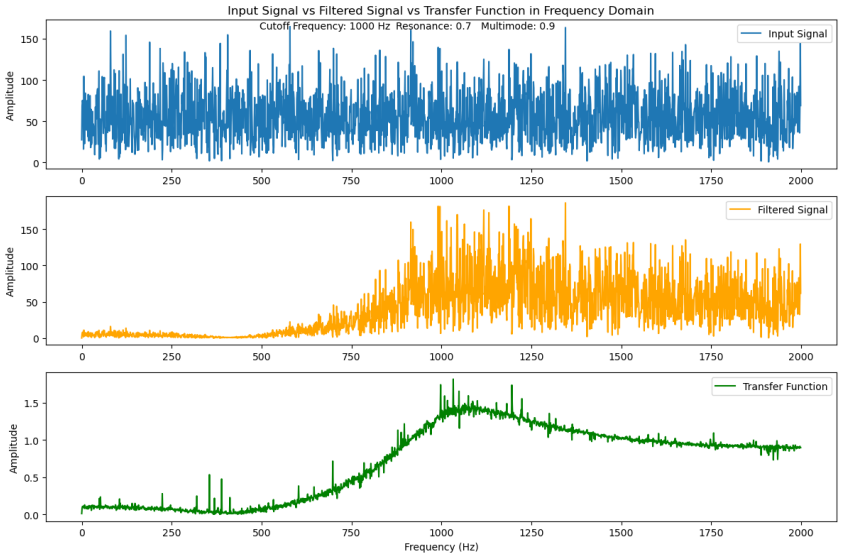
and when mm is 1, the output is a high-pass filter. If **bandPassSw** is true, the output is a combination of band-pass (BP) and notch filter outputs.

The **mm** value controls the blend between them. When mm is 0.5, the output is a notch filter, and when mm is less than 0.5 or greater than 0.5, the output is a band-pass filter with varying bandwidth.

We have tested this blendable filter in python (see notebook file in github repo).

Generated plots are self-explanated:





4 Conclusion

The OB-Xd synthesizer effectively emulates the sound and features of the classic Oberheim OB-X analog synthesizers while incorporating modern improvements. Its well-structured codebase, built on top of the Juce Plugin Framework, ensures compatibility with various DAWs and platforms.

The project's open-source nature allows for further development, feature additions, and customization by the community, but precompiled version are commercially distributed for various platform.

The continuous blendable multimode filter has been rewriten in Python and tested in depth. It could be for sure inspiration for other projects.

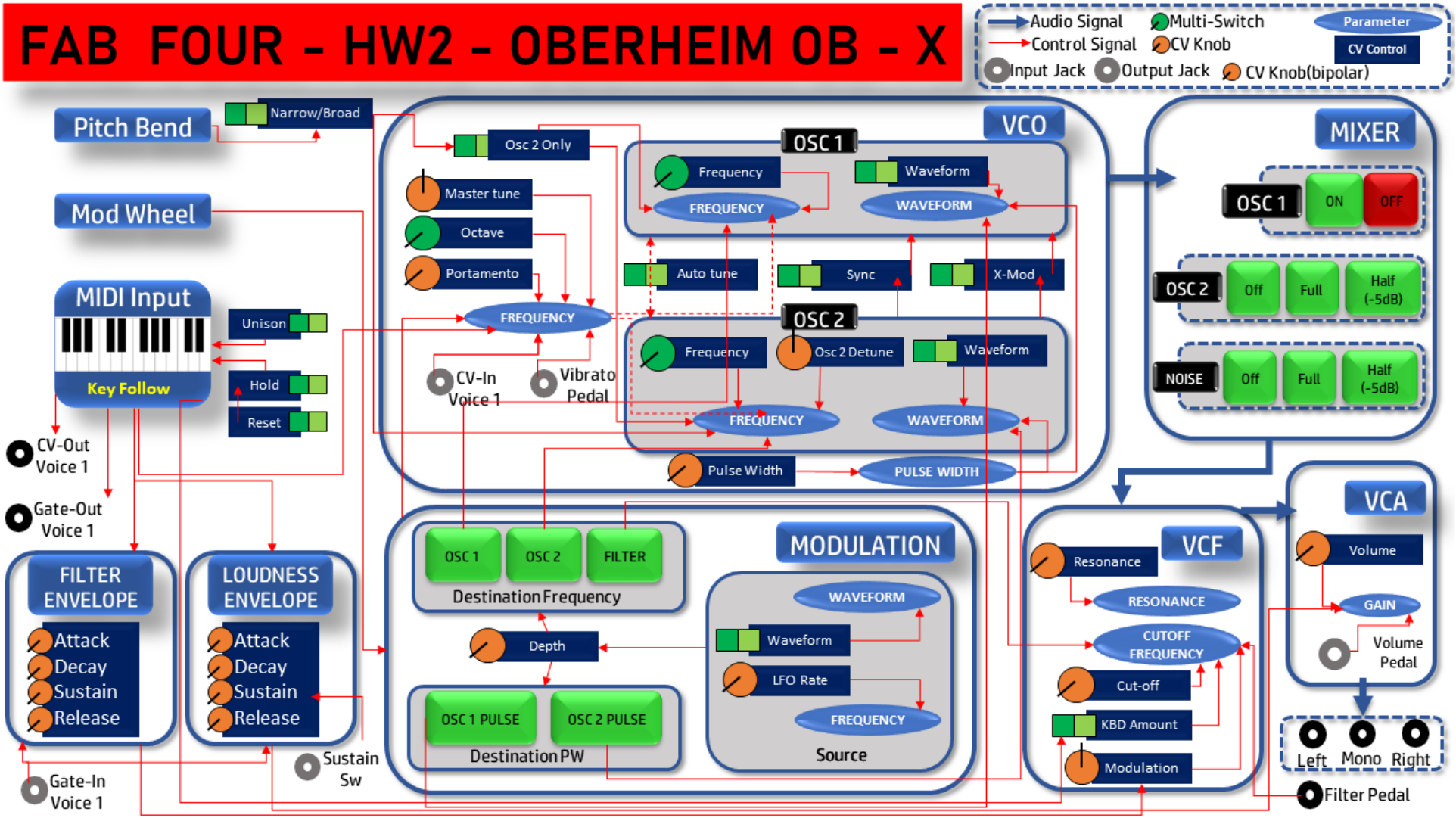
5 Bibliography

https://en.wikipedia.org/wiki/Oberheim-OB-X

https://www.discodsp.com/obxd

https://oberheim.com

https://www.vintagesynth.com/oberheim/obx.php


FAB FOUR - HW2 - OBERHEIM OB - X
Audio Signal
Control Signal
Multi-Switch
CV Knob
Parameter
CV Control
Input Jack
Output Jack
CV Knob(bipolar)
Pitch Bend
Narrow/Broad
Mod Wheel
MIDI Input
Key Follow
Unison
Hold
Reset
CV-Out Voice 1
Gate-Out Voice 1
Gate-In Voice 1
Sustain Sw
VCO
Osc 2 Only
Master tune
Octave
Portamento
FREQUENCY
CV-In Voice 1
Vibrato Pedal
OSC 1
Frequency
Waveform
FREQUENCY
WAVEFORM
Auto tune
Sync
X-Mod
OSC 2
Frequency
Osc 2 Detune
Waveform
FREQUENCY
WAVEFORM
Pulse Width
PULSE WIDTH
MIXER
OSC 1
ON
OFF
OSC 2
Off
Full
Half (-5dB)
NOISE
Off
Full
Half (-5dB)
FILTER ENVELOPE
Attack
Decay
Sustain
Release
LOUDNESS ENVELOPE
Attack
Decay
Sustain
Release
MODULATION
OSC 1
OSC 2
FILTER
Destination Frequency
Depth
OSC 1 PULSE
OSC 2 PULSE
Destination PW
WAVEFORM
Waveform
LFO Rate
FREQUENCY
Source
VCF
Resonance
RESONANCE
CUTOFF FREQUENCY
Cut-off
KBD Amount
Modulation
VCA
Volume
GAIN
Volume Pedal
Left
Mono
Right
Filter Pedal