StringSauce:
A Semantically-Mapped Multi-Stage Guitar Bus Processor

Technical Documentation and Research Report

Khris Finley
Department of Computer Science
Trent University

This report documents the design and implementation of *StringSauce*, a semantic, macro-controlled guitar bus processor developed as part of undergraduate research in audio DSP and software engineering. It is intended to serve as foundational work toward future graduate research in perceptually-informed DSP interfaces and pedagogically-driven audio processing tools.

Contents

Abstract

This document presents *StringSauce*, a semantic, macro-driven guitar bus processor designed to streamline the spectral, dynamic, harmonic, and spatial shaping of grouped guitar tracks. Implemented in modern C++ using the JUCE framework, the system introduces six perceptually-grounded macro controls (*character*, *thump*, *body*, *shimmer*, *slap*, and *space*) that map to deterministic multi-parameter transformations across four DSP modules: Equalizer, Dynamics, Saturation, and Spatial processing. A mode-dependent routing architecture further adapts the signal flow to common production roles for electric guitar.

Technically, textitStringSauce demonstrates real-time DSP implementation with nonlinear processing, numerically stable IIR filtering, parameter smoothing, and structured module orchestration. Conceptually, it explores how higher-level perceptual descriptors can serve as intuitive control primitives, enabling users to make musically meaningful adjustments without precision knowledge of lower-level DSP parameters.

As an undergraduate research project, this work also lays the foundation for a broader graduate-level research direction centered on perceptually-informed audio processing and pedagogically-motivated interface design. By treating semantic descriptors as first-class citizens in the DSP mapping process, textitStringSauce illustrates a path toward audio tools that are not only effective for production workflows, but also supportive of clearer, more intuitive learning of audio concepts. Future work will extend this framework through formal perceptual evaluation, generalized mapping strategies, and the development of educational DSP tools.

1 Introduction

In contemporary guitar production workflows, mixing engineers frequently route multiple guitar tracks, be they double-tracked rhythms, harmonized leads, or layered clean textures, into a shared bus for collective processing. At this stage, the goal is rarely to perform drastic tone shaping, which is typically handled upstream by recorded amplifiers, in-box amp simulators, or individual channel processing. Instead, bus processing aims to provide cohesion, frequency balance, controlled dynamics, and spatial uniformity across several sources.

Despite this, most available tools for guitar bus processing are either adapted from general-purpose channel strip plugins or consist of ad hoc chains assembled manually by the engineer. These approaches offer great technical flexibility, but often lack intuitive control vocabulary. Users must operate on individual DSP parameters such as compressor threshold, Q factor, or frequency cutoff, which do not always intuitively map directly to client production goals like "tighten the lows", "add shimmer", or "make it punchier."

StringSauce addresses this gap by providing a real-time guitar bus processor built around a small set of perceptually meaningful macro controls. These controls (*character*, *thump*, *body*, *shimmer*, *slap*, and *space*) form a semantic interface for shaping multi-track guitar buses. Internally, the plugin contains a multi-stage processing chain consisting of equalization, dynamics, saturation, and spatial effects. A dedicated parameter mapping system translates these macro controls into deterministic module parameters, allowing complex transformations to be expressed through simple gestures.

Unlike a guitar amp simulator, *StringSauce* is not designed to emulate amplifiers or modify the fundamental timbre of a single guitar track. Instead, it operates downstream as a unified, mode-dependent contouring and enhancement tool for stacked guitar parts. The three processing modes (*Rhythm*, *Lead*, and *Clean*) provide distinct production-oriented behaviours, each reflecting typical mix workflows for those guitar roles.

This document presents the architecture, design rationale, parameter mapping framework, and DSP modules that constitute *StringSauce*. The goal is to offer both a reproducible technical specification and a conceptual model for how semantic, macro-driven processing can be used effectively in real-world music production contexts.

2 System Overview

StringSauce is structured as a mode-aware, macro-driven bus processor for grouped guitar tracks. Its design emphasizes contouring and cohesion rather than instrument-level timbre creation. In a typical workflow, several guitar sources are routed to a shared bus per group. In this implementation, *StringSauce* applies unified spectral shaping, dynamic control, harmonic enhancement, and spatial processing.

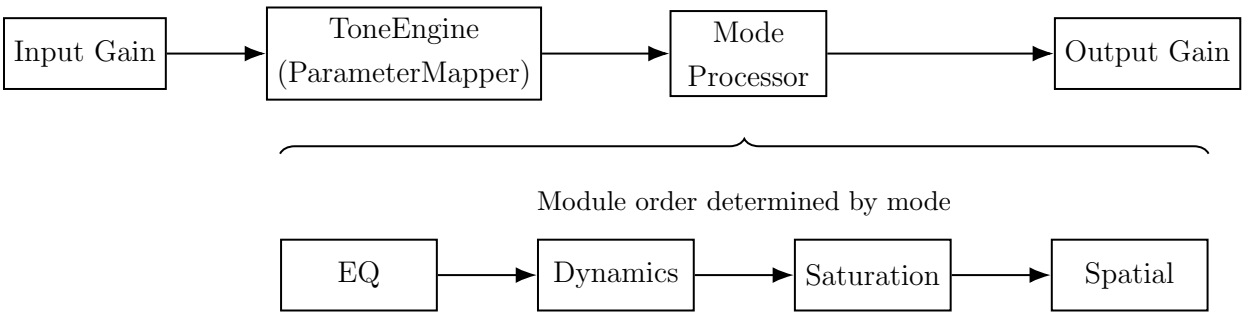


Figure 1: High-level architecture of the *StringSauce* processing pipeline.

Figure 1 depicts the primary components of the system. Audio enters through an input gain stage and is then passed to the `ToneEngine`, which consolidates macro control values, applies parameter smoothing, and generates module-level DSP parameters using the `ParameterMapper`. These parameters configure a mode-specific processing chain selected by the `ModeProcessor`. Each mode selects a different ordering of the four DSP modules:

- **Rhythm**: EQ → Dynamics → Saturation → Spatial
  Prioritizes tight low end, transient control, and cohesion for dense rhythm stacks.
- **Lead**: Saturation → EQ → Dynamics → Spatial
  Emphasizes harmonic richness, forward mids, and controlled sustain.
- **Clean**: EQ → Saturation → Spatial → Dynamics
  Supports shimmer, clarity, and natural ambience for clean textures.

Because the system is designed for bus processing, each module is tuned for broad spectral and dynamic gestures rather than surgical correction. The macro controls do not target isolated parameters; instead, they drive multi-parameter transformations that collectively shape the bus in a manner consistent with typical mixing practices. For example, adjusting

the *shimmer* parameter simultaneously affects high-shelf EQ, de-essing sensitivity, and post-saturation tone.

The output stage provides final gain staging, ensuring relatively stable loudness regardless of mode or macro settings. This architecture enables *StringSauce* to function as a single, cohesive processing layer capable of gluing multi-track guitars into a unified mix element.

3 ToneEngine Architecture

The ToneEngine is the central coordination layer that translates the plugin's macro controls into specific DSP parameters suitable for bus processing. Rather than performing any signal processing itself, the ToneEngine acts as a parameter aggregator: it gathers the macro control values, applies smoothing, computes mode-dependent behavioural adjustments, and outputs four module-level parameter blocks (EQ, Dynamics, Saturation, and Spatial). These parameter blocks describe the instantaneous configuration of the bus processor for the current audio block.

Macro Parameter Acquisition

At the beginning of each audio callback, the plugin reads the six macro controls from the `AudioProcessorValueTreeState`. All macro parameters are defined on the normalized interval $[0, 1]$, enabling consistent behaviour when automated or modified by the host DAW.[1]

Because the ToneEngine operates at the bus level, the goal is not to respond to micro transients or individual articulations, but to adjust broader spectral, dynamic, and spatial tendencies across a collective guitar source. As such, the mapping is intentionally smooth, stable, and resistant to rapid fluctuations.

Centring and Preprocessing

Before being mapped, each macro parameter is transformed into a centred representation $c \in [-1, 1]$. This centring stage establishes a meaningful neutral point ($c = 0$), corresponding to a musically-transparent baseline configuration that introduces minimal colouration to the guitar bus. Positive and negative deviations from this point encode perceptual shifts (brighter/darker, tighter/looser, more/less transient emphasis).

This symmetry is particularly useful for bus processing, where excessive sensitivity or asymmetrical behaviour can cause the entire group to feel unbalanced or unstable.

Mode-Aware Parameter Mapping

After preprocessing, the ToneEngine invokes the `ParameterMapper`. The mapper generates four parameter processing groups:

- EQParameters
- DynamicsParameters
- SaturationParameters
- SpatialParameters

Each group contains all of the numerical DSP parameters required by its corresponding module: filter frequencies, gains, Q values, compressor thresholds, saturation drive, modulation rates, reverb time, and so forth.

The mapping is explicitly mode dependent. The three operational modes (Rhythm, Lead, and Clean) apply different frequency anchors and dynamic tendencies, reflecting real-world mix practices:

- Rhythm mode favours low-end control, midrange punch, and tight transient consistency across double-tracked parts.
- Lead mode emphasises upper-mid clarity, controlled sustain, and harmonic richness without overwhelming the supporting layers.
- Clean mode encourages width, shimmer, and gentle modulation, allowing arpeggiated or picked textures to occupy space without masking denser parts.

Because the mapping expresses each macro as a multi-parameter gesture, the `ToneEngine` becomes the bridge between semantically-meaningful controls and reproducible, deterministic DSP transformations.

Smoothing and Temporal Stability

To ensure smooth transitions, especially when macro controls are automated, the `ToneEngine` maintains a set of short-time smoothing filters. Each smoothed parameter is updated toward its target value using a time constant of 20 ms.

This smoothing behaviour is particularly important in a bus context. Sudden jumps in EQ, dynamics, or saturation parameters can destabilize the stereo image or cause inconsistent transients between stacked guitars. The smoothing layer ensures that the bus responds musically and cohesively, even during rapid automation changes or intentional modulation.

Output Parameter State

At the end of the update step, the `ToneEngine` exposes a fully-populated `EngineParameters` struct. This struct is consumed by the `ModeProcessor`, which applies the per-module parameters in the appropriate order for the active mode.

Because the `ToneEngine` is responsible only for parameter flow, not audio processing, it remains lightweight, deterministic, and easy to extend. New DSP modules or additional macro controls can be integrated without altering the structure of the processing pipeline, preserving the separation of concerns between parameter semantics and the underlying DSP implementation.

`ToneEngine` serves as the semantic core of *StringSauce*, enabling expressive, grounded control of a complex and mode-dependent bus processing chain while maintaining the predictability and real-time safety required for professional audio work.

4 Parameter Mapping Framework

The Parameter Mapping Framework is responsible for translating the six semantic macro controls into concrete DSP parameters for the four processing modules. This is implemented via the `ParameterMapper` and forms the computational core of *StringSauce*'s design.

The mapping framework is organized around several design principles:

- **Semantic transparency**—macro control gestures should align with intuitive mixing

vocabulary (e.g., "tighten the lows," "add shimmer," "make it slap").

- **Mode sensitivity**—each mode adjusts the mapping to support typical production goals for Rhythm, Lead, and Clean guitar buses.
- **Neutral midpoint**—macro values near the center of the range should approximate transparent processing.
- **Multidimensional influence**—each macro affects several DSP domains simultaneously.
- **Real-time safety**—mappings must use strictly bounded, allocation-free arithmetic suitable for the audio thread.

4.1 Normalization and Centring

All macro controls in *StringSauce* are defined on a normalized interval $v \in [0, 1]$ to unify automation behaviour across DAWs. To facilitate symmetric, meaningful deviations from a neutral midpoint, each macro parameter is first converted to a centred representation:

$$c = 2(v - 0.5), \qquad c \in [-1, 1]. \tag{1}$$

The centered value $c = 0$ corresponds to a musically-transparent configuration, yielding:

- unity EQ gain,
- dynamics with ratio of ≈ 1,
- no saturation,
- no spatial processing.

Positive and negative values encode opposing musical gestures. For example:

- $c_{\text{thump}} > 0$: tighter, more controlled lows.
- $c_{\text{thump}} < 0$: fuller, looser low-end foundation.
- $c_{\text{shimmer}} > 0$: brighter highs, more sparkle.
- $c_{\text{body}} < 0$: leaner mids, less density.

This formulation allows the mapping to remain stable, predictable, and easily extendable.

4.2 EQ Mapping

The EQ mapping converts the four spectral macros (*thump*, *body*, *character*, and *shimmer*) into a set of filter frequencies, gains, and Q values for a six-band IIR equalizer:[2]

- high-pass filter
- low shelf
- two mid peaking filters
- high shelf
- air band

Across all modes, the EQ mapping aims to adjust the spectral contour of the bus as a whole. The macro definitions behave as follows:

- **Thump**—Shifts low-frequency contouring. Positive thump values raise the high-pass cutoff and tighten the lows, while negative values relax the cutoff and boost the low shelf for a fuller group sound. This provides control over low-end headroom in dense rhythm stacks.
- **Body**—Modulates low-mid energy and the shape of the second mid band. Higher body values promote midrange density and forwardness, which can help guitars occupy space in the mix without overpowering vocals or drums.
- **Character**—Adjusts the first mid band. Increasing character brightens and brings out articulation; decreasing it yields warmer, more subdued textures.
- **Shimmer**—Controls brightness and sense of "air." Positive shimmer boosts the high shelf and air band while subtly shifting their corner frequencies upward. Higher values help clean or lead parts remain distinguishable in a dense mix.

Each mode modifies the baseline frequencies and sensible gain ranges:

- **Rhythm**—lower mid anchors, reduced maximum air gain, higher high-pass flexibility.
- **Lead**—higher mid anchors, more presence emphasis, slightly extended top-end capability.
- **Clean**—increased air band range, gently wider Q values to avoid harshness.

These mode-specific EQ maps reflect common mix practices across guitar production.

4.3 Dynamics Mapping

The dynamics mapping transforms the dynamics macros (*thump*, *body*, *shimmer*, and *slap*) into the parameters for:

- a broadband compressor,
- a de-esser with band-limited detection,
- a transient shaping stage,
- a makeup gain stage.

The most influential macro is *slap*, which serves as the primary "glue" control for bus compression.

The control overview is as follows:

- **Slap**—Controls compressor threshold, ratio, and attack/release characteristics. Higher slap yields tighter, more cohesive guitar stacks with stronger transient articulation, making multiple rhythm tracks feel powerful and chunky.
- **Body**—Adjusts the dynamic sustain portion. Higher body values encourage longer sustain and thicker tails across the layered guitars.
- **Thump**—Influences attack timing and the compressor envelope response in the low-frequency region, preventing excessive pumping when guitars are aggressively low-end heavy.
- **Shimmer**: Controls de-esser sensitivity and frequency. Higher shimmer yields increased high-frequency detection, ensuring upper harmonics remain controlled even at elevated bus levels.

Mode-specific dynamic behaviour ensures:

- Rhythm buses receive tight, stable, mix-ready transient control.
- Lead buses emphasize sustain and articulation without overwhelming supporting elements.
- Clean buses maintain natural dynamics with subtle, transparent control.

4.4 Saturation Mapping

Saturation is controlled by *character*, *body*, and *shimmer*, with the selected operational mode determining which waveshaping curves are engaged. The saturation curves are tape, tube, transistor, and exciter. The principles behind these waveshapers follow standard nonlinear audio techniques.[3,4] Here, the mapping focuses on bus glue and harmonic density rather than distortion.

- **Character**—Determines both saturation type and intensity. Higher values of character increase drive and dry/wet balance, adding harmonic thickness across stacked guitars without altering their fundamental tone.
- **Body**—Controls low-mid saturation asymmetry. Introducing a small DC bias produces subtle harmonic shifts that alter perceived warmth on the bus.
- **Shimmer**—Modulates the tone filter following the waveshaper, allowing the harmonics to become brighter or darker depending on the spectral needs of the mix.

Mode biases:

- Rhythm—tape/tube flavours for transparent thickness.
- Lead—tube/exciter flavours for forward midrange and articulation.
- Clean—tape/exciter for clarity and sparkle.

4.5 Spatial Mapping

The spatial mapping uses *space* to configure delay, chorus, reverb, and stereo width block parameters. The spatial effects are calibrated to provide subtle mix integration. These effects rely on classical delay-line and feedback structures.[4,5]

- **Space**—Controls the blend and intensity of ambience. Low values keep the bus tight and forward; high values introduce sense of depth and space. All modes keep reverb times short enough to avoid masking rhythm precision.

Across modes:

- Rhythm mode minimises ambience and width.
- Lead mode introduces moderate depth and widening.
- Clean mode allows the widest stereo field and highest ambience.

5 DSP Modules

The four DSP modules collectively define the sonic behaviour of the *StringSauce* bus processor. Each module is designed to be real-time safe, mode-aware, and numerically stable. The following sections describe the internal design and mathematical foundations of each module.

Mathematical Foundations

All DSP operations used in *StringSauce* rely on standard discrete-time signal processing formulations. Biquad filters follow the classical second-order IIR difference equation.[2] Compressor behavior is modeled using soft-knee static curves and first-order envelope followers.[6] Nonlinear processing employs common waveshaping functions (e.g., $\tanh(x)$) and simple polynomial terms,[3] with oversampling applied to minimize aliasing.[7] Spatial effects draw from conventional delay-line and feedback structures.[4,5] No novel mathematical derivations are introduced; instead, these established foundations support the implementation of the macro-mapped bus processor described in this work.

5.1 Equalizer

The Equalizer module implements a six-band IIR topology using JUCE's `dsp::IIR::Filter`. Each band is represented by a second-order IIR (biquad) section of the standard form:[1,2,7]

$$y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] - a_1 y[n-1] - a_2 y[n-2], \tag{2}$$

where $x[n]$ is the input, $y[n]$ is the output, and $\{b_0, b_1, b_2, a_1, a_2\}$ are filter coefficients derived from user- and mode-dependent parameters. *StringSauce* uses six such filters:

1. High-pass filter

2. Low shelf
3. Mid-band 1
4. Mid-band 2
5. High shelf
6. Air-band shelf

All coefficients are generated using RBJ's Audio EQ Cookbook formulations,[2] which provide closed-form expressions for shelving and peaking filters based on a desired centre frequency f, quality factor Q, and linear gain G. Before computing the coefficients, these parameters are clamped to conservative bounds:

$$f \leftarrow \text{jlimit}(20,\ 0.45 f_s,\ f), \qquad Q \leftarrow \text{jlimit}(0.2,\ 4.0,\ Q), \qquad G \leftarrow \text{jlimit}(0.05,\ 8.0,\ G). \quad (3)$$

These bounds serve two purposes:

- **Frequency stability**—restricting f to $\leq 0.45 f_s$ prevents RBJ coefficients from becoming ill-conditioned near the Nyquist frequency, where numerical precision degrades.[7]
- **Reasonable bandwidth**—$Q < 0.2$ becomes so broad as to be unstable in shelving, while $Q > 4$ yields extremely narrow resonances inappropriate for bus processing.

The mapping functions translate the macro controls into sets of these filter parameters:

- **Thump** adjusts the HPF cutoff and low-shelf gain.
- **Body** shifts mid-band centre frequencies and bandwidth.
- **Character** sculpts the primary midrange contour.
- **Shimmer** boosts high-shelf and air-band frequency regions.

Because the equalizer operates at the bus level, gain ranges are intentionally moderate (typically within ±3–4 dB). This minimizes phase rotation and preserves coherence across layered guitars, especially when multiple instances of the processor are used in parallel.

For efficiency, the EQ stage bypasses itself entirely when all bands are effectively at unity:

$$|G_k - 1| < 10^{-3}, \quad \forall k, \tag{4}$$

preventing unnecessary processing and avoiding cumulative floating-point noise.

5.2 Dynamics Processor

The Dynamics module contains four subsystems:

1. A broadband compressor
2. A band-limited de-esser (HPF $\rightarrow$ LPF $\rightarrow$ RMS detector)
3. A transient shaper using dual envelope followers
4. Makeup gain compensation

Together, these stages act as a cohesive bus glue processor, stabilizing transients, and controlling perceived volume in the output.

Broadband Compression

The compressor applies a soft-knee static curve combined with a first-order attack/release envelope follower. Given an input signal measured in decibels, $x_{\text{dB}}[n]$, and a threshold T, the instantaneous linear gain is:[6]

$$g[n] = \begin{cases} 1, & x_{\text{dB}}[n] < T, \\ \left(\frac{x_{\text{dB}}[n]-T}{R}\right)_{\text{lin}}, & x_{\text{dB}}[n] \geq T, \end{cases} \tag{5}$$

where R is the compression ratio and $(\cdot)_{\text{lin}}$ denotes conversion back to linear gain. This static curve defines how much attenuation should occur for a given input level, independent of time dynamics.

To produce smooth time-varying behavior, the compressor uses a standard one-pole envelope follower:[7]

$$e[n] = \begin{cases} \alpha_a \, e[n-1] + (1-\alpha_a)|x[n]|, & |x[n]| > e[n-1], \\ \alpha_r \, e[n-1] + (1-\alpha_r)|x[n]|, & |x[n]| \leq e[n-1], \end{cases} \tag{6}$$

where the attack and release coefficients are:

$$\alpha = e^{-1/(tf_s)}. \tag{7}$$

Here t is either the attack or release time constant, and f_s is the sample rate. This formulation ensures fast response to increases in level and smoother, slower decay after transients subside.

Macro influence:

- **Slap** adjusts threshold. ratio, attack, and release. Higher values for *slap* slow the attack and quicken the release, allowing for attack to become extremely well-defined.
- **Body** lengthens release time to promote sustain.
- **Thump** slows the attack slightly, allowing low-frequency energy to pass before compression engages.

De-esser

The de-esser suppresses excessive upper-mid harmonics by feeding a band-limited version of the input into an RMS detector.[6] A simple two-stage filter isolates the sibilant band:

$$x_{\mathrm{HP}}(n) = \mathrm{HPF}(x[n], f_{\mathrm{HP}}), \tag{8}$$

$$x_{\mathrm{BL}}(n) = \mathrm{LPF}(x_{\mathrm{HP}}(n), f_{\mathrm{LP}}). \tag{9}$$

The RMS detector then estimates signal energy over a window of N samples:

$$\mathrm{RMS} = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} x_{\mathrm{BL}}^2[n]}. \tag{10}$$

This RMS value modulates de-essing gain reduction. The *shimmer* macro shifts the detection band upward and lowers the de-ess threshold, increasing sensitivity to high frequency build-up.

Transient Shaper

Transient shaping enhances or reduces attack and sustain by comparing two envelope followers—one "fast" and one "slow." Let $e_f[n]$ and $e_s[n]$ denote the fast and slow envelopes, respectively. Their difference provides a simple transient estimate:

$$\tau[n] = \text{jlimit}(-1, 1,\ e_f[n] - e_s[n]\ ). \tag{11}$$

When $e_f[n] > e_s[n]$, $\tau[n]$ is positive, indicating a transient attack. When $e_f[n] < e_s[n]$, the signal is in its sustain or decay phase.

Attack and sustain gains are then computed as:

$$g_{\text{atk}} = 1 + k_{\text{slap}} \cdot 2\tau[n], \qquad g_{\text{sus}} = 1 + k_{\text{body}} \cdot 0.5 e_s[n]. \tag{12}$$

Here, k_{slap} controls transient enhancement, while k_{body} emphasizes sustain.

Makeup Gain

Makeup gain compensates for level reduction introduced by compression:

$$g_{\text{makeup}} = 10^{G_{\text{dB}}/20}. \tag{13}$$

This avoids signal loss while maintaining headroom and preventing clipping, with limits applied to guarantee bus stability.[8]

5.3 Saturation Processor

The Saturation module applies controlled harmonic enhancement using a combination of oversampling, waveshaping, tone filtering, and dry/wet mixing. Its goal is not to introduce

heavy distortion, but to provide subtle harmonic thickening and nonlinear glue suitable for a guitar bus.

The processing pipeline consists of:

1. 2× oversampling using a polyphase half-band IIR filter,
2. nonlinear waveshaping,
3. post-shaper high-shelf tone filtering,
4. dry/wet mixing,
5. output-level compensation.

Oversampling

Nonlinearities generate harmonics above the Nyquist frequency, causing aliasing when processed at the host sample rate. To minimize this, the input signal is first upsampled:

$$x_{\uparrow}[n] = \sum_{k} h[k]\, x[n-k], \tag{14}$$

where $h[k]$ are the coefficients of the polyphase half-band IIR filter used for interpolation.[4,7]

Oversampling ensures that most harmonics generated by the nonlinear stage fall below the oversampled Nyquist frequency, allowing the subsequent downsampling filter to remove them cleanly.

Nonlinear Waveshaping

After oversampling, the signal is processed by a nonlinear transfer function $f()$:[3]

$$y[n] = f\left(g_{\text{drive}}\left(x[n] + b\right)\right), \tag{15}$$

where:

- g_{drive} controls the amount of input gain,

- b is a small DC bias (primarily modulated by the *body* macro) used to influence the odd/even harmonic balance,
- $f(x)$ is the selected waveshaping curve (tape, tube, transistor, or exciter).[4]

Biasing the input before a symmetric nonlinear function such as $\tanh(x)$ causes the output to contain a controllable mix of even and odd harmonics, providing a warmer or more aggressive character depending on the mode.

An example of the tube-style nonlinear curve used in *StringSauce* is:

$$f_{\text{tube}}(x) = \tanh(1.5x - 0.2x^3), \tag{16}$$

which combines soft saturation with a cubic term to introduce a slight hardening of the transfer curve at higher amplitudes.

Post-Shaper Tone Filtering

After waveshaping, a high-shelf or tilt-style filter shapes the spectral tilt of the harmonics. This prevents harshness and allows the *shimmer* macro to brighten or darken the saturation artifacts in a musically controlled way.[4]

Dry/Wet Mixing

The saturation output is blended with the unprocessed signal using JUCE's linear dry/wet rule:[1]

$$y[n] = (1 - w)\, x[n] + w\, y_{\text{sat}}[n], \tag{17}$$

where $w \in [0, 1]$ is the mix parameter. This ensures that small saturation amounts remain subtle and that extreme settings do not obscure the direct sound of the bus.

Output Compensation

The processed signal is compensated using a level-matching gain factor so that drive changes do not result in large perceived loudness jumps. This keeps the processor predictable and prevents downstream processors from being overdriven.

5.4 Spatial Processor

The Spatial Processor enhances the sense of depth, width, and movement in the bus using a combination of delay-based effects and mid/side manipulation. All components are derived from classical delay-line algorithms used in chorus, echo, and artificial reverb.[4,5]

The module integrates:

- stereo delay,
- chorus via sinusoidal modulation,
- a lightweight reverb,
- stereo width adjustment through mid/side processing.

These elements are tuned for subtle, mix-friendly behaviour, ensuring that spatial processing enhances the bus without masking articulation.

Delay

The delay line stores past samples and retrieves them after a specified time offset. Delay time in samples is computed as:[4]

$$d = f_s \cdot \frac{t_{\text{ms}}}{1000}, \tag{18}$$

where t_{ms} is the desired delay in milliseconds and f_s is the sample rate.

Feedback is introduced by feeding a scaled version of the previous delayed output back into the delay line:

$$x_{\text{fb}}[n] = x[n - d] \cdot k_{\text{fb}}, \tag{19}$$

where k_{fb} is the feedback coefficient, which controls echo density and decay time. Values of k_{fb} are restricted to keep the feedback loop stable.

Chorus

Chorus is implemented as a short delay line, where the delay time is continuously modulated by a low-frequency oscillator. The time-varying delay is:

$$d[n] = d_0 + D \sin\left(2\pi f_{\text{mod}} \frac{n}{f_s}\right), \tag{20}$$

where:

- d_0 is the base delay,
- D is the modulation depth,
- f_{mod} is the modulation rate,
- n is the sample index.

This modulation produces small, periodic pitch variations that generate a sense of width and movement.[4]

In *StringSauce*, the *space* macro increases both depth and rate, allowing clean and lead textures to become more spacious without losing clarity.

Reverb

Reverb is implemented using JUCE's `dsp::Reverb`, a tuned, feedback-delay-network structure suitable for real-time musical use.[1]

The relevant parameters operate within normalized ranges:

$$\text{roomsize} \in [0.1, 1.0], \qquad \text{damping} \in [0, 1]. \tag{21}$$

Within *StringSauce*:

- *space* increases room size and wetness,
- *shimmer* slightly reduces damping to create brighter tails,
- *body* influences low-frequency damping to maintain clarity.

The reverb is intentionally subtle, ensuring that rhythm buses remain tight while lead and clean modes gain depth and dimension.

Stereo Width

Stereo width adjustment is performed through mid/side (M/S) processing, which rotates the stereo signal into two orthogonal components:[9]

$$M = \frac{L + R}{\sqrt{2}}, \tag{22}$$

$$S = \frac{L - R}{\sqrt{2}}, \tag{23}$$

where L and R are the left and right input channels, M is the mid (center) component, and S is the side (difference) component.

Width control is achieved by scaling the side channel:

$$S' = w \cdot S,$$

where $w \in [0, 1]$ in Rhythm mode (narrower image) and $w > 1$ in Clean mode (wider image). The stereo signal is then reconstructed:

$$L' = \frac{M + S'}{\sqrt{2}}, \qquad R' = \frac{M - S'}{\sqrt{2}}.$$

This provides spatial enhancement while preserving mono compatibility and preventing excessive stereo spread that could destabilize the bus image.

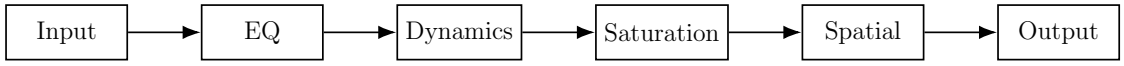
6 Mode-Dependent Routing

Although all four DSP modules are present in every configuration, the order in which they are applied has a significant impact on the resulting timbre, transient behaviour, and mix integration of the guitar bus. To accommodate the differing needs of rhythm stacks, leads, and clean textures, *StringSauce* implements three distinct routing configurations governed by the `ModeProcessor`.

Routing order is a musically consequential design choice—placing a module earlier or later in the chain determines which spectral and temporal features are preserved, which are emphasized, and which are suppressed.

The following subsections describe each routing configuration, its DSP and musical rationale, and the corresponding block diagrams.

6.1 Rhythm Mode



Rhythm mode is designed for dense, double- or quad-tracked guitar parts typical of rock, metal, and modern pop productions. These parts generally benefit from strong low-end control, consistent transient presentation, and tight stereo cohesion. To support this, the chain begins with equalization.

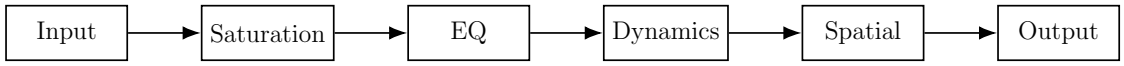
EQ First. Leading with EQ allows low-end management and midrange contouring to be applied before compression. High-pass filtering and low-shelf shaping reduce mud and preserve headroom, preventing the dynamics processor from responding too aggressively to uncontrolled low frequencies.

Dynamics Second. Placing the compressor after EQ enables it to operate on a cleaned-up signal. This reduces frequency-dependent pumping and allows *slap*-driven transient control to unify multiple rhythm layers.

Saturation Third. Moderate harmonic enhancement after dynamics introduces glue and thickness without compromising transient clarity. Since rhythm stacks can build up harsh midrange, placing saturation third ensures harmonics are generated from a stable, well-controlled signal.

Spatial Last. Spatial effects, if used at all, are subtle in rhythm mode. Keeping them last retains clarity and prevents ambience from muddying the timing precision essential for tight rhythm guitars.

6.2 Lead Mode



Lead mode is optimized for single-note melodies and solos that require articulation, sustain, and tonal focus. This mode emphasizes harmonic richness and presence while maintaining mix clarity.

Saturation First. Placing saturation first colors the raw tonal character of the lead bus, adding harmonic interest early in the chain. This ensures that downstream EQ and dynamics operate on a harmonically enriched signal, meaning the EQ shapes both $x[n]$ and its generated harmonics $f(x[n])$, yielding a more controlled and musical spectral emphasis.

EQ Second. With saturation introducing harmonic density, EQ is used to sculpt the post-saturation spectrum. Boosts in the upper midrange (1–3 kHz) are more effective and stable when applied after harmonic generation.

Dynamics Third. Compression after EQ stabilizes the lead performance, promoting sustain and control. Because the input to the compressor now includes EQ-enhanced harmonics, the release stage naturally reinforces the emphasized frequencies, supporting forwardness in the mix.

Spatial Last. Spatial processing is more pronounced in Lead mode than in Rhythm, but less than Clean mode. Placing it last supports a sense of depth without compromising

articulation.

6.3 Clean Mode

Figure 2: Clean mode: EQ → Saturation → Spatial → Dynamics.

Clean mode targets textures such as arpeggiated chords, plucked lines, and ambient guitar layers. These sounds often need width, shimmer, and gentle movement while avoiding harshness and maintaining transient delicacy.

EQ First. Starting with EQ reduces resonances and prepares the signal for the small, harmonically subtle saturation that follows. High shelf and air-band shaping from the *shimmer* macro can remove dullness before adding width or ambience.

Saturation Second. Saturation in clean mode is kept subtle and used primarily for gentle harmonic enrichment. Placing it before spatial processing allows the generated harmonics to diffuse naturally in the reverb and chorus tails.

Spatial Third. Spatial processing is the defining element of clean mode. Chorus and reverb are more active here than in the other modes, creating width and sparkle. Placing spatial effects before compression ensures that ambience retains a natural dynamic arc.

Dynamics Last. Placing the compressor last gently reins in the overall level, preventing spatial effects from becoming overly dynamic. This final glue stage helps clean guitars sit consistently in the mix without compromising on space.

7 Implementation Considerations

The design of StringSauce is constrained by the requirements of real-time audio processing. Unlike offline rendering systems, a plugin inserted on a bus must produce output deterministically

within strict time bounds, with no dynamic memory allocation, unpredictable branching, or blocking operations.[8] The following subsections outline the engineering considerations that shaped the implementation.

7.1 Real-Time Safety

StringSauce adheres to the standard real-time safety rules of modern audio DSP.[8] In particular:

- No memory allocation occurs during `processBlock()`.
- No STL containers or JUCE containers change capacity on the audio thread.
- No system calls, file accesses, mutex locking, or atomics with contention are used in the signal path.
- All parameter updates are performed using lock-free mechanisms provided by the `AudioProcessorValueTreeState`.[1]

During preparation, the plugin allocates all necessary buffers, constructs all DSP modules, and configures oversamplers.[8] After this point, the audio thread performs only arithmetic operations and pointer dereferencing, ensuring bounded runtime per block.

7.2 Parameter Smoothing

Because *StringSauce* is designed for bus processing, sudden parameter changes would be disruptive across a group of layered guitar tracks. To avoid abrupt spectral/dynamic shifts, each mapped DSP parameter is passed through an exponential smoothing filter of the form:[6,7]

$$p[n] = (1 - \alpha)\, p[n-1] + \alpha\, p_{\text{target}} \tag{24}$$

with time constants in the range of 10–30 ms.

This ensures:

- seamless automation
- controlled transitions between modes

- stable behaviour during parameter modulation
- natural dynamics on the bus.

Parameters that control highly sensitive processes use slightly longer smoothing times than those controlling insensitive parameters.

7.3 Oversampling and Nonlinear Processing

The Saturation module employs 2× oversampling to reduce aliasing caused by nonlinear waveshaping functions:[3,7]

$$y[n] = f\big(g_{\text{drive}}\,(x[n] + b)\big), \tag{25}$$

where $f()$ is a nonlinear transfer function (tape, tube, transistor, or exciter) and b is a DC bias.

Oversampling is implemented using JUCE's `dsp::Oversampling` class with a polyphase half-band IIR filter.[1,4] Host block sizes are assumed to be sufficiently small for oversampling overhead to remain manageable.

To minimize CPU load, oversampling is bypassed entirely when both:

$$\text{drive} \approx 0 \qquad \text{and} \qquad \text{mix} \approx 0. \tag{26}$$

This avoids unnecessary overhead when saturation contributes no audible effect.

7.4 Numerical Stability

Several precautions are taken to ensure numerical stability:

- All biquad coefficients are clamped to conservative bounds, preventing unstable poles near the unit circle.[2]
- Filter frequencies are limited to avoid Nyquist proximity issues:

$$f \leq 0.45 f_s \tag{27}$$

- Compressor envelopes are computed using stable first-order filters, avoiding denormal numbers by applying JUCE's denormal protection.[1]
- The spatial processor's delay lines use fixed-size buffers allocated during `prepareToPlay()`, avoiding wrap-around instability.

Together, these measures guarantee that the plugin remains numerically predictable under fast automation, extreme macro values, or atypical input signals.

7.5 Mode Switching

Switching between the Rhythm, Lead, and Clean modes is implemented in a way such that it does not introduce discontinuities in the audio stream. When the mode changes:

1. The `ToneEngine` generates new target parameters for each module.
2. Smoothing ramps these parameters over time.
3. The `ModeProcessor` swaps to the new module ordering.

Because the ordering change itself cannot be smoothed (as a module is either before or after another), the plugin ensures that:

- each module receives parameters compatible with its new position,
- transient buildup is minimized via envelope continuity,
- spatial processors reset modulation phases only when needed.

This allows mid-performance or automated mode changes with minimal artifacts.

7.6 CPU and Memory Efficiency

StringSauce is intended to run on a guitar bus, where CPU resources may be constrained by amp simulators, IR loaders, and reverbs already present in the session. Efficiency considerations include:

- saturation and spatial blocks are bypassed when their mix levels are effectively zero.
- EQ calculations are precomputed and updated only when a parameter changes.[8]
- buffer allocations occur only during `prepareToPlay()`.
- mid/side width processing reuses stack-allocated buffers.
- all filters and DSP components reuse memory between blocks.[8]

Empirically, the processor maintains real-time performance at 48 kHz with typical DAW block sizes (64–256 samples).

7.7 JUCE Integration and Plugin Lifecycle

StringSauce follows JUCE's standard plugin lifecycle:[1]

- `prepareToPlay()`: allocate buffers, initialize modules, construct oversamplers, compute initial parameters.
- `processBlock()`: retrieve smoothed macro parameters, update per-module parameters, process the audio through the mode-specific chain.
- `releaseResources()`: free large buffers used by spatial processors.

Interaction between the GUI and audio thread is handled by `AudioProcessorValueTreeState`, ensuring thread-safe parameter communication without locks or atomics on the audio thread.[1]

7.8 Perceptual Coherence Across Tracks

Finally, because *StringSauce* processes groups of guitar tracks rather than individual tracks, temporal and spectral coherence is essential. The implementation therefore stresses:

- smooth transitions across macro controls
- minimal latency variation between modules, which is important for phase alignment
- low-level noise consistency

- consistent, deterministic behaviour across channels.

These considerations ensure that a multi-track guitar bus sounds like a single, unified musical element rather than a collection of individually processed tracks.

8 Evaluation

Evaluating a bus processor presents challenges distinct from those encountered in traditional DSP research. While certain module-level behaviours can be measured objectively (filter responses, compressor curves, harmonic distortion, are all clear examples), the overall effectiveness of a bus processor is inherently perceptual and strongly dependent on musical context. For this reason, the evaluation of *StringSauce* combines methodological descriptions of objective measurements with a structured listening-based assessment.

The following subsections outline the evaluation methodology used in this project. Quantitative measurements are described in a way that supports future extension, while perceptual evaluations reflect the current state of the system.

8.1 Objective Evaluation Methodology

Several components of *StringSauce*'s processing chain can be evaluated through standard DSP analysis techniques.[7] While full quantitative results are left for future work, the following methodology outlines how each subsystem may be assessed in future work in an objective manner.

Frequency Response Analysis

The tonal behaviour of the EQ and post-saturation tone filters can be examined using magnitude and phase response plots. Each EQ band is implemented as a biquad filter with transfer function $H_k(z)$. Because the six filters are arranged in series, the overall transfer function of the EQ is:

$$H(z) = \prod_{k=1}^{6} H_k(z). \tag{28}$$

To obtain the discrete-time frequency response, the transfer function is evaluated on the unit circle of the z-plane. This is done by substituting:

$$z = e^{j\omega},$$

where ω is the normalized angular frequency in the interval $0 \leq \omega \leq \pi$.

The resulting frequency response is

$$H(e^{j\omega}),$$

where j denotes the imaginary unit ($j^2 = -1$), following the DSP convention of using j rather than i to avoid confusion with electrical current. From this expression, both the magnitude response $|H(e^{j\omega})|$ and the phase response $\arg(H(e^{j\omega}))$ can be plotted. These plots would reveal how the EQ and post-saturation tone filters shape the spectrum under different modes and macro parameter settings.[2,9]

This would allow inspection of:

- low-frequency tightening or loosening via *thump*
- midrange shaping from *body* and *character*
- high-frequency tilt from *shimmer*
- mode-dependent spectral tendencies across Rhythm, Lead, and Clean

Dynamic Behaviour and Gain Reduction Curves

Compressor behaviour can be assessed through static gain reduction curves, which plot output level y_{dB} as a function of input level x_{dB}.[6] The gain reduction characteristic is:

$$G_{\text{GR}}(x_{\text{dB}}) = x_{\text{dB}} - y_{\text{dB}}. \tag{29}$$

These curves would verify whether the *slap* macro produces the expected changes in threshold and ratio.

Transient behaviour can be evaluated by measuring envelope responses to standard test signals such as impulses and step transitions, following classical envelope-follower analysis methods.[7] Such measurements validate:

- the smoothness of parameter interpolation
- the intended effect of *slap* on attack characteristics
- release-time shaping introduced by *body*
- differences in dynamic behaviour across Rhythm, Lead, and Clean modes

Harmonic Distortion and Aliasing

Nonlinear processing in the saturation module can be characterized using single-tone harmonic sweeps and multi-tone intermodulation distortion (IMD) tests. These are standard tools for analyzing waveshapers and other nonlinear systems.[3,4]

FFT-based spectral analysis would reveal:

- the harmonic structure produced by each saturation type
- the reduction of aliasing afforded by the 2× oversampling stage
- bias-controlled odd/even harmonic balance influenced by the *body* macro

Although numerical results are not included here, the framework formalizes how processor modules may be evaluated in future iterations of the system.

CPU Load and Real-Time Performance

Real-time performance can be measured under representative DAW operating conditions, following standard plugin benchmarking practices.[8] Measurements should include:

- multiple sample rates (44.1–96 kHz)
- host buffer sizes from 64–256 samples

- oversampling enabled and disabled
- different macro control configurations and modes

The goal is to ensure that the processor remains low-latency, allocation-free, and real-time safe even in worst-case scenarios.

8.2 Perceptual Evaluation

Because *StringSauce* is explicitly designed as a musical tool, perceptual evaluation is central to assessing its effectiveness. At the current stage, evaluation has been performed primarily through structured critical listening using a variety of musical materials.

Listening Procedure

A set of representative guitar arrangements was collected:

- double-tracked distorted rhythm guitars
- single-note and harmonized lead lines
- layered clean arpeggios and sustained ambient chords
- sparse acoustic textures

For each arrangement, the following listening procedure was used:

1. Evaluate the dry bus as a baseline
2. Enable *StringSauce* in each mode with neutral macro settings
3. Adjust each macro control from –100% to +100% while listening for:
    - spectral movement
    - transient response
    - harmonic density
    - stereo width

- perception of cohesion
- changes in depth and ambience

4. Compare mode behaviours on identical audio material
5. Compare *StringSauce* processing to manually assembled reference chains (EQ + compression + saturation + reverb)

This procedure helps assess whether:

- the macro controls produce musically intuitive changes
- the modes exhibit the expected stylistic tendencies
- the processor enhances coherence across stacked guitars
- spatial processing remains and mix-friendly
- saturation behaves harmonically without unpleasant artifacts

Preliminary Listening Observations

While a full perceptual study is reserved for future work, early observations indicate that:

- Rhythm mode provides strong low-end tightening and transient glue, with subtle saturation adding body without smearing articulation.
- Lead mode produces a forward upper-midrange and smooth harmonic sustain, particularly when *character* and *slap* are driven positively.
- Clean mode delivers a controlled, wide, and shimmer-rich ambience, especially when combining small saturation amounts with moderate spatial processing.
- macro controls interact coherently, preserving the intended semantics even when adjusted simultaneously.

These findings suggest that the parameter mapping and module ordering successfully capture auditory tendencies relevant to guitar bus processing.

8.3 Future Evaluation Work

Several avenues for deeper evaluation remain open:

- quantitative spectral plots for each macro and mode
- dynamic step-response analysis
- aliasing measurements with high-order FFTs
- CPU benchmarking across multiple hardware configurations
- formal listening tests involving multiple participants
- comparative analysis with commercial guitar bus processors

These will be incorporated into future revisions of the system once the core DSP modules have fully stabilized.

9 Conclusion

This document has presented the design, architecture, and implementation of *StringSauce*, a semantic, macro-driven guitar bus processor built in modern C++ using the JUCE framework. The system integrates four DSP modules (equalization, dynamics, saturation, and spatial effects) under a unified parameter-mapping framework and a mode-dependent routing topology. Rather than acting as a traditional guitar tone tool, *StringSauce* treats the guitar bus as a single entity and shapes its behaviour through a set of musically-meaningful macro controls.

At the technical level, the project demonstrates real-time DSP implementation, oversampled nonlinear processing, parameter smoothing, numerically-stable filtering, and modular, mode-aware routing. More importantly, at the conceptual level, the system serves as a testbed for a broader research question: how can high-level semantic descriptors of sound, using real words or even a random assortment of syllables, be translated into deterministic, multi-parameter DSP transformations? This semantic-to-DSP mapping is motivated by the premise that musicians and engineers naturally describe sound using intuitive, perceptually-grounded vocabulary, and that DSP tools can be made more accessible by honouring those descriptors directly.

The goal of this project is therefore not to simply continue developing *StringSauce* as a standalone processor, but to use it as an entry point into a sustained research trajectory focused on semantic interfaces and meaningful DSP design. It is the opinion of the the author that exploring this idea has direct implications for audio pedagogy. Many learners encounter barriers when first engaging with audio engineering because the conceptual vocabulary within DSP can feel disconnected from their lived musical intuition. By exploring structured mappings between semantic descriptors and deterministic processing behaviours, this work aims to contribute to a future in which educational DSP tools can introduce signal processing concepts through the language musicians already use.

Such systems could serve as pedagogical bridges, helping learners build mental models of timbre, dynamics, and spatialisation from intuition, rather than rigor. This research direction aligns with the broader goal of designing audio technologies that are not only technically robust, but also cognitively approachable and educationally motivated.

StringSauce is therefore best understood not as a final product. Rather, it is a launchpad. It is a concrete exploration of how perceptually meaningful language can drive deterministic signal-processing behaviour. It establishes a foundation for graduate-level research in semantic audio, DSP interfaces, and the pedagogical design of audio technology. The work supports the author's broader intention to pursue graduate study in Computer Science and Music Technology, with a focus on improving how audio concepts are taught, understood, and translated into expressive technical tools.

References

[1] JUCE Development Team. *JUCE Framework Documentation*, 2025.

[2] Robert Bristow-Johnson. Cookbook formulae for audio equalizer biquad filter coefficients. `https://webaudio.github.io/Audio-EQ-Cookbook/audio-eq-cookbook.html`, 2019. Technical report.

[3] Julius O. Smith. Nonlinear waveshaping methods, 2010. Online book chapter.

[4] Udo Zölzer, editor. *DAFX: Digital Audio Effects*. Wiley, 2 edition, 2011.

[5] Julius O. Smith. Physical audio signal processing, 2010. Online book.

[6] Joshua D. Reiss, Michael Massberg, and Dimitrios Giannoulis. Digital dynamic range compressor design: A tutorial and analysis. *Journal of the Audio Engineering Society*, 60(6):399–408, 2012.

[7] Julius O. Smith. Introduction to digital filters with audio applications, 2007. Online book.

[8] Will C. Pirkle. *Designing Audio Effect Plugins in C++*. Taylor & Francis, 2019.

[9] Joshua D. Reiss and Andrew P. McPherson. *Audio Effects: Theory, Implementation and Application*. CRC Press, 2014.