Experiment Report: NgSpice-based Frequency-Domain Optimization Methods and Implementation Details for cTamp

Research objectives and experimental rationale

For stimulated Raman scattering (SRS/T-SRS) applications, our design targets are defined by frequency-domain performance and practical stability. Specifically, we aim to place the transimpedance gain resonance at 20 MHz, to achieve an exact −3 dB bandwidth of 1 MHz at that center frequency, to minimize output noise density subject to adequate phase margin and robust transient behavior, and to maximize first-stage gain without degrading the signal-to-noise ratio, thereby enabling downstream amplification stages to operate at an optimal overall SNR. To reach these targets in a cTamp topology, we leverage a resonant RLC feedback network and formulate the design as a constrained search problem over multiple physical components (Rf, Cf, Lf, and a compensation inductor Lc), with the op amp and the high-Q feedback inductor modeled by realistic vendor macro-models to capture high-frequency nonidealities. Because the interaction between Lc and Rf, and their impact on both the resonance and noise cannot be reliably determined from simple analytical approximations, we resort to automated, programmatic parameter exploration and convergence in silico.

Problem formulation for simulation

The center frequency near the transimpedance peak is predominantly set by the product Lf·Cf, with $f0 \approx 1/(2\pi\sqrt{(Lf \cdot Cf)})$; however, component parasitics and frequency-dependent device behavior perturb the ideal relationship. At the target operating point, and for fixed Lf, Cf, Lc and a given op amp open-loop response, the closed-loop bandwidth exhibits a

monotonic dependence on Rf—larger Rf narrows the bandwidth, while smaller Rf broadens it—enabling a bracketed search strategy. The op amp and the feedback inductors are emulated using realistic macro-models; to ensure good high-frequency performance, we select Coilcraft 1812CS inductors with the highest available self-resonant frequency. Because the composite network impedance Re{Zfb} and noise sources are frequency-dependent, naive hand calculations are insufficient; hence, an automated approach is necessary to explore and tune combinations of Lf, Cf, Lc, and Rf to meet frequency and noise goals while maintaining stability.

Toolchain

We use NgSpice (v44.2) as the SPICE simulator and Python (v3.12) as the automation and analysis environment. The experimental code is released under GPLv2 at https://github.com/YWh0301/pyng. This combination enables batch parameter sweeps, scripted convergence routines, and robust parsing for post-processing of AC and noise spectra, with open-model compatibility for op amps and magnetics. Installation and execution are as follows:

1. Install NgSpice (v44.2 or later) and Python dependencies (e.g., numpy, matplotlib, rich).
2. Prepare the components directory (including inductor .cir models and the ADA4817 macro-model).
3. Run python change_Lc_Rf.py -s to execute simulations; run python change_Lc_Rf.py -d <pkl_file> to plot results.

NgSpice overview

NgSpice, derived from SPICE3f5, supports AC, TRAN, NOISE analyses and parameter sweeps, making it well suited for analog and mixed-signal design optimization and reproducible validation. Our design requires fast iteration on feedback parameters (Rf, Lf, Cf, Lc), automated and repeatable searches, and reliable machine-readable outputs for noise spectra and gain curves, all of which NgSpice provides in an open, scriptable form. Its compatibility with third-party models via .include ensures realistic behavior of the op amp and inductors, and its command-line batch mode integrates cleanly with Python for large-scale sweeps.

Version note and known issue: On NgSpice v44.2, concurrent AC and NOISE execution within the same run can yield unstable NOISE outputs.

We therefore decouple analyses: AC is used first to extract frequency response and bandwidth, followed by a separate NOISE run for spectral density evaluation.

Python wrapper (pyng): driving NgSpice for automated, reproducible search

To support repeatable, scalable automation, we developed a lightweight Python wrapper, pyng, which rewrites template netlists, injects device models and .dot commands, modifies component values programmatically, runs NgSpice in batch, and parses the out.raw files into numpy arrays for numerical post-processing.

- Netlist rewriting and parameter injection:
    1. Read a base netlist (.cir) and inject "component name → new value" updates for modifiable elements (Rf, Cf, Lf, Lc, etc.).
    2. Insert .include directives for device and magnetic models and add .ac/.tran/.noise commands as needed.
    3. Write each modified netlist into an isolated working directory to ensure parameter isolation and reproducibility.
- NgSpice invocation and results parsing: pyng executes ngspice in batch mode (-b) and directs outputs to out.raw (-r). A raw parser (rawread) then extracts variable names, units, complex/real flags, and frequency/waveform data into structured numpy arrays, enabling direct computation of transimpedance magnitude $|V_{out}|/|I_{in}|$, −3 dB points, and noise spectra at specific frequencies.
- Interface robustness and safeguards: The wrapper provides named interfaces to add includes or dot commands, modify or delete components, and set simulator behavior modes (e.g., ltpsa), thereby avoiding manual netlist editing errors. Return codes and exceptions are handled explicitly so that failures are reported promptly and can be reproduced.

Experimental setup

Circuit netlist and key component mapping

A netlist excerpt used for the simulations is shown below (full netlist provided in the data files):

```
.title compensate_L
XU1 0 Net-_U1--_ +5V -5V out Net-_U1-FB_ +5V ADA4817
R3 out 0 10k
V5 +48V Net-_I1-Pad2_ DC 0
I1 Net-_C1-Pad1_ Net-_I1-Pad2_ DC 0 SIN( 0 10n 19.69Meg 0 0 0 ) AC 10n
L2 Net-_L2-Pad1_ 0 1.5u
V2 +5V 0 DC 5
V1 +48V 0 DC 48
C1 Net-_C1-Pad1_ +48V 45p
R1 Net-_C1-Pad1_ +48V 2.4G
V3 0 -5V DC 5
V4 Net-_C1-Pad1_ Net-_L2-Pad1_ DC 0
R2 Net-_U1--_ Net-_U1-FB_ 25k
C2 Net-_U1--_ Net-_U1-FB_ 6.3p
XL1 Net-_U1--_ Net-_U1-FB_ 1812cs103
V6 Net-_L2-Pad1_ Net-_U1--_ DC 0
.end
```

Key components and parameters:

1. R2: Feedback resistor Rf (kΩ range), setting transimpedance gain and closed-loop bandwidth.
2. C2: Feedback capacitor Cf (pF range), jointly with Lf setting the resonance near 20 MHz.
3. XL1: Feedback inductor Lf, referencing external device models (e.g., 1812cs103); pyng swaps these at runtime.
4. L2: Compensation inductor Lc (to ground or the inverting node), tuned for phase margin and high-frequency stability.
5. XU1: Op amp macro-model (ADA4817 via .include).
6. I1: Photodiode equivalent current source, used for AC transimpedance calculation and as the NOISE reference (noise v(out) i1).

Search and simulation workflow

We treat the design as a structured two-stage search with stability checks and noise evaluation:

1. For a fixed feedback inductor Lf, search the feedback capacitor Cf so that the transimpedance peak aligns to 20 MHz, yielding an Lf–Cf pairing per inductor model. We use AC analysis over 18–22 MHz with 10,000 linear points to compute ramp = |Vout|/|Iin| and locate f_peak. If f_peak exceeds 20 MHz, we increase Cf; if f_peak is below, we decrease Cf. We start with coarse steps to bracket the solution and transition to weighted bisection. Convergence is declared when |f_peak − 20 MHz| ≤ 0.2%. If iteration limits are

exceeded or bounds are violated, we select the closest feasible solution.

2. With Lf and the paired Cf fixed, sweep a set of compensation inductors Lc (e.g., 1.25–1.56 µH in 0.005 µH steps), and, for each Lc, search Rf so that the −3 dB bandwidth meets 1 MHz. We compute the transimpedance peak and the half-power threshold half_peak = peak/√2, then identify the first and last frequencies satisfying ramp ≥ half_peak, denoted f_low and f_high, and define Δf = f_high − f_low. If Δf exceeds 1 MHz, bandwidth is too wide (Rf too small), and we increase Rf; if Δf is too narrow, we decrease Rf. We use bracketing with adaptive step sizes that shrink near the target to ensure convergence. Validity requires that both −3 dB points lie well inside the analysis range, rather than pinning at frequency boundaries.

3. Stability check: We run a short transient analysis (time step 3 ns, total 3 µs) and use output amplitude thresholds (e.g., >1 V sustained growth) to flag oscillation or large overshoot. Any solution with −3 dB points at the analysis boundaries is rejected.

4. Noise evaluation: We perform a separate NOISE analysis (noise v(out) i1) to extract the output noise spectral density, and sample at the AC resonance frequency index identified in stage 1, avoiding the known AC+NOISE concurrency issue in NgSpice v44.2.

5. For each candidate Lf model (e.g., Coilcraft 1812CS variants), we compile the Lc–Rf solutions that meet the 1 MHz bandwidth constraint and generate summary plots of gain, frequency, stability, and noise across Lc.

Rationale for the strategy

Separating structural parameters (Lf, Cf) from tuning parameters (Rf, Lc) reduces multi-variable coupling and improves search stability. Optimizing against explicit frequency-domain targets with realistic op amp and magnetic models enables automatic and reproducible convergence to implementable solutions with adequate phase margin. The approach facilitates cross-comparison among Lf models and yields design maps (e.g., Lc–Rf relationships) while boundary checks, adaptive step sizing near thresholds, and cached bounds (Rf_top/Rf_btm with bandwidth_top/bandwidth_btm) ensure robust convergence to practical parameter sets.

Results

We save simulation outputs as pickle (.pkl) files, each aggregating, for every Lf across the swept Lc values, the paired Cf and Rf, peak frequency, bandwidth, stability flags, and noise density at the resonance. Visualization utilities plot the dependence of peak gain and corresponding noise on Lc, track the peak frequency versus Lc, and display the optimal Rf and achieved bandwidth versus Lc, with invalid or unstable regions distinctly highlighted.

Across three Lf choices—1812CS103 (10 µH), 1812CS183 (18 µH), and 1812CS333 (33 µH)—the analysis shows that, under the fixed 1 MHz bandwidth constraint, the cTamp transimpedance gain at the resonance is essentially invariant to Rf for a given Lf model, as expected from the monotonic bandwidth–Rf relationship at the operating point. The output noise is minimized when the impedance matching condition $Lc \cdot Cpd \approx Lf \cdot Cf$ holds, reflecting balanced partitioning among resistor thermal noise, op amp noise, and network impedance; when Lc is too small, achieving 1 MHz bandwidth requires Rf to be too large, and resistor thermal noise dominates, whereas when Lc is too large, op amp noise becomes dominant due to mismatch. Larger Lf enables higher achievable transimpedance at the resonance while maintaining similar SNR, and, importantly, the realized transimpedance can be well below the nominal Rf due to the frequency-dependent $Re\{Zfb\}$ of the RLC feedback. For example, with 1812CS333 and a nominal Rf of 300 kΩ, the maximum realized transimpedance near 20 MHz is approximately 52 kΩ.

Summary and outlook

We have established a target-driven, reproducible optimization workflow for cTamp feedback networks: AC/TRAN/NOISE analyses in NgSpice, decoupled to avoid known concurrency issues, are orchestrated via a Python wrapper (pyng) that performs two-stage searches—first locking 20 MHz via Lf–Cf, then meeting the 1 MHz bandwidth via Lc–Rf—followed by stability checks and noise evaluation. By separating structural and tuning parameters and embedding validity and convergence safeguards, the method reliably yields high transimpedance and low noise solutions with implementable component values and sufficient phase margin. The results demonstrate invariance

of resonance gain to Rf under fixed bandwidth for a given Lf, noise minimization near Lc·Cpd ≈ Lf·Cf, and the ability of larger Lf to produce higher gains without compromising SNR. Methodologically, the workflow is portable (center frequency and bandwidth are parameterized), extensible (op amp and inductor models can be swapped), and reproducible (open codebase, structured data outputs). Sensitivity to device macro-models, PCB parasitics, temperature, and inductor Q is expected; accordingly, we will add parasitic extraction, Monte Carlo tolerance analysis, temperature sweeps, and T-SRS delay/scan parameters, and, when NgSpice resolves AC+NOISE concurrency issues, unify noise evaluation in a single run. These enhancements will tighten simulation–experiment correspondence and broaden applicability to SRS/T-SRS detector front-ends.