ECE397 Independent Study

Project Overview & Results

Granular Synthesizer

Paul Jablonski

pjj3@illinois.edu

April 21st, 2025

Table of Contents

1. Introduction

1.1 Problem Statement

Real-time audio manipulation remains a challenging task due to the computational demands of processing large streams of data with minimal latency. Granular synthesis, a technique that breaks sound into small "grains" and reassembles them to create new textures, requires precise control over timing, pitch, grain overlap, and windowing to achieve high-quality results. The problem lies in achieving this control in a way that is both low-latency and scalable, particularly for real-time musical performance or sound design environments.

1.2 Proposed Solution

To address this issue, this project proposed the development of an FPGA Granular Synthesizer: a physical hub designed to perform real-time granular synthesis using dedicated hardware. The initial plan was to leverage the parallel processing capabilities of an FPGA to handle intensive audio operations such as real-time grain slicing, envelope application, pitch shifting, and overlap handling with minimal latency. The envisioned system would allow for direct user interaction via button input and could interface with other audio hardware via standard protocols and the audio codec. By offloading computation to an FPGA, the goal was to create a highly responsive, low-latency instrument capable of dynamic sound manipulation for live performance and recording alike.

1.3 Implemented Solution

During development, it became apparent that the complexity of designing and debugging FPGA-based audio systems, especially in the context of real-time control and iteration, posed significant challenges. Limitations in documentation, difficulties in prototyping flexible control interfaces, and time constraints led to a reevaluation of the implementation strategy.

As a result, the project pivoted to a software-based implementation in the form of a VST plugin via the JUCE C++ framework [2]. This plugin was developed to simulate many of the originally intended FPGA functionalities - including buffer-based grain slicing, pitch/density/time manipulation, and grain randomization - within a digital audio workstation (DAW) environment. While the plugin could not fully replicate the ultra-low latency or standalone nature of the FPGA system, it offered broader accessibility, potential for embedded use, and a user-friendly interface. Ultimately, this shift enabled a deeper focus and study on the musical capabilities of granular synthesis without the overhead of hardware design.

2. Literature Review

2.1 Source Material

This project draws exclusively on the foundational concepts presented by Curtis Roads in his 1988 paper *Introduction to Granular Synthesis*. Roads defines granular synthesis as the construction of complex sounds from thousands of short sonic grains, each ranging from 1 to 50 milliseconds in duration. He describes the technique as a variant of additive synthesis, where grains, each with an individual waveform, amplitude envelope, duration, and density, combine to form larger audio structures [1].

Roads discusses both the theory and early digital implementations of granular synthesis. He outlines a model where grains are organized into higher-level "events" defined by twelve parameters, including frequency, amplitude, density, and their respective slopes. These events can be represented as shapes on a time-frequency graph, allowing for the structured evolution of sound over time. He also emphasizes the role of randomness and variation in grain attributes, which is essential for creating rich, evolving textures [1]. Importantly, Roads highlights the musical potential of real-time granular manipulation, even in the context of limited computational resources.

2.2 Application of References

Roads' paper directly influenced the design and implementation of this project. The plugin replicates core features he outlines, including grain slicing, pitch control, time manipulation, and density control. Each grain is generated with defined parameters - timing, pitch, and envelope - and these are manipulated dynamically to simulate the event-based structures Roads describes.

The concept of grain density and its role in shaping the texture of sound was particularly influential in the inclusion of a density control slider. Similarly, the randomness discussed by Roads was implemented as a spread/randomization feature to introduce variation in grain timing and pitch, contributing to more organic textures. The wet/dry mix control was inspired by Roads' emphasis on blending granular output with other sonic material to create compound or hybrid sounds.

Ultimately, Roads' work provided both the theoretical basis and structural framework for the implemented features, guiding decisions on what parameters to expose and how to organize them in a way that reflects the flexibility of granular synthesis.

3. Technical Description

3.1 Granular Synthesis Overview

Granular synthesis in this plugin is achieved by decomposing the incoming DAW audio stream into short audio grains. These grains are generated continuously based on a configurable interval, manipulated in pitch and timing, passed through an amplitude envelope, and then reassembled to create new audio textures. The real-time nature of the implementation demands high performance, handled within a multichannel audio buffer.

Each grain is represented internally by a structure holding attributes such as start position, length, pitch, level, pan, and playback progress. These attributes are updated in real time, and grains are rendered with interpolation and a Hann window to avoid clicks and artifacts. The output is a mix of all currently active grains, blended with the dry signal according to the user-defined dry/wet mix.

3.2 Parameter Design and Control

Each user-facing parameter is linked to the DSP backend through a JUCE AudioProcessorValueTreeState, allowing real-time changes and automation support. Here's a breakdown of each implemented parameter:

Grain Size (grainSize): Defines the duration per grain in seconds. Ranges from 10ms to 500ms. Affects both the texture and responsiveness of the sound. Internally determines the number of samples per grain.

Pitch Shift (pitchShift): Scales the playback rate of each grain, affecting pitch. Ranges from 0.5× (one octave down) to 2.0× (one octave up). Applied as a multiplier to the grain's playback step, affecting interpolation and duration.

Randomization (randomization): Introduces randomness to the start position of each grain. Internally, it shifts the buffer read location by a random offset up to ±1 second, depending on the parameter value.

Dry/Wet Mix (dryWet): Controls the balance between original and processed audio. A value of 1.0 outputs only the granulated signal, while 0.0 outputs only the original. Applied post-processing via a linear crossfade.

Density (density): Controls the overlap and spacing of grains. A higher density leads to a more continuous or smear-like texture. Internally, it influences the interval between grain triggers by scaling the base grain size.

Time Stretch (timeStretch): Allows stretching or compressing playback speed without pitch change. It functions by either repeating grains in sequence to stretch playback, or cutting grains out to compress it.

All parameters are updated dynamically via the parameterChanged callback, ensuring seamless interaction between UI and DSP logic. This will be discussed later in the *Functions Breakdown* section.

3.3 DSP Pipeline

The plugin's real-time audio processing is managed within the processBlock method. The pipeline proceeds through the following stages for each audio buffer passed into the processor:

1. Input Buffering

Incoming audio is copied into a circular buffer (inputBuffer) of a fixed size of 2 seconds. This enables grains to be sourced from a recent history of audio samples and supports randomized grain start positions.

2. Grain Triggering

Grains are triggered/created at a calculated interval based on both grainSize and density, adjusted by timeStretch. This ensures appropriate spacing between grains and controls the texture's sparsity. Each grain is initialized in triggerGrain(), with parameters such as: Length (based on grain size), Start Position (with optional random offset), Pitch Shift, and Stereo Panning (randomized between 0.3 and 0.7).

3. Grain Playback

For every active grain, the playback position is incremented by 1 per sample. Then, the corresponding sample is retrieved from the input buffer using linear interpolation to support fractional positions. A Hann window is applied to the grain's amplitude envelope to avoid discontinuities. Following that, each grain's signal is panned between stereo channels and scaled by gain and pitch, where the result is summed into the output buffer.

4. Dry/Wet Mixing

A copy of the original (dry) signal is preserved. After all grains are summed, the processed (wet) buffer is blended with the dry buffer according to the dryWet parameter. This provides users with control over how much of the granular effect is present in the final output.

3.4 DSP Block Diagram

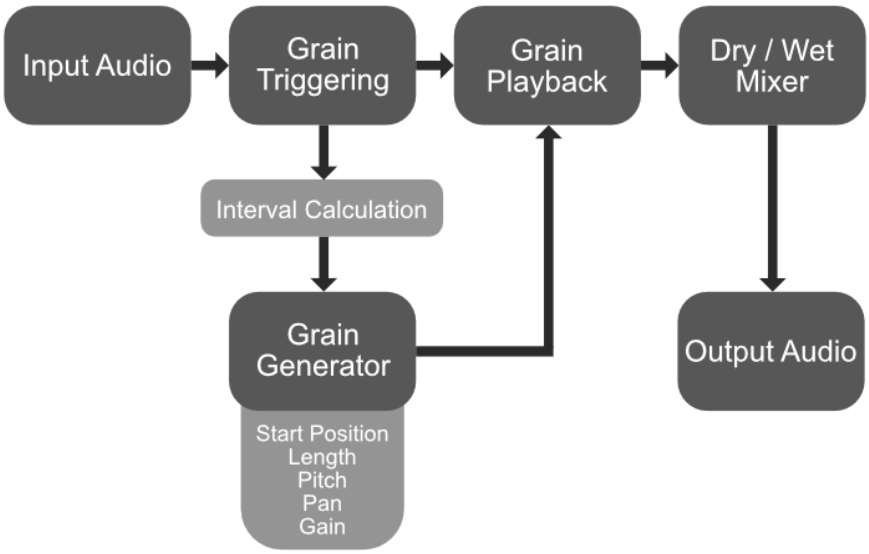

Figure 1. DSP pipeline step-by-step diagram.

3.5 GUI Design

The graphical user interface of the granular synthesizer plugin was built using JUCE and designed with a focus on minimalism, and intuitive control. The overall layout is within a compact 700 x 200 pixel window, divided into a top section for core synthesis parameters and a bottom section for the dry/wet mix control. All of these parameters are automatable within a DAW environment as well, meaning that they may be automatically adjusted over time through a track. This, alongside the GUI itself, is displayed below:

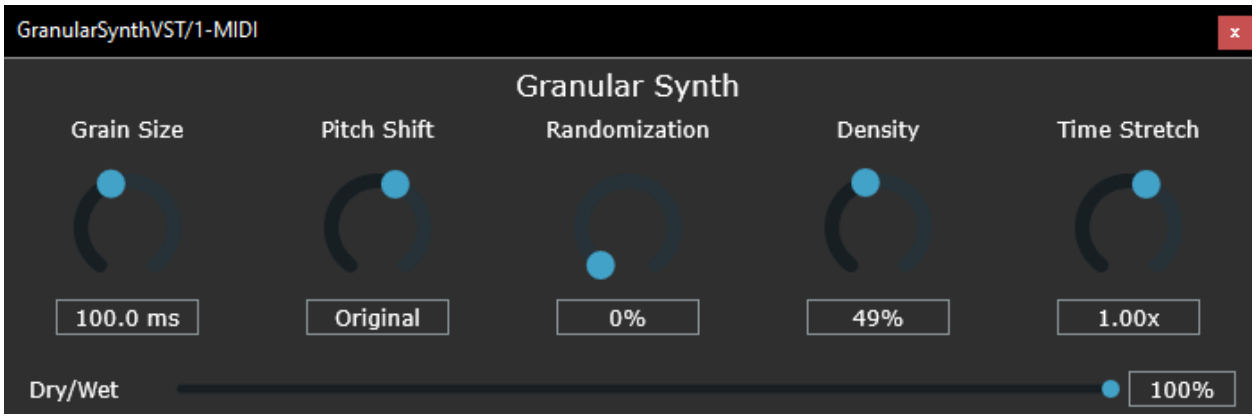

Figure 2. Graphic user interface for the plugin, featuring all controllable parameters.

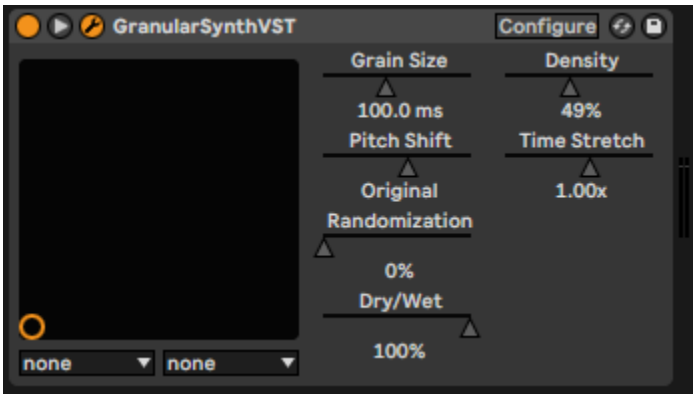

Figure 3. Ableton Live DAW compatibility with parameter control and automation.

4. Final Results

4.1 Results Overview

The final outcome of this project was a fully functional VST plugin that performs real-time granular synthesis on incoming audio. While the original goal was to implement this system in FPGA hardware, the pivot to a software approach allowed for a more flexible and musically usable implementation. The plugin supports control of grain size, pitch shift, density, time stretch, randomization, and a dry/wet mix, all of which are accessible through a compact, intuitive GUI. Within a DAW, the plugin enables users to experiment with texture-based sound design, from subtle time smearing to pitch-scattered effects.

The real-time responsiveness of parameter changes and the high-quality output achieved within the software environment affirm the plugin's success as a creative and educational tool for exploring granular synthesis. Although it does not replicate the low-latency, standalone behavior of an FPGA-based device, the plugin preserves the conceptual architecture originally envisioned and delivers practical capabilities.

4.2 Testing Performed

Testing focused on validating the functionality and musical quality of the granular synthesis plugin in a digital audio workstation environment. Using Ableton Live, three parallel audio tracks were used to capture a comparative visual analysis of the processing stages: a raw instrumental sample, a frozen and flattened version (to verify rendering), and a final version with the granular synthesis plugin applied.

As shown in the screenshot, the top track contains the unprocessed input audio. The middle track is the DAW's rendering of the same audio without any plugin processing, used as a control to ensure baseline behavior. The bottom track clearly illustrates the effects of granular processing: the waveform is visibly broken into more fragmented segments, reflecting time manipulation, pitch variation, and grain spread.

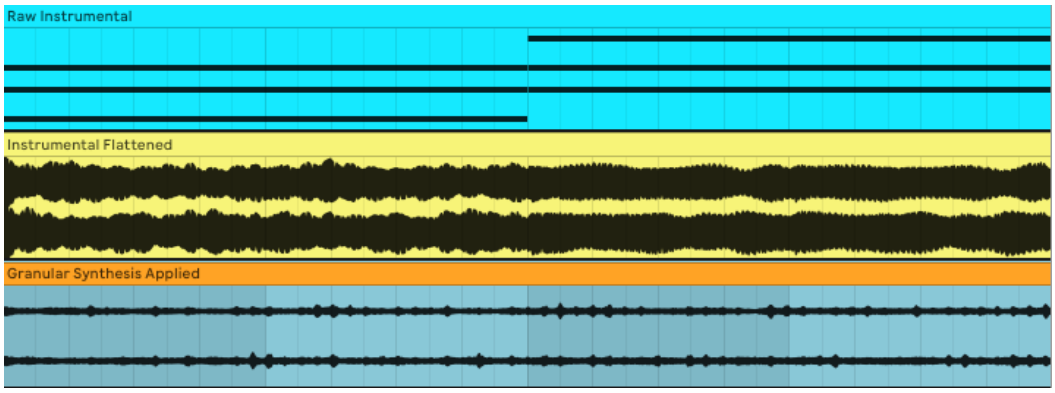

Figure 4. Three comparative and visual tracks, best accompanied via demo video (Linked Here).

While the visual differences are helpful for identifying structural changes, granular synthesis is inherently a textural effect, and much of its impact is best understood through listening. To address this, a demo video included on the GitHub repository presents real-time plugin usage within Ableton, showcasing how changes to parameters like pitch, grain density, and playback affect the sound. Together, the waveforms and demo video provide a visual and auditory validation of the plugin's granular synthesis capabilities.

4.3 Challenges Encountered

Several notable challenges arose during the course of this project. Initially, the hardware implementation using the DE10-Standard FPGA board proved more time-intensive and less flexible than anticipated. Issues such as a steep debugging curve and sparse documentation for audio IP cores made iteration difficult, particularly when testing or attempting to implement audio parameters. As development progressed, it became clear that a hardware prototype would not allow for sufficient exploration of granular synthesis behavior within the project timeline, prompting the pivot to a VST plugin architecture.

Transitioning to JUCE and software DSP presented its own set of hurdles. Implementing real-time grain triggering required careful synchronization between parameter updates and audio buffer processing. Early versions suffered from timing drift and improper overlap handling, which led to audible artifacts such as clicks and inconsistent playback. Grain pitch shifting also presented a challenge due to aliasing and boundary conditions when reading from the circular input buffer. These issues were resolved by refining the interpolation logic and adding window functions such as the Hann envelope to smooth each grain's onset and offset.

Despite these obstacles, each challenge contributed valuable insight into the technical and creative demands of real-time audio development, and the final result stands as a strong foundation for future refinement or hardware revisitation.

5. Future Suggestions

5.1 Modifications

While the current version of the plugin functions effectively and covers essential granular synthesis features, several improvements could enhance performance, usability, and sound quality. One area for modification is the envelope function applied to grains. Currently, a Hann window is used for smoothing grain transitions, which works well but is static. Future versions could implement user-selectable window shapes (e.g. Gaussian, Tukey, or Blackman-Harris) to give users finer control over the character and attack/decay behavior of grains.

Another important modification would be to implement real time parameter smoothing. At present, abrupt parameter changes, particularly to pitch or grain size, can cause minor audio artifacts. Smoothing these values over a short buffer could create more musical transitions and reduce artifacts in live performance.

Finally, the user interface, while compact and informative, could be improved by organizing controls into collapsible sections or tabbed panels to accommodate potential future parameters. Introducing visual feedback, such as a waveform display or grain activity monitor, could also make the plugin more intuitive and visually engaging for sound designers.

5.2 Further Extensions

In addition to new features at the plugin level, a promising future direction for this project is porting the granular synthesis engine to an embedded hardware platform. While the initial FPGA-based approach was paused in favor of software, the C++ implementation developed for the plugin lays a strong foundation for revisiting a hardware-based deployment, particularly on embedded platforms that support real-time audio processing. This doesn’t necessarily have to be an FPGA either.

Because the DSP engine is written in C++, with no dependence on JUCE-specific classes in the core grain processing logic (aside from JUCE's buffer and parameter abstractions), it can be modularized and adapted for use in embedded systems such as ARM processors, Teensy boards, or even Raspberry Pi. Functions like triggerGrain(), getInterpolatedSample(), and windowFunction() are self-contained and would require minimal changes. The primary requirement would be to replace JUCE’s AudioBuffer and Random utilities with platform-appropriate equivalents, and to interface with embedded I/O systems dependent on the choice of hardware.

6. Software & Hardware Documentation

6.1 File Responsibilities

This project is organized across four core source files, each responsible for some aspects of the plugin:

PluginProcessor.h and PluginProcessor.cpp define the audio processing logic of the plugin. The header declares all parameters, internal buffers, grain management structures, and utility functions. The corresponding implementation handles parameter updates, audio block processing, grain scheduling, interpolation, and windowing. Together, these files form the audio engine of the plugin.

PluginEditor.h and PluginEditor.cpp define the graphical user interface. They manage the layout, creation, and styling of all visible components, including the sliders for grain size, pitch shift, density, randomization, time stretch, and dry/wet mix. They also connect GUI controls to internal parameter state using JUCE's AudioProcessorValueTreeState system.

6.2 Functions Breakdown

Functions in the plugin are based around three areas: parameter management, grain processing, and I/O.

createParameters(): Initializes all user-controllable parameters and defines their ranges, default values, and display text. This function sets up the plugin's parameter tree and enables communication between the GUI and DSP logic.

prepareToPlay(): Called when the plugin is initialized or sample rate changes. It sets up the internal circular buffer, grain storage, and resets the scheduling state for real-time audio processing.

processBlock(): The main audio processing loop. It handles input buffering, grain triggering, sample interpolation, grain mixing, and dry/wet blending. It is called for each block of audio and is the heart of the DSP pipeline.

triggerGrain(): Allocates and configures a new grain based on the current buffer position and parameter state. This function handles randomization, pitch, pan, and envelope setup.

windowFunction(): Implements a Hann window to apply amplitude smoothing to each grain, reducing audio artifacts caused by sudden onsets or offsets.

getInterpolatedSample(): Performs linear interpolation between audio samples in the buffer, allowing fractional playback positions for pitch shifting.

parameterChanged(): Responds to real-time updates from the GUI, keeping internal variables in sync with the user-controlled parameters.

createEditor() / resized() / paint(): Create and arrange GUI elements, draw the plugin's visual components, and handle window resizing.

References

[1] C. Roads, "Introduction to Granular Synthesis," *Computer Music Journal*, vol. 12, no. 2, pp. 11–13, 1988. [Online]. Available: https://www.jstor.org/stable/3679937

[2] JUCE, "JUCE: The C++ framework for audio plugins, applications, and more," JUCE.com. [Online]. Available: https://juce.com