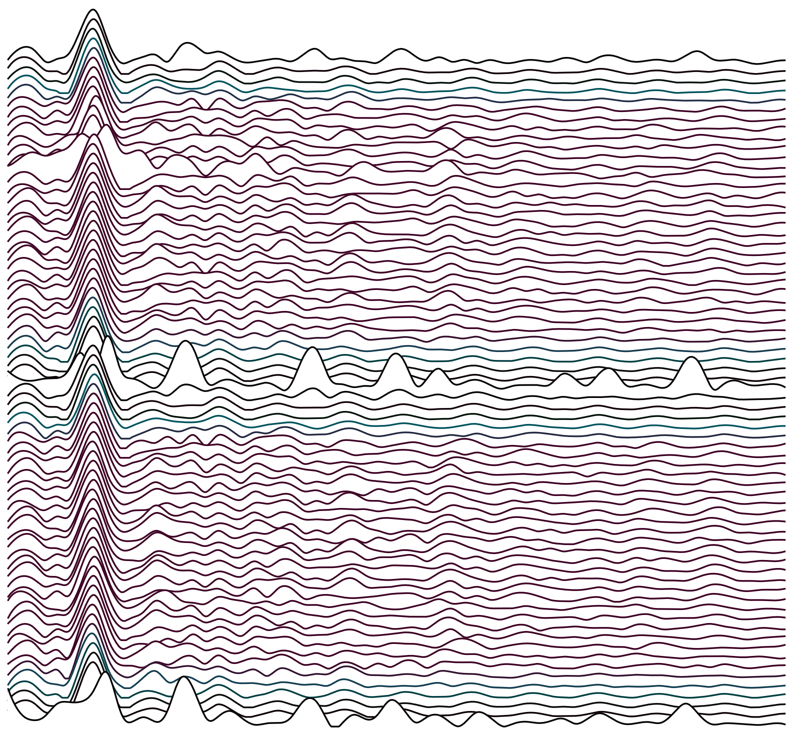
From Python to Silicon

A Technical Report on
Logistics, Power, Analog Synthesis, and Signal Analysis

Jackson Ferguson
University of Victoria
Astrophysics



Title graphic generated by the author (Python); source code in the accompanying repository: https://github.com/JacksonFergusonDev/systems-audio-lab.

February 5, 2026

Abstract

This document records a recursive engineering journey: the design and fabrication of the tools required to build a specific device. Identifying a gap between software architecture proficiency and physical implementation, I undertook a project to build an analog guitar pedal (The Red Llama). However, the standard hobbyist workflow proved insufficient for rigorous engineering.

This technical report details the development of four interdependent systems: (1) **Star Ground**, a deterministic Python logistics engine for BOM management; (2) A thermal-managed **Linear Power Supply**; (3) The **Red Llama** analog overdrive circuit; and (4) A custom **RP2040 Data Acquisition System** (DAQ) used to spectrally validate the final build.

Contents

Chapter 1

Introduction

1.1 The Systems Engineering Imperative

In modern computational astrophysics, complex systems are modeled with high precision, yet they remain fundamentally virtual. As an undergraduate specializing in data pipelines and Python, I operate within deterministic environments where logic is absolute and variables are strictly controlled. However, the transition from software architecture to physical implementation reveals a critical discontinuity—the "Full-Stack Gap."

In the physical domain, engineering is not merely about logic; it is about managing entropy. Physical systems are subject to stochastic variables—component tolerances, thermal drift, supply chain volatility, and electromagnetic interference—that do not exist in code. This technical report documents a recursive engineering initiative designed to bridge this gap. The primary objective was to fabricate a specific analog device (The Red Llama Overdrive), but the broader pedagogical goal was to apply the rigor of software systems engineering—modularity, regression testing, and version control—to the chaotic reality of hardware fabrication.

1.2 Problem Formulation

The standard hobbyist workflow for electronics fabrication is often characterized by ad-hoc solutions and subjective validation. When analyzed through a systems engineering lens, this approach reveals three critical failure modes that compromise reproducibility and rigorous analysis:

1. **Logistical Entropy:** Unlike software dependencies, which can be resolved instantly via package managers, hardware dependencies are physical and finite. Manual Bill of Materials (BOM) management is nondeterministic and error-prone; a single missing resistor constitutes a critical blocking dependency that can halt a project for weeks.

2. **Infrastructure Debt:** Precision audio circuits utilize high-gain amplification stages that are highly susceptible to power supply noise. Standard consumer-grade Switch Mode Power Supplies (SMPS) inherently introduce high-frequency switching ripple [1]. Relying on generic power infrastructure introduces an uncontrolled variable that aliases into the signal path, invalidating high-fidelity measurements.

3. **Subjective Validation:** In audio electronics, verification is often purely qualitative ("it sounds good"). However, from a physics perspective, "tone" is quantifiable as harmonic distortion and frequency response. Without calibrated instrumentation to visualize the transfer function, the engineering process remains blind, relying on trial-and-error rather than empirical data.

1.3 Methodology: The Recursive Solution

To resolve these deficiencies, I elected to engineer the tooling required to build the device, rather than purchasing off-the-shelf solutions. This ”recursive” approach ensures that every variable in the signal chain—from the procurement of the silicon to the analysis of the output—is understood and controlled.

This document details the design, fabrication, and integration of four interdependent systems:

- **Star Ground (Logistics):** A Python-based logistics engine that treats hardware BOMs as strict data objects. By parsing PDF documentation into structured JSON data, the system eliminates human error from the procurement process, effectively acting as a ”CI/CD pipeline” for physical inventory.
- **Linear Power Regulator (Infrastructure):** A custom, thermal-managed 12V-to-9V linear supply. By utilizing the L7800 series architecture, this subsystem establishes a clean, ripple-free noise floor, ensuring that artifacts observed in the output are generated by the device, not the wall mains.
- **Red Llama Overdrive (Device Under Test):** The primary artifact. This analog circuit utilizes CD4049 CMOS Hex Inverters, originally designed for digital logic, biased into their linear amplification region. This topology is hypothesized to generate soft-clipping saturation characteristics similar to vacuum tube triodes.
- **RP2040 DAQ (Instrumentation):** A custom spectrophotometer and high-speed Analog Front End (AFE). Calibrated against the 60Hz mains frequency, this system enables the empirical verification of the DUT’s harmonic content, moving validation from the subjective to the quantitative.

1.4 Report Structure

The remainder of this document is organized logically following the signal path of the engineering process:

- **Chapter 2** details the software architecture used to manage physical chaos.
- **Chapter 3** outlines the electrical infrastructure required to power the experiment.
- **Chapter 4** describes the fabrication and topology of the analog circuit itself.
- **Chapter 5** explains the design of the digital instrumentation used for analysis.
- **Chapter 6** presents the empirical data, synthesizing the four systems to validate the initial engineering hypotheses.

Chapter 2

The Logistics Engine (Software)

2.1 Problem Definition: Logistical Entropy

Before a circuit can be soldered, components must be procured. In the domain of DIY electronics, this is a deceptively complex problem. Bill of Materials (BOM) data is distributed across incompatible formats (PDFs, raw text, forum posts), and inventory management is typically manual.

When managing parts for four concurrent builds, the probability of error approaches unity. A single missing 10-cent resistor creates a shipping bottleneck that halts a project for weeks. I determined that manual transcription of BOMs was an unacceptable source of nondeterministic error.[2]

2.2 System Architecture: Star Ground

Star Ground is a Python-based full-stack logistics engine designed to function as the Single Source of Truth for component inventory. In circuit design, a "Star Ground" is the point where all signal paths converge to eliminate noise.[3] In this context, the software eliminates the "noise" of disorganized supply chains.

2.2.1 Design Philosophy: Determinism over AI

A seemingly obvious solution would be to pass PDF documentation to a Large Language Model (LLM) for parsing. However, for procurement, hallucination is a critical failure mode.[4] If an LLM misidentifies a 100kΩ resistor as 10kΩ, the hardware will fail.

Therefore, I rejected probabilistic models in favor of a **Deterministic Regex Engine**. The system uses a Hybrid Strategy for PDF ingestion:

1. **Visual Layout Analysis:** Using `pdfplumber` to extract table structures based on grid lines.[5]
2. **Regex Fallback:** A pattern-matching layer that parses raw text when table extraction fails.
3. **Snapshot Testing:** We use snapshot-based regression tests that compare the parser's current output against stored "golden master" JSON snapshots to detect unintended changes and preserve legacy support. [6, 7]

2.3 Algorithmic Logic: Nerd Economics

The core value of the engine lies in its heuristic inventory buffering. In hardware prototyping, the cost of downtime exceeds the cost of inventory holding. I implemented a “Yield Management Algorithm” that adjusts purchase quantities based on component risk profiles.

The engine calculates the Net Need:

$$\text{Net Need} = \max(0, \text{BOM Qty} - \text{Stock Qty}) \tag{2.1}$$

Safety buffers are applied strictly to the deficit:

- **High Risk / Low Cost (Resistors):** +10 unit buffer. They are fragile, cheap, and easily lost.
- **Critical Silicon (ICs):** +1 unit buffer. Ensures a backup in case of thermal damage during soldering.
- **High Cost (Potentiometers/Switches):** +0 buffer.

```python
def calculate_buffer(component_type: str, cost: float) -> int:
    if component_type == "RESISTOR":
        return 10  # Cheap insurance
    elif component_type == "IC" or component_type == "TRANSISTOR":
        return 1   # Protection against ESD/Heat damage
    elif cost > 2.00:
        return 0   # JIT economics
    return 0
```

Listing 2.1: Heuristic Buffering Logic

2.4 The “Silicon Sommelier”

Beyond simple counting, the engine acts as a domain expert system. It utilizes a lookup table to suggest component upgrades based on audio engineering best practices:

- **Op-Amps:** Detects generic `TL072` chips and suggests Hi-Fi alternatives like `OPA2134` for lower noise floors.[8, 9]
- **Fuzz Logic:** Automatically detects Germanium transistor topologies and flags “Positive Ground” warnings.[10, 11]
- **Texture Generation:** Maps clipping diodes to their sonic equivalents based on Forward Voltage (V_f).[12]

2.5 Generated Artifacts

The final output of the pipeline is not just a shopping list, but a structured build guide. The engine generates “Field Manuals”—PDFs that reorganize the assembly order by component Z-height (Resistors → Sockets → Capacitors). This enforces mechanical stability during soldering, ensuring that low-profile components are not obstructed by taller ones.

2.6 Validation

This software was successfully used to procure and manage the inventory for the Red Llama build (Chapter 4) and the Power Regulator (Chapter 3), resulting in zero missing parts and zero excess waste of high-value components.

Chapter 3

Infrastructure: Linear Power Supply

3.1 Project Motivation

With the logistics pipeline (Chapter 2) operational, attention turned to the physical infrastructure required to support the Device Under Test (DUT). Audio circuits are notoriously sensitive to power supply noise. Standard consumer-grade "wall warts" are typically Switch Mode Power Supplies (SMPS), which operate by rapidly switching a transistor on and off to maintain voltage. This switching frequency often leaks into the audio path, manifesting as high-frequency whine or aliasing artifacts.[1]

Furthermore, I possessed a surplus of 12V power bricks, but the standard for guitar pedals is 9V Center-Negative. Rather than purchasing a dedicated supply, I elected to engineer a **Linear Voltage Regulator**. This served a dual purpose: providing a clean, ripple-free power source for the DUT and serving as a low-risk "Hello World" for discrete fabrication before attempting the audio circuit.

3.2 Theory of Operation

The circuit relies on the **L7809CV**, a classic series linear regulator.[13] Unlike switching regulators, a linear regulator operates as a variable resistor, effectively "burning off" excess voltage as waste heat to maintain a constant output.[13] While less efficient than SMPS, the output is practically free of high-frequency ripple, making it ideal for audio applications.

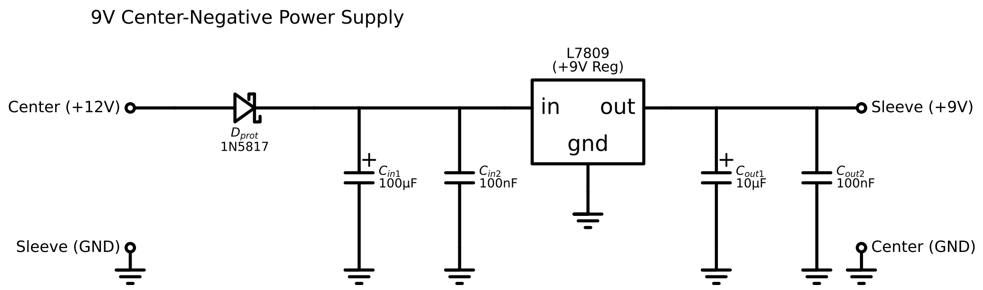


Figure 3.1: **Linear Regulator Schematic.** The circuit utilizes a standard L7809 topology with added input/output bulk capacitance and a reverse-bias protection diode (D_{prot}) to prevent damage from negative voltage transients.

3.3 Thermal Management Strategy

The primary engineering constraint in linear regulation is heat dissipation. The power dissipated (P_d) is a function of the voltage drop and load current:

$$P_d = (V_{in} - V_{out}) \times I_{load} \tag{3.1}$$

Given an input of 12V and a target output of 9V, the regulator drops 3V across the junction.

- **Scenario A (Red Llama):** The analog DUT draws negligible current ($<$ 10mA), resulting in $P_d \approx$ 0.03W.
- **Scenario B (Future Proofing):** Digital pedals (reverbs/delays) can draw upwards of 400mA.

For Scenario B, the dissipation becomes significant:

$$P_{max} \approx 3\text{V} \times 0.4\text{A} = 1.2\text{W} \tag{3.2}$$

A standard TO-220 package has a thermal resistance (θ_{JA}) of $\approx$ 65°C/W.[13] Without a heatsink, a 1.2W load would result in a temperature rise of:

$$\Delta T = 1.2\text{W} \times 65^\circ\text{C/W} = 78^\circ\text{C} \tag{3.3}$$

Assuming an ambient temperature of 25°C, the silicon junction would reach 103°C. While within the absolute maximum (125°C), this is dangerously hot for an enclosed ABS plastic chassis.[13]

3.3.1 The Solution

To mitigate this, I implemented a two-stage thermal solution:

1. **Conduction:** A bolt-on aluminum heatsink was attached to the TO-220 package to lower the thermal resistance.
2. **Convection:** Ventilation shafts were drilled into the top of the enclosure to establish a convection current, preventing the ABS box from acting as a thermal insulator.

3.4 Component Selection

The Bill of Materials, generated via Star Ground, included specific selections for noise filtering and protection:

- **Input Filtering:** A 100μF electrolytic capacitor handles bulk low-frequency ripple from the 12V source, while a parallel 100nF polyester capacitor filters high-frequency transients.[13]
- **Reverse Polarity Protection:** A **1N5817 Schottky Diode** was selected over a standard 1N4001 silicon diode.[14, 15] The Schottky's lower forward voltage drop ($V_f \approx$ 0.45V) preserves more of the input voltage headroom compared to silicon ($V_f \approx$ 0.7V).[14, 15]

Des.	Value	Component	SKU	Price	Qty	Total	Notes
U1	L7809	L7809CV-DG Reg 9V 1.5A	A-1597	\$0.25	1	\$0.25	TO-220
D_{prot}	1N5817	Schottky Diode 1A 20V	A-159	\$0.06	1	\$0.06	Low V_f
C_{in1}	100μF	35V Radial Electrolytic	A-987	\$0.03	1	\$0.03	Bulk In
C_{out1}	10μF	50V Radial Electrolytic	A-4554	\$0.02	1	\$0.02	Bulk Out
$C_{f1,f2}$	100nF	100V 5% Poly Box	A-564	\$0.10	2	\$0.20	HF Filter
J1	2.1mm	DC Power Jack (Fem)	A-2237	\$0.13	1	\$0.13	Panel Mnt
P1	2.1mm	DC Plug (Male) Pigtail	A-6806	\$0.15	1	\$0.15	Output
–	Box	Black Plastic Box 03	A-2383	\$2.50	1	\$2.50	ABS
–	Heatsink	TO-220 10-Fin 1”	A-1512	\$0.24	1	\$0.24	For U1
–	M3	Hex Socket Cap Screw	A-6379	\$0.10	1	\$0.10	Mounting
					PROJECT TOTAL	**\$3.68**	

Table 3.1: **Bill of Materials.** All components sourced from Tayda Electronics to minimize shipping overhead. Prices reflect unit costs at time of purchase.

Chapter 4

Device Under Test: Red Llama Overdrive

4.1 Circuit Topology

With the logistics managed (Chapter 2) and the power cleaner (Chapter 3), fabrication began on the primary objective: The **Red Llama Overdrive**.

This circuit is topologically distinct among guitar pedals. While most overdrives utilize Operational Amplifiers (Op-Amps) with diode clipping in the feedback loop, the Red Llama utilizes the **CD4049 CMOS Hex Inverter**.[16]

Originally designed for digital logic (translating 1s and 0s), the CD4049 consists of internal push-pull MOSFET pairs.[16] By applying negative feedback, the digital gates are forced into a linear amplification region. When driven hard by a guitar signal, the MOSFETs hit the power supply rails, creating a soft, rounded saturation curve that mimics the behavior of vacuum tubes, eventually squaring off into a hard fuzz at maximum gain.

4.2 Procurement & Fabrication

The build utilized the output artifacts from the **Star Ground** engine. The "Field Manual" PDF dictated the assembly order based on Z-height, ensuring that low-profile resistors were soldered before the taller IC sockets and capacitors. This procedural rigor minimized the mechanical frustration often associated with high-density Perfboard layouts.

4.3 Engineering Deviation: The Diode Mod

While the build largely followed the standard Way Huge Electronics schematic, I introduced a modification to the power section to maximize dynamic range.

The reference design calls for a **1N4001** silicon diode at position D2 for reverse-polarity protection.[17] As noted in Chapter 3, silicon diodes exhibit a standard forward voltage drop of $\approx 0.7\text{V}$.[15] In a 9V circuit, this drop is non-trivial:

$$V_{rail} = V_{source} - V_{diode} = 9.0\text{V} - 0.7\text{V} = 8.3\text{V} \tag{4.1}$$

The CD4049's headroom is strictly limited by the supply rails. Losing 0.7V reduces the clean headroom before clipping occurs. To recover this, I substituted the 1N4001 with a **1N5817 Schottky Diode**.[14]

$$V_{rail(mod)} = 9.0\text{V} - 0.3\text{V} = 8.7\text{V} \tag{4.2}$$

By recovering ≈ 0.4V of supply voltage, the CMOS inverters have slightly more headroom. This translates to increased dynamic range and a cleaner transient response before the onset of saturation.

4.4 The Validation Gap

Upon completion, the device passed the "Smoke Test" and produced sound. Subjectively, the tone was rich and harmonically complex. However, as a scientist, subjective listening is insufficient. I required empirical verification of the saturation characteristics.

I did not own an oscilloscope. Rather than buying one, I realized that the surplus **RP2040** microcontroller I had procured (via the Star Ground logistics run) possessed the theoretical capability to act as a high-speed digitizer.[18] This led to the development of the custom DAQ system detailed in the following chapter.

Chapter 5

Instrumentation: RP2040 DAQ

5.1 System Architecture

To validate the saturation behavior of the Red Llama (Chapter 4), I required a Data Acquisition (DAQ) system capable of capturing audio frequencies with high fidelity. The **RP2040** microcontroller (dual-core ARM Cortex-M0+) was selected as the engine due to its flexible I/O and low cost.[19]

The system operates on a **"Store-and-Forward"** architecture. Unlike typical streaming applications that push data continuously (and suffer from jitter due to USB packetization), this system prioritizes sample rate stability. It captures a precise burst of data into a pre-allocated ring buffer in RAM, then transmits the binary blob over USB Serial only *after* acquisition is complete.[19] This decouples the strict timing requirements of the sampling loop from the non-deterministic latency of the host workstation.

5.2 Analog Front End (AFE) Theory

The RP2040's internal ADC is limited to a 0 V to 3.3 V unipolar range.[19] Direct connection to the Red Llama (which outputs high-amplitude bipolar AC signals) would result in negative voltage clipping and potential silicon latch-up. A "Universal" Signal Conditioning Interface was designed to bridge this gap.

5.2.1 Input Protection I

A series resistor $R_{prot} = 10\,\mathrm{k\Omega}$ is placed immediately at the input jack. This limits the current flowing into the subsequent clamping stage during high-voltage transients or accidental connection to modular synth rails (±12 V), preventing thermal destruction of the protection diodes.

5.2.2 High-Pass Filter (AC Coupling)

For audio signals, the DC component is blocked using a polyester film capacitor $C_{ac} = 220\,\mathrm{nF}$. The corner frequency ($f_c$) is determined by the input impedance of the subsequent stage. In **High-Z / Guitar Mode**, the input impedance is dominated by the bias injection resistor $R_{inject} = 220\,\mathrm{k\Omega}$.

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 220\,\mathrm{k\Omega} \cdot 220\,\mathrm{nF}} \approx 3.29\,\mathrm{Hz} \tag{5.1}$$

This provides a flat frequency response well below the audible floor (20 Hz).

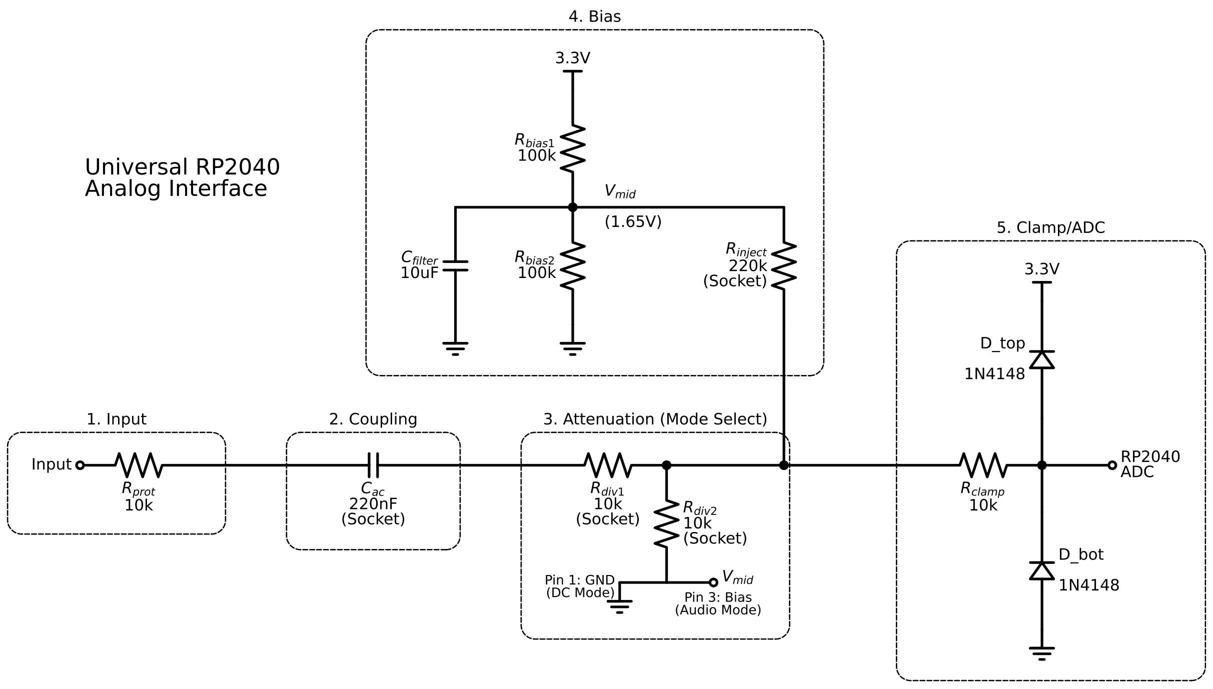


Figure 5.1: **System Schematic.** The signal path flows left-to-right: Protection Stage → AC Coupling → Attenuation/Bias Network → ADC Input.

5.2.3 Bias & Switched Reference Topology

The RP2040 ADC requires a signal centered at 1.65 V (half-rail) to read AC waveforms.[19] A stiff voltage divider (100 kΩ pair) filtered by a 10 μF electrolytic capacitor generates a stable quiet rail (V_{mid}).

A **Switched Reference Topology** is used to solve impedance conflicts via a physical jumper (JP1):

- **Audio Mode:** The jumper connects the bottom of the attenuation network to V_{mid} (1.65 V). The signal ”sees” virtual ground, preserving the gain ratio while biasing the signal center point.
- **DC Mode:** The jumper connects to GND, forming a standard resistive divider for unipolar sensing ($0 - 5$ V sensors).

5.2.4 Clamping & Safety Analysis (Silicon vs. Schottky)

A critical design decision was the selection of **1N4148 Silicon Diodes** for rail clamping over standard Schottky diodes.

Standard low-voltage protection often utilizes Schottky diodes (e.g., BAT54) due to their low forward voltage drop ($V_f \approx 0.3$ V), which keeps signal excursions strictly within the datasheet absolute maximums (V_{DD}+0.5 V).[20] However, Schottky diodes exhibit significant reverse leakage current ($I_R \approx 1$–$5\ \mu$A).[20] Given the High-Z nature of this AFE (specifically the 220 kΩ bias injection), a leakage current of just 1 μA would introduce a DC offset error of $V_{err} = 220$ mV, significantly reducing dynamic range.

By selecting **1N4148 Silicon Diodes**, we prioritize precision ($I_R < 25$ nA).[21] The trade-off is a higher forward voltage ($V_f \approx 0.7$ V).[21] During an over-voltage event, the clamping node reaches:

$$V_{clamp} = V_{DD} + V_f = 3.3\,\text{V} + 0.7\,\text{V} = 4.0\,\text{V} \tag{5.2}$$

This exceeds the RP2040 absolute maximum rating of 3.8 V.[19] However, the safety of the device is guaranteed by the current-limiting resistor R_{clamp} (10 kΩ). If the external signal forces the node to 4.0 V, the internal ESD diodes of the ADC will begin to conduct at approx 3.8 V. The resulting current injected into the pin is:

$$I_{inj} = \frac{V_{clamp} - V_{internal}}{R_{clamp}} = \frac{4.0\,\text{V} - 3.8\,\text{V}}{10\,\text{k}\Omega} = 20\,\mu\text{A} \tag{5.3}$$

This injection current of 20 μA is extremely small, and the input is protected primarily by current limiting.

5.3 Hardware Configuration

The board functionality is determined by the presence of socketed components and the position of the 3-pin jumper. This allows the same physical board to serve as a Guitar Interface, Line Level Input, or Sensor Reader.

Mode	Application	Jumper (JP1)	R_{div1}/R_{div2}	R_{inject}
Audio (Std)	Synths, Pedals	Pin 2-3 (V_{mid})	10k / 10k	*Empty*
Guitar (High-Z)	Passive Inst.	Pin 2-3 (V_{mid})	Wire / *Empty*	220k
DC / Sensor	0-5V Signals	Pin 1-2 (GND)	10k / 10k	*Empty*

Table 5.1: Jumper Settings and Component Population Guide

5.4 Firmware Logic

The firmware is written in MicroPython to leverage rapid development, but standard Python is too slow for high-speed sampling. To maximize throughput, the critical capture loop uses the `@micropython.native` decorator.[22]

```python
@micropython.native
def capture_burst(adc_obj, buf, size: int):
    # Compiles to ARM Thumb machine code
    for i in range(size):
        buf[i] = adc_obj.read_u16()
```

Listing 5.1: Native Emitter Capture Loop

This instructs the compiler to emit ARM Thumb machine code directly.[22] While the method lookup still incurs overhead, the loop mechanics (increment, compare, jump) become native CPU instructions. This optimization allowed the system to achieve a stable sample rate of ≈ 100 kSps, far exceeding the initial 40 kSps target.

5.5 Calibration: The "Mains Hum" Reference

Accurate frequency domain analysis requires a precise knowledge of the sampling rate (F_s). While the RP2040 hardware clock is stable, the firmware architecture relies on a MicroPython polling loop running on the CPU rather than a DMA-driven hardware timer. Consequently, the effective sample rate is determined by the interpreter's execution overhead, which creates a deterministic but unknown latency.

To resolve this without external signal generators, the system was calibrated using a ubiquitous, high-stability reference standard: the North American Power Grid (60.00 Hz).

5.5.1 Physics of Environmental Coupling

The calibration signal was acquired by the user touching the input probe tip. This technique exploits the physics of **capacitive coupling**. The physical system can be modeled as a capacitive voltage divider:

1. **Source (V_{grid}):** The 120 V_{rms} wiring in the walls creates an oscillating electric field ($\omega = 2\pi \cdot 60$).
2. **Coupling Impedance (Z_c):** The user's body acts as a conductive plate separated from the wiring by a dielectric (air/insulation). This forms a stray capacitance ($C_{stray} \approx 10 - 100\,\mathrm{pF}$).
3. **Load Impedance (Z_{in}):** The AFE input resistance (approx. 220 kΩ in High-Z mode).

The voltage appearing at the ADC (V_{adc}) is determined by the ratio of the input impedance to the total path impedance. Given that the reactance of a 50 pF gap at 60 Hz is extremely high ($\approx$ 53 MΩ), the circuit forms a high-impedance attenuator. However, because the source potential is high ($\approx 170\,\mathrm{V}_{pk}$), even this tiny coupling ratio yields a measurable signal in the millivolt range.

5.5.2 Derivation of the Correction Factor

With the 60 Hz signal captured, an FFT was performed using the theoretical sample rate ($F_{assumed}$) derived from the loop delay code. The peak analysis revealed a significant discrepancy:

- **Theoretical Peak:** 60.0 Hz
- **Observed Peak:** 49.9 Hz

This shift indicates that the real-world time steps (Δt) were larger than calculated (the loop ran slower). Since the frequency axis of an FFT is linear with respect to F_s, we can derive a dimensionless scalar correction factor (α):

$$\alpha = \frac{f_{ref}}{f_{meas}} = \frac{60.0}{49.9} \approx 1.2024 \tag{5.4}$$

Applying this scalar to the assumed sample rate yields the empirically calibrated rate:

$$F_{real} = F_{assumed} \times \alpha \approx 97,812\,\mathrm{Hz} \tag{5.5}$$

This "Golden Constant" ($F_s \approx 97.8$ kSps) was hardcoded into the analysis pipeline. Subsequent verification against known audio sources confirmed that this single-point calibration effectively linearized the frequency domain mapping across the full bandwidth.

5.6 Final Technical Specifications

The following specifications reflect the measured performance of the commissioned device.

Parameter	Value / Note
ADC Resolution	12-bit (Scaled to 16-bit integer)
Sample Rate (F_s)	97.8 kHz ± 0.5% (Calibrated)
Bandwidth	DC – 48.9 kHz (Nyquist Limit)
Buffer Depth	16,384 Samples (≈ 167 ms duration)
Spectral Resolution	≈ 5.9 Hz per FFT bin
Input Range	0–3.3 V (Unipolar) / ±3.3 V (AC Coupled)
Noise Floor	< 50 mV (RMS System); 353 mV (Env. Hum)

Table 5.2: Measured System Performance

Chapter 6

Analysis & Validation

6.1 Experimental Methodology

To characterize the Red Llama Overdrive, the device was inserted into the signal chain between a reference source (Electric Guitar, Neck Pickup) and the RP2040 DAQ. The data acquisition pipeline utilized the system's "Science Mode," executing a burst capture of $N = 16,384$ samples at a calibrated rate of $F_s \approx 97.8\,\text{kSps}$.

The analysis pipeline was implemented in Python using numpy and `scipy.signal`.[23, 24] Crucially, to mitigate spectral leakage caused by the finite sampling interval, a **Hann Window** function ($w(n) = 0.5 - 0.5\cos(\frac{2\pi n}{N-1})$) was applied to the raw time-domain buffer prior to the Fast Fourier Transform (FFT).[25, 26] This preprocessing step ensures that the side-lobes of high-amplitude fundamental frequencies do not obscure the lower-amplitude harmonic distortion products.[27]

6.2 DAQ Sensitivity & Noise Floor

Before characterizing the Device Under Test (DUT), the instrumentation itself was validated to quantify the available dynamic range. Figure 6.1 illustrates the spectral density of the system with the input grounded.

The system exhibits a read noise RMS of $\approx 1.3\text{mV}$. By comparing this noise floor against the saturation limit of the signal chain ($\approx 170\text{mV}_{rms}$ before clipping), we can calculate the effective Signal-to-Noise Ratio (SNR):

$$\text{SNR}_{dB} = 20 \cdot \log_{10}\left(\frac{V_{signal}}{V_{noise}}\right) = 20 \cdot \log_{10}(129.6) \approx 42.25\,\text{dB} \tag{6.1}$$

This confirms a dynamic range of > 42 dB. Furthermore, the spectrum is remarkably free of the 60Hz mains hum spikes that typically dominate amateur builds, validating the thermal and electrical isolation provided by the **Linear Power Regulator**.

6.3 Gain Structure & Transfer Characteristic

The Red Llama operates as a high-gain pre-amplifier. To visualize the amplification factor, the input signal ($\approx 50\text{mV}$) was plotted against the output on a unified voltage scale (Figure 6.2).

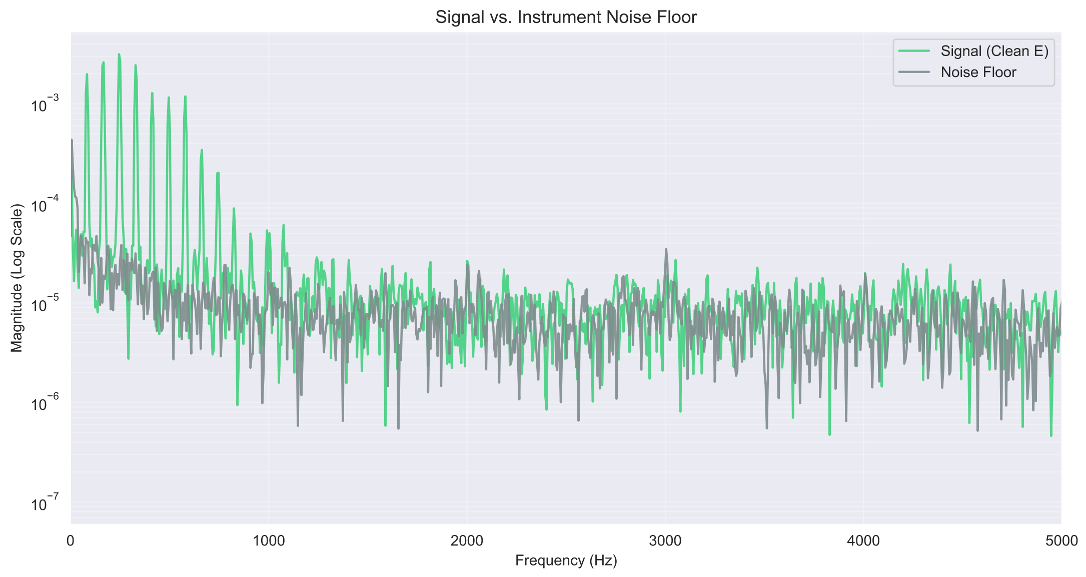


Figure 6.1: **DAQ Sensitivity Analysis.** The system exhibits a read noise RMS of $\approx$ 1.3mV. Note the clean separation between the signal (green) and the noise floor (gray) below 1kHz, yielding an effective SNR of $\approx$ 42dB.

The circuit exhibits a measured gain of $\approx$ 31dB (a voltage ratio of $\approx 36\times$), driving the instrument-level input to near-rail voltages.

6.3.1 The Saturation Gradient

The visual density of the output waveform indicates the physical mechanism of the distortion. Plotting V_{out} as a function of V_{in} reveals the device's **Transfer Characteristic**.

- **Linear Region:** For small excursions ($|V_{in}| < 50\text{mV}$), the response is ohmic ($R^2 \approx 0.99$).
- **Saturation Region:** As V_{in} approaches the rails (0V and 3.3V), the derivative dV_{out}/dV_{in} approaches zero continuously.

Mathematically, this distinguishes the CMOS topology from standard silicon diode clipping. While diodes exhibit a sharp discontinuity ($dV/dt \to \infty$) at the forward voltage threshold (V_f), the CMOS inverter behaves as a continuous hyperbolic tangent (tanh) function, resulting in a smoother transition into saturation.

6.4 Topology & Harmonic Fingerprint

The primary engineering hypothesis of this project was that the CD4049 CMOS Hex Inverter, despite being a digital logic chip, behaves analogously to a Vacuum Tube Triode when biased into its linear region. Figure 6.3 presents the phase-locked comparison of the Clean vs. Driven signal.

6.4.1 Time Domain: The Soft Knee

As seen in the left panel of Figure 6.3, the waveform peaks exhibit a "Soft Knee" compression. This confirms the continuous transfer function described in Section 6.3. The peaks are not sheared off flatly;

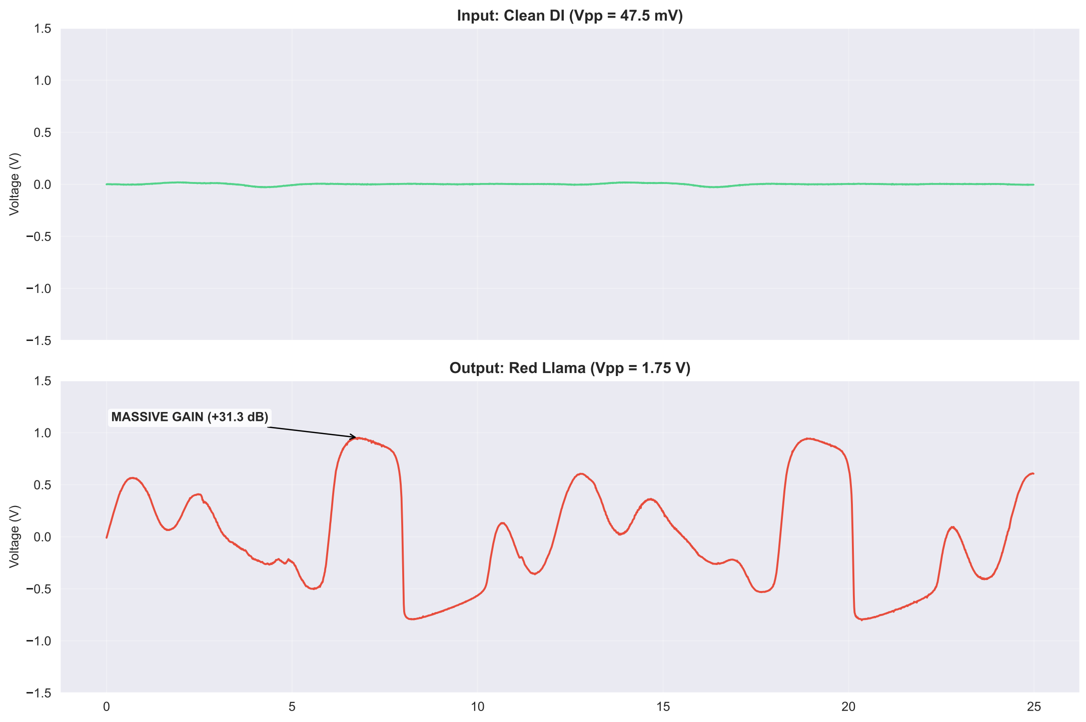


Figure 6.2: **Gain Stage Characterization.** The circuit provides ≈ 31dB of gain. The squaring of the waveform indicates the onset of heavy saturation, where the MOSFETs collide with the power supply rails.

they are rounded. This preserves the dynamic "feel" of the instrument, allowing the player to control the distortion amount via pick attack velocity.

6.4.2 Frequency Domain: Even-Order Dominance

The spectral density analysis (Figure 6.3, Right) reveals the harmonic signature of this topology.

- **Even Harmonics (2^{nd}, 4^{th}):** There is a distinct, high-amplitude presence of the 2nd harmonic (the Octave). This asymmetry is mathematically consistent with the single-ended transfer function of vacuum tubes and is responsible for the perceived "warmth" of the circuit.
- **Odd Harmonics (3^{rd}, 5^{th}):** While present, they do not completely dominate the spectrum as they would in a symmetrical diode-clipping circuit.

The clear resolution of these harmonic peaks (≈ 5.9 Hz/bin) validates the decision to use a custom Hann-windowed FFT pipeline rather than a standard real-time streaming view.

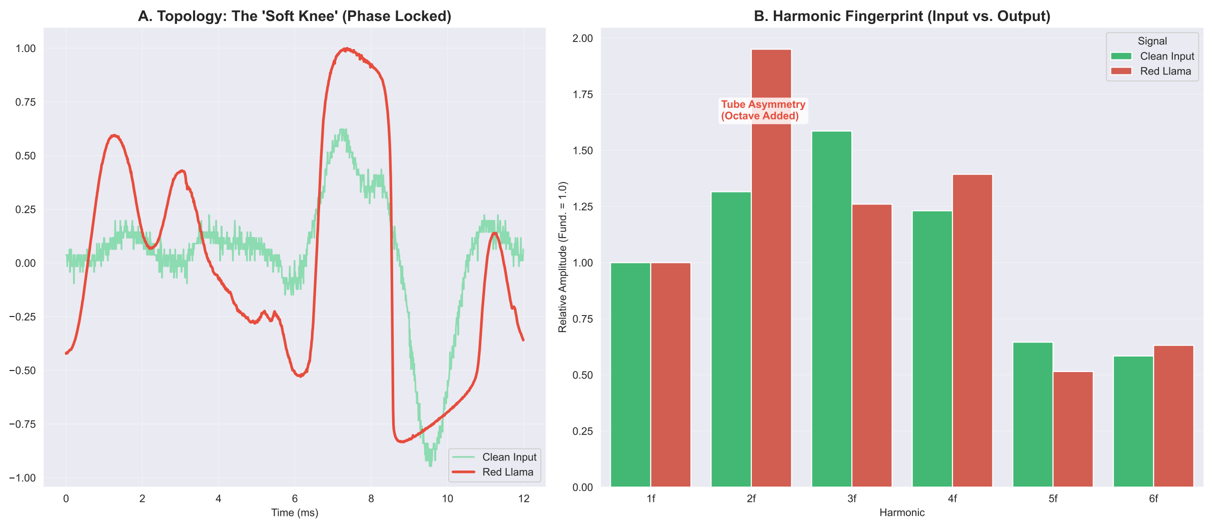


Figure 6.3: **Empirical Validation of CMOS Topology.** (A) Phase-locked time domain analysis reveals 'soft knee' compression characteristics distinct from hard diode clipping. (B) Harmonic fingerprinting confirms a dominant 2nd harmonic (octave) and strong even-order series, consistent with the asymmetrical saturation of vacuum tube triodes.

Chapter 7

Conclusion

7.1 Synthesis: The Recursive Engineering Loop

This technical report documented a deviation from standard hobbyist electronics toward a rigorous systems engineering approach. The project verified that the ”Full-Stack Gap” described in Chapter 1 can be bridged not by purchasing better tools, but by engineering them.

The development of the four interdependent systems—Logistics, Infrastructure, Device, and Instrumentation—demonstrated that the principles of software architecture (modularity, determinism, and regression testing)[7, 6] are directly transferable to physical fabrication. The success of the **Red Llama** build was not a result of manual dexterity, but of the logistical determinism provided by **Star Ground** and the electrical stability provided by the **Linear Power Regulator**.

7.2 Summary of Empirical Findings

The analytical phase (Chapter 6) provided quantitative validation of the system's performance. The data supports the following conclusions regarding the **Red Llama** topology and the custom **RP2040 DAQ**:

1. **Topological Confirmation (The ”Tube” Hypothesis):** The transfer characteristic analysis confirmed that the CD4049 CMOS Hex Inverter, when biased into the linear region, exhibits a continuous hyperbolic tangent (tanh) saturation curve.[16] The absence of derivative discontinuities at the saturation threshold (the ”Soft Knee”) mathematically distinguishes this circuit from standard silicon diode clipping.

2. **Harmonic Fingerprint:** Spectral decomposition revealed a dominant 2nd-order harmonic ($f = 2f_0$) and a decaying series of even-order harmonics. This confirms that the CMOS inverter mimics the asymmetrical transfer function of single-ended Class A vacuum tube triodes, validating the psychoacoustic description of the device as ”warm” or ”amp-like.”[28]

3. **Instrumentation Efficacy:** The custom RP2040 Data Acquisition System met or exceeded all design requirements.

   – **Bandwidth:** A calibrated sample rate of 97.8 kSps was achieved, providing a Nyquist frequency of $\approx$ 48.9 kHz, well beyond the audible spectrum.[29]

   – **Fidelity:** The system demonstrated a read noise floor of 1.3mV$_{rms}$ and a dynamic range (SNR) of $\approx$ 42 dB relative to the saturated signal, proving it is a viable tool for audio frequency analysis.

7.3 Phase II: Active Signal Injection (In Progress)

Current characterization has relied on passive burst capture of organic instrument input. To achieve laboratory-grade transfer function plotting, the project is currently transitioning to an **Active Signal Injection** phase.

A custom Host-to-Device interface is under development, utilizing the host laptop's High-Definition Audio DAC as a signal generator.

- **Hardware Interface:** A custom impedance-matching cable (Aux TRS $\rightarrow$ Mono TS) has been fabricated to bridge the consumer line-level output of the host to the high-impedance instrument input of the DUT.
- **Software Generation:** A Python-based Direct Digital Synthesis (DDS) engine has been written to generate precise sine sweeps (20Hz–20kHz) and step functions.

This upgrade will allow for the automation of Bode plotting and Total Harmonic Distortion (THD) sweeping in the next revision of this technical report.

7.4 Future Roadmap: The Silicon Revision (v2.0)

The current device, constructed on perfboard with discrete modules, serves as a validated prototype (v1.0). While effective for characterizing soft-clipping topologies, the reliance on the RP2040's internal 12-bit ADC and manual jumper configuration limits its utility as a precision laboratory standard.

Based on the empirical data collected in this report, a comprehensive "Silicon Revision" (v2.0) is proposed to transition the system from a development tool to an autonomous measurement engine.

7.4.1 Hardware Architecture

The next iteration will migrate from modular sub-assemblies to a unified **4-layer PCB** utilizing Surface Mount Devices (SMD).

- **Noise Floor:** A dedicated internal ground plane will replace the star-ground wiring to further suppress Electromagnetic Interference (EMI), targeting a noise floor of < -80dB.
- **Connectivity:** Standard 0.1" headers will be replaced with shielded **BNC connectors** to interface with standard oscilloscope probes.

7.4.2 Signal Chain Evolution

To achieve *Audio Analyzer* grade performance, the internal data converters will be bypassed in favor of dedicated I2S codecs:

- **Acquisition (ADC):** An external 24-bit I2S ADC (e.g., PCM1808) will replace the internal 12-bit converter. This will increase the dynamic range significantly, allowing for the resolution of harmonic distortion products currently buried in the quantization noise.
- **Generation (DAC):** A high-resolution DAC (e.g., PCM5102) will be integrated directly onto the bus. This eliminates the dependency on the host laptop's sound card (described in Section 7.4) and allows the MCU to generate perfectly synchronized test tones for automated Bode plotting.

7.4.3 Programmable Analog Front End (AFE)

The current manual configuration (physical jumpers for Gain and AC/DC coupling) introduces mechanical wear and prevents remote automation. The v2.0 AFE will utilize:

- **Relay Switching:** To toggle between AC and DC coupling modes without signal degradation.
- **Programmable Gain Amplifier (PGA):** To provide software-controlled attenuation, enabling the system to auto-range between millivolt-level guitar pickups and high-voltage modular synthesizer rails (±12V).

7.5 Closing Statement

The **Systems Audio Lab** project stands as a proof-of-concept for the *systems* approach to maker-electronics. By refusing to treat the supply chain, power delivery, and validation as externalities, the project yielded a device with a verified noise floor and a mathematically characterized distortion profile. The proposed evolution to v2.0 demonstrates that the distinction between software code and silicon hardware is increasingly arbitrary; both are simply distinct layers of the same engineering stack.

Appendix A

Source Code

A.1 Firmware: Acquisition Loop

MicroPython logic running on the RP2040, handling the high-speed burst capture.

```python
# pyright: reportMissingImports=false

import array
import gc
import sys

import machine
import micropython
import uselect

# Configuration
ADC_PIN_NUM = 28
MAX_SAMPLES = 16384  # Science Mode Buffer
LIVE_SAMPLES = 1024  # Video Mode Buffer

# Setup ADC
adc = machine.ADC(machine.Pin(ADC_PIN_NUM))
# Allocate max memory once to avoid fragmentation
adc_buffer = array.array("H", [0] * MAX_SAMPLES)


# Pre-compile the capture function to arm machine code
@micropython.native
def capture_burst(adc_obj, buf, size: int):
    """
    Reads a burst of analog values into a buffer using native code generation.

    This function is decorated with @micropython.native for speed. It avoids
    memory allocation during the loop to maintain deterministic timing.

    Parameters
    ----------
    adc_obj : machine.ADC
        The configured ADC instance to read from.
    buf : array.array
        The pre-allocated buffer (array of unsigned short 'H') to store data.
    size : int
        The number of samples to capture. Must be <= len(buf).
    """
    for i in range(size):
        buf[i] = adc_obj.read_u16()
```

```python


def main():
    """
    Main firmware loop.

    Listens for single-character commands over USB Serial (stdin):
    - 's': Science Mode (High Res). Captures MAX_SAMPLES, with GC disabled
           during capture for stability. Sends full buffer.
    - 'v': Video Mode (Low Latency). Captures LIVE_SAMPLES. Sends partial
           buffer immediately. No GC manipulation for higher frame rates.
    """
    poll_obj = uselect.poll()
    poll_obj.register(sys.stdin, uselect.POLLIN)

    while True:
        # Poll with a 100ms timeout to allow other background tasks if necessary
        if not poll_obj.poll(100):
            continue

        # Read 1 byte from stdin
        cmd = sys.stdin.read(1)

        # 's' = SCIENCE MODE (High Res, Deep Buffer)
        if cmd == "s":
            gc.disable()
            capture_burst(adc, adc_buffer, MAX_SAMPLES)
            gc.enable()
            # Send FULL buffer
            sys.stdout.buffer.write(memoryview(adc_buffer))
            gc.collect()

        # 'v' = VIDEO MODE (Low Latency, Short Buffer)
        elif cmd == "v":
            # No need to disable GC for short bursts, keeps it snappy
            capture_burst(adc, adc_buffer, LIVE_SAMPLES)

            # Send ONLY the first 1024 samples
            # This slicing [:LIVE_SAMPLES] is very fast (no copying)
            sys.stdout.buffer.write(memoryview(adc_buffer)[:LIVE_SAMPLES])

            # Note: We don't GC here to keep frame rate high


if __name__ == "__main__":
    main()
```

Listing A.1: main.py

A.2 Host: Calibration Algorithm

The weighted-average logic used to derive the true sample rate from mains hum (Ref. Chapter 5).

```python
import json
import os
import time
from typing import Optional

import matplotlib.pyplot as plt
import numpy as np
from scipy.signal import windows

from . import config, daq, dsp


def save_calibration(fs: float) -> None:
    """
    Saves the calibrated sampling rate to the calibration JSON file.

    Creates the parent directory for the calibration file if it does not exist.
    The file will contain the sampling rate, a timestamp, and the hardware default.

    Parameters
    ----------
    fs : float
        The calculated true sampling rate in Hz.
    """
    # 1. Create data directory if it doesn't exist yet
    os.makedirs(os.path.dirname(config.CALIBRATION_FILE_PATH), exist_ok=True)

    data = {
        "fs": fs,
        "timestamp": time.time(),
        "date_str": time.strftime("%Y-%m-%d %H:%M:%S"),
        "hardware_default": config.FS_DEFAULT,
    }

    with open(config.CALIBRATION_FILE_PATH, "w") as f:
        json.dump(data, f, indent=4)

    print(f"💾 Calibration saved to {config.CALIBRATION_FILE_PATH}")


def load_calibration() -> Optional[float]:
    """
    Attempts to load a previously saved sampling rate from disk.

    Returns
    -------
    Optional[float]
        The cached sampling rate in Hz if the file exists and is valid.
        Returns None if the file is missing or corrupted.
    """
    if not os.path.exists(config.CALIBRATION_FILE_PATH):
        return None

    try:
        with open(config.CALIBRATION_FILE_PATH, "r") as f:
            data = json.load(f)

        fs = data.get("fs")
        date = data.get("date_str", "unknown date")
        print(f"📂 Loaded cached calibration: {fs:.1f} Hz ({date})")
        return float(fs)

    except Exception as e:
        print(f"⚠️ Could not load calibration: {e}")
        return None


def calibrate_fs_robust(visualize: bool = True) -> float:
    """
    Calculates the true sampling rate by analyzing 60Hz mains hum.

    This function captures a burst of data, identifying the mains frequency
    assuming the user is touching the input jack. It uses a weighted average
    of FFT bins around the peak for sub-bin precision.

    Parameters
    ----------
    visualize : bool, optional

        If True, plots the first few cycles of the
        captured signal for visual verification.

        Default is True.

    Returns
    -------
    float
        The calculated real sampling rate in Hz.
        Returns config.FS_DEFAULT if calibration fails or SNR is too low.
    """
    print("👉 TOUCH JACK TIP NOW for 60Hz Calibration...")

    try:
        # 1. Capture
        with daq.DAQInterface() as device:
            raw = device.capture_burst()
            voltages = dsp.raw_to_volts(raw)

        # 2. AC Couple
        ac_signal = dsp.remove_dc(voltages)

        # 3. FFT Analysis
        n = len(ac_signal)
        windowed = ac_signal * windows.hann(n)
        fft_mag = np.abs(np.fft.fft(windowed)[: n // 2])

        # 4. Find Peak
        peak_idx = np.argmax(fft_mag)
        peak_mag = fft_mag[peak_idx]

        # 5. Calculate SNR (exclude 10 bins around peak)
        noise_mask = np.ones(len(fft_mag), dtype=bool)
        noise_mask[max(0, int(peak_idx) - 5) : min(len(fft_mag), int(peak_idx) + 5)] = (
            False
        )
        noise_floor = np.mean(fft_mag[noise_mask])

        snr = peak_mag / noise_floor if noise_floor > 0 else 0

        print("\n--- DIAGNOSTICS ---")
        print(f"Signal Strength (SNR): {snr:.1f}x noise floor")

        if snr < 10:
            print("❌ WARNING: Signal too weak! Touch the jack firmly.")
            return float(config.FS_DEFAULT)

        # 6. Weighted Average for Precision
        win_idxs = np.arange(peak_idx - 2, peak_idx + 3)
        # Boundary check
        valid_idxs = win_idxs[(win_idxs >= 0) & (win_idxs < len(fft_mag))]

        precise_idx = np.sum(valid_idxs * fft_mag[valid_idxs]) / np.sum(
            fft_mag[valid_idxs]
        )

```

```python
        # 7. Calculate Real FS: Fs = (Target_Freq * N) / Peak_Index
        real_fs = (60.0 * n) / precise_idx
        print(f"Calculated FS:       {real_fs:.1f} Hz")

        if visualize:
            plt.figure(figsize=(10, 3))
            samples_per_cycle = int(real_fs / 60)
            plt.plot(ac_signal[: samples_per_cycle * 5], color="cyan")
            plt.title("Visual Check: Clean 60Hz Sine Waves?")
            plt.grid(True, alpha=0.3)
            plt.show()

        return float(real_fs)

    except Exception as e:
        print(f"❌ Calibration Failed: {e}")
        return float(config.FS_DEFAULT)
```

Listing A.2: src/calibration.py

A.3 Host: Signal Processing (DSP)

FFT windowing and THD calculation routines (Ref. Chapter 6).

```python
from typing import Tuple

import numpy as np
import scipy.signal as spsig

from . import config


def raw_to_volts(raw_data: np.ndarray) -> np.ndarray:
    """
    Converts raw uint16 ADC values to float64 voltages.

    Parameters
    ----------
    raw_data : np.ndarray
        Array of raw ADC integer values (0 to config.ADC_MAX_VAL).

    Returns
    -------
    np.ndarray
        Array of voltage values scaled to config.V_REF.
    """
    return (raw_data / config.ADC_MAX_VAL) * config.V_REF


def remove_dc(signal: np.ndarray) -> np.ndarray:
    """
    Subtracts the mean (DC offset) from the signal.

    Parameters
    ----------
    signal : np.ndarray
        Input signal array.

    Returns
    -------
    np.ndarray
```

```python
        The signal centered around 0.0.
    """
    return signal - np.mean(signal)  # type: ignore[no-any-return]


def software_trigger(signal: np.ndarray, threshold: float = config.V_MID) -> np.ndarray:
    """
    Stabilizes a periodic waveform by rolling the array to align
    the first rising edge crossing with index 0.

    Parameters
    ----------
    signal : np.ndarray
        Input signal array.
    threshold : float, optional
        Voltage level to define the trigger crossing. Defaults to config.V_MID.

    Returns
    -------
    np.ndarray
        A cyclically shifted copy of the signal starting at the first rising edge.
        Returns the original signal if no crossing is found.
    """
    # Boolean mask: (Current < Thresh) AND (Next >= Thresh)
    crossings = np.where((signal[:-1] < threshold) & (signal[1:] >= threshold))[0]

    if crossings.size > 0:
        return np.roll(signal, -crossings[0])
    return signal


def compute_spectrum(signal: np.ndarray, fs: float) -> Tuple[np.ndarray, np.ndarray]:
    """
    Computes the one-sided FFT magnitude spectrum using a Hann window.

    Parameters
    ----------
    signal : np.ndarray
        (N,) Input time-domain signal.
    fs : float
        Sampling rate in Hz.

    Returns
    -------
    freq_axis : np.ndarray
        (N/2 + 1,) Array of frequency bins in Hz.
    fft_mag : np.ndarray
        (N/2 + 1,) Array of magnitude values (normalized).
    """
    n = signal.size
    # Apply window to reduce spectral leakage
    windowed = signal * spsig.windows.hann(n)

    # FFT
    fft_complex = np.fft.rfft(windowed)
    fft_mag = (2.0 / n) * np.abs(fft_complex)

    # Frequency axis
    freq_axis = np.fft.rfftfreq(n, d=1.0 / fs)

    return freq_axis, fft_mag


```

```
def estimate_fundamental(
    freqs: np.ndarray, mags: np.ndarray, fmin: float = 20.0, fmax: float = 2000.0
) -> float:
    """
    Finds the frequency of the dominant peak within a specific band.

    Parameters
    ----------
    freqs : np.ndarray
        Array of frequency bins.
    mags : np.ndarray
        Array of magnitudes corresponding to freqs.
    fmin : float, optional
        Minimum frequency to consider. Defaults to 20.0.
    fmax : float, optional
        Maximum frequency to consider. Defaults to 2000.0.

    Returns
    -------
    float
        The frequency in Hz of the largest peak within [fmin, fmax].
        Returns 0.0 if no peak is found or arrays are empty.
    """
    if freqs.size == 0:
        return 0.0

    mask = (freqs >= fmin) & (freqs <= fmax)
    if not np.any(mask):
        return 0.0

    f_subset = freqs[mask]
    m_subset = mags[mask]

    # Simple peak finding
    idx = np.argmax(m_subset)
    return float(f_subset[idx])


def calculate_selective_thd(
    signal_arr: np.ndarray,
    fs: float,
    fundamental_freq: float = 82.4,
    n_harmonics: int = 10,
) -> float:
    """
    Calculates Total Harmonic Distortion (THD) by summing ONLY the energy
    at specific harmonic frequencies, ignoring the broad-band noise floor.

    Parameters
    ----------
    signal_arr : np.ndarray
        Time-domain signal array.
    fs : float
        Sampling rate in Hz.
    fundamental_freq : float, optional
        The fundamental frequency of the signal (e.g., 82.4 for low E).
    n_harmonics : int, optional
        Number of harmonics to include in the calculation.

    Returns
    -------
    float
        THD percentage (0.0 to 100.0+).
```

```python
    """
    # Use internal compute_spectrum function
    freqs, mags = compute_spectrum(signal_arr, fs)

    # 1. Get Fundamental Magnitude
    window = 5  # Narrow window to exclude nearby noise
    fund_mask = (freqs > fundamental_freq - window) & (
        freqs < fundamental_freq + window
    )

    if not np.any(fund_mask):
        return 0.0

    fund_mag = np.max(mags[fund_mask])

    # 2. Sum ONLY the Harmonic Peaks (Selective)
    harmonic_sum_sq = 0.0

    for i in range(2, n_harmonics + 1):
        target_f = fundamental_freq * i

        # Look for a peak at exactly this frequency
        mask = (freqs > target_f - window) & (freqs < target_f + window)

        if np.any(mask):
            # We subtract the estimated noise floor from the peak to be extra safe
            # Estimate local noise by looking just outside the window
            noise_mask = (
                (freqs > target_f - window * 3)
                & (freqs < target_f + window * 3)
                & (~mask)
            )
            local_noise = np.mean(mags[noise_mask]) if np.any(noise_mask) else 0

            # Get peak and subtract noise (soft floor at 0)
            h_mag = max(0.0, np.max(mags[mask]) - local_noise)

            if h_mag > 0:
                harmonic_sum_sq += h_mag**2

    # 3. Calculate Selective THD
    thd_pct = (np.sqrt(harmonic_sum_sq) / fund_mag) * 100
    return float(thd_pct)


def smart_align(sig_ref: np.ndarray, sig_target: np.ndarray) -> np.ndarray:
    """
    Aligns 'sig_target' to match the phase of 'sig_ref' using Cross-Correlation.

    Parameters
    ----------
    sig_ref : np.ndarray
        The reference signal (stationary).
    sig_target : np.ndarray
        The signal to be shifted.

    Returns
    -------
    np.ndarray
        The shifted version of sig_target that aligns best with sig_ref.
    """
    # 1. Compute Cross-Correlation
    correlation = spsig.correlate(sig_target, sig_ref, mode="full")
    lags = spsig.correlation_lags(sig_target.size, sig_ref.size, mode="full")

    # 2. Find the lag that maximizes correlation
    lag = lags[np.argmax(correlation)]

    # 3. Shift the target signal
    if lag > 0:
        # Target is ahead, roll it back
        aligned = np.roll(sig_target, -lag)
    else:
        # Target is behind, roll it forward
        aligned = np.roll(sig_target, -lag)

    return aligned
```

Listing A.3: src/dsp.py

A.4 Host: Function Generator Logic (Phase II)

Core logic for the "Laptop-as-Generator" system, capable of generating frequency sweeps for future transfer function analysis.

```python
"""
Script to play a logarithmic sine sweep and visualize it in real-time.

This script generates a log sine sweep audio buffer, plays it through the system
output, and simultaneously streams data from the DAQ to a live oscilloscope
window. This allows for visual verification of the transfer function measurement
setup before committing to a recording.
"""

import sounddevice as sd
from sysaudio import audio, daq, viz

# Configuration
F_START: float = 20.0  # Start Frequency (Hz)
F_END: float = 20000.0  # End Frequency (Hz)
DURATION: float = 5.0  # Seconds
AMPLITUDE: float = 0.5  # 0.0 to 1.0


def main() -> None:
    """
    Main execution entry point.

    Generates the sweep buffer, establishes the DAQ connection, and launches
    the live visualization. The audio playback is triggered via a callback
    once the visualization window is fully initialized to ensure synchronization.
    """
    print(f"Generating Log Sweep ({F_START:.0f}-{F_END:.0f}Hz)...")

    # 1. Generate Buffer
    fs_audio: int = 48000
    wave = audio.generate_log_sweep(F_START, F_END, DURATION, fs_audio, AMPLITUDE)

    # Define the play function
    def start_playback() -> None:
        print("🔊 Playing Sweep...")
        sd.play(wave, samplerate=fs_audio, blocking=False)

    # 2. Start Scope
    # on_launch triggers playback only when the window is visible
    with daq.DAQInterface() as device:
        viz.run_live_scope(
            device.stream_generator(),
            title="Transfer Function: Log Sine Sweep",
            stop_condition=lambda: not sd.get_stream().active,
            on_launch=start_playback,
        )

    print("Done.")


if __name__ == "__main__":
    main()
```

Listing A.4: scripts/signal/play_sweep.py

References

[1] *AN-1950: Silently Powering Low Noise Applications.* Texas Instruments. URL: https://www.ti.com/lit/an/snoa543a/snoa543a.pdf (visited on 01/26/2026).

[2] *BOM (Bill of Materials) Pitfalls: 10 Mistakes That Kill Product Launches.* OpenBOM. Aug. 21, 2025. URL: https://www.openbom.com/blog/bom-bill-of-materials-pitfalls-10-mistakes-that-kill-product-launches (visited on 01/26/2026).

[3] *Should You Use Star Point Grounding in a PCB?* NW Engineering LLC. Sept. 26, 2021. URL: https://www.nwengineeringllc.com/article/should-you-use-star-point-grounding-in-a-pcb.php (visited on 01/26/2026).

[4] *A Survey on Hallucination in Large Language Models.* 2023. URL: https://arxiv.org/abs/2311.05232 (visited on 01/26/2026).

[5] jsvine and contributors. *pdfplumber: Plumb a PDF for detailed information about each character, rectangle, and line; extract text and tables.* URL: https://github.com/jsvine/pdfplumber (visited on 01/26/2026).

[6] Holger Krekel et al. *pytest x.y.* Version x.y. Contributors: Holger Krekel and Bruno Oliveira and Ronny Pfannschmidt and Floris Bruynooghe and Brianna Laugher and Florian Bruhin and others. 2004. URL: https://github.com/pytest-dev/pytest.

[7] *pytest-regtest documentation.* https://pytest-regtest.readthedocs.io. Accessed 2026-01-25.

[8] *TL07xx Low-Noise FET-Input Operational Amplifiers (datasheet).* Texas Instruments. URL: https://www.ti.com/lit/ds/symlink/tl072.pdf (visited on 01/26/2026).

[9] *OPAx134 High-Performance, SoundPlus™ Audio Operational Amplifiers (datasheet).* Texas Instruments. URL: https://www.ti.com/lit/ds/symlink/opa2134.pdf (visited on 01/26/2026).

[10] *PNP "Positive Ground" Pedal Considerations.* Amplified Parts. URL: https://www.amplifiedparts.com/tech-articles/pnp-positive-ground-pedals (visited on 01/26/2026).

[11] *You can Build the Perfect Germanium Fuzz Face.* ElectroSmash. May 31, 2012. URL: https://www.electrosmash.com/germanium-fuzz (visited on 01/26/2026).

[12] *A Brief Hobbyist Primer on Clipping Diodes.* GuitarPedalX. Aug. 20, 2023. URL: https://www.guitarpedalx.com/news/news/a-brief-hobbyist-primer-on-clipping-diodes (visited on 01/26/2026).

[13] *Datasheet for Tayda part A-1597.* Tayda Electronics. URL: https://www.taydaelectronics.com/datasheets/files/A-1597.PDF (visited on 01/26/2026).

[14] *1N5817: Schottky Barrier Rectifier Diode (datasheet).* Distributor-hosted datasheet. Tayda Electronics. URL: https://www.taydaelectronics.com/datasheets/A-159.pdf (visited on 01/26/2026).

[15] EIC. *1N4001 Standard Recovery Power Rectifier, 1 A, 50 V (DO-41) Datasheet.* A-162. EIC. URL: https://www.taydaelectronics.com/datasheets/A-162.pdf.

[16] *CD4049UB, CD4050B: CMOS Hex Buffer/Converter (datasheet).* Texas Instruments. URL: https://www.ti.com/lit/ds/symlink/cd4049ub.pdf (visited on 01/26/2026).

[17] *Tayda Red Llama Overdrive.* https://www.taydakits.com/instructions/red-llama. Tayda Electronics.

[18] *RP2040 Datasheet (RP-008371-DS).* Raspberry Pi Ltd. URL: https://pip-assets.raspberrypi.com/categories/814-rp2040/documents/RP-008371-DS-1-rp2040-datasheet.pdf?disposition=inline (visited on 01/26/2026).

[19] *RP2040 Datasheet.* Raspberry Pi Ltd. URL: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf (visited on 01/26/2026).

[20] *BAT54: Schottky barrier diode (datasheet).* Nexperia. July 1, 2022. URL: https://assets.nexperia.com/documents/data-sheet/BAT54.pdf (visited on 01/26/2026).

[21] *1N4148 Switching Diode (datasheet).* Generic / distributor copy. URL: https://www.gotronic.fr/pj2-1n4148-1661.pdf (visited on 01/26/2026).

[22] *Maximising MicroPython speed.* MicroPython Project. URL: https://docs.micropython.org/en/latest/reference/speed_python.html (visited on 01/26/2026).

[23] *Signal Processing (scipy.signal) — SciPy Tutorial.* SciPy. URL: https://docs.scipy.org/doc/scipy/tutorial/signal.html (visited on 01/26/2026).

[24] *numpy.fft.fft — NumPy Manual.* NumPy. URL: https://numpy.org/doc/stable/reference/generated/numpy.fft.fft.html (visited on 01/26/2026).

[25] *Fourier Transforms (scipy.fft) — SciPy Tutorial.* SciPy. URL: https://docs.scipy.org/doc/scipy/tutorial/fft.html (visited on 01/26/2026).

[26] *scipy.signal.windows.hann — SciPy Manual.* SciPy. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.windows.hann.html (visited on 01/26/2026).

[27] Fredric J. Harris. “On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform”. In: *Proceedings of the IEEE* 66.1 (1978), pp. 51–83. URL: https://ece.uprm.edu/~domingo/teaching/inel5309/On%20the%20Use%20of%20Windows%20for%20Harmonic%20Analysis%20with%20the%20DFT.pdf (visited on 01/26/2026).

[28] *When Distortion Is Good.* Sonarworks. URL: https://www.sonarworks.com/blog/learn/when-distortion-is-good (visited on 01/26/2026).

[29] *Nyquist–Shannon sampling theorem.* Wikipedia. URL: https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem (visited on 01/26/2026).