XycLOps II

Comprehensive System Documentation & Technical Reference

Fall 2025

Status: Stable / Active Development
Backend Engine: Xyce (Sandia National Labs)
Framework: Python 3.10+ / Tkinter / SciPy

Contents

1 Executive Summary & Introduction

XycLOps (**Xyc**e **L**oop **Op**timizer **s**ystem) is an automated circuit optimization platform. It interfaces with the Xyce parallel electronic simulator to iteratively tune analog component values (resistors, capacitors, inductors) until the circuit's behavior matches a specific target curve.

XycLOps II represents a complete architectural overhaul of the original prototype. While the core "Least Squares" optimization logic remains, the surrounding infrastructure has been rebuilt to support multi-physics simulations, interactive visual data entry, and enterprise-grade logging.

1.1 Key Upgrades in XycLOps II

- **Multi-Physics Engine:** Now supports **AC Frequency Response** and **Noise Analysis** in addition to Transient analysis **[XycLOps II Upgrade]**.
- **Asynchronous Threading Architecture:** The optimization engine now runs on a dedicated background thread, completely decoupling simulation logic from the UI. This ensures the interface remains responsive and interactive even during computationally intensive multi-physics sweeps **[XycLOps II Upgrade]**.
- **Interactive Visual Editors:** Users can "draw" target curves (Piecewise Linear or Step) using drag-and-drop plot interfaces instead of manually creating CSV files. Includes a one-click "Apply to Target" workflow **[XycLOps II Upgrade]**.
- **Data Stream Optimization:** Implemented smart downsampling algorithms to reduce massive simulation datasets (100k+ points) into manageable visual payloads, preventing memory exhaustion and GUI lag **[XycLOps II Upgrade]**.
- **Safe Process Management:** A robust "Abort" mechanism using thread-safe events allows users to instantly terminate runaway simulations without crashing the application or leaving orphaned processes **[XycLOps II Upgrade]**.
- **Modern UI Framework:** A centralized theming engine replaces the standard Tkinter look with a modern, flat design system **[XycLOps II Upgrade]**.
- **Workspace-Aware Logging:** A new session management system creates structured, versioned logs for every optimization run. Stale state is automatically cleared on new netlist uploads **[XycLOps II Upgrade]**.
- **Robust Parser:** Enhanced support for LTSpice-style `.PARAM` definitions and UTF-8-SIG (BOM) stripping **[XycLOps II Upgrade]**.

2 Installation & Environment

2.1 Prerequisites

- **Operating System:** Windows 10/11, macOS, or Linux.
- **Python:** Version 3.10 or higher.
- **Simulator:** Xyce must be installed and added to the system `PATH`. The UI includes a validator to help locate the Xyce executable cross-platform.

2.2 Dependencies

The application relies on a strictly versioned set of libraries. **[XycLOps II Upgrade]** The dependency list was expanded to support the new plotting and theming engines.

- **Core Logic:** `numpy==2.2.3`, `scipy==1.15.2` (Optimization & Interpolation)
- **Visualization:** `matplotlib==3.10.1` (Real-time plotting & Visual Editors)
- **GUI & Imaging:** `pillow==11.1.0`, `tkinter` (Standard Lib)

2.3 Installation Strategy

To ensure stability, XycLOps II is designed to run within a virtual environment.

2.3.1 Windows Setup

```
python -m venv xyclopsvenv
.\xyclopsvenv\Scripts\activate
pip install -r requirements.txt
```

2.3.2 macOS/Linux Setup

```
python -m venv xyclopsvenv
source xyclopsvenv/bin/activate
pip install -r requirements.txt
```

2.4 Startup Profiling [XycLOps II Upgrade]

A new utility, `analyze_startup_time.py`, was added to diagnose slow application launch times. It measures the import overhead of heavy libraries like `matplotlib` and `scipy`.

- **Command:** `python analyze_startup_time.py`
- **Output:** A "snail/lightning" report identifies bottlenecks.

3 System Architecture

XycLOps II follows a **Model-View-Controller (MVC)** pattern with strict thread separation to prevent UI freezing during heavy calculations.

3.1 Directory Structure

The codebase is organized into modular subsystems:

```
project_root/
|-- main_app.py                 # Application Bootstrap (Freeze Support)
|-- analyze_startup_time.py     # [XycLOps II Upgrade] Profiler
|-- backend/                    # THE MODEL (Logic)
|   |-- curvefit_optimization.py # Main Solver Loop (SciPy + Xyce)
|   |-- netlist_parse.py         # SPICE Parser & Injection Logic
|   |-- optimization_process.py  # Threading Wrapper
|   |-- xyce_parsing_function.py # Output Parser (.prn files)
|-- frontend/                   # THE VIEW/CONTROLLER (UI)
|   |-- app_controller.py        # Central State Manager
|   |-- ui_theme.py              # [XycLOps II Upgrade] Design System
|   |-- optimization_summary.py  # Dashboard & Real-time Plotting
```

```
|   |-- optimization_history.py  # [XycLOps II Upgrade] Session Viewer
|   |-- optimization_settings/   # Configuration Modules
|   |   |-- visual_curve_editors.py # [XycLOps II Upgrade] Interactive Editors
|   |   |-- ac_settings_dialog.py   # [XycLOps II Upgrade] AC Config
|   |   |-- noise_settings_dialog.py# [XycLOps II Upgrade] Noise Config
|   |   |-- ...
|-- netlists/                    # Example Circuits
```

3.2 Data Flow

1. **User Input:** The user interacts with the **Frontend**, loading a netlist and configuring parameters.
2. **State Storage:** The `AppController` aggregates this data. *Note: Stale state is explicitly cleared when a new netlist is uploaded to prevent data leakage between runs.*
3. **Dispatch:** When "Run" is clicked, `OptimizationSummary` spawns a **Backend Thread**.
4. **Simulation Loop:**
   - The **Backend** writes a modified netlist to disk.
   - **Xyce** is invoked via subprocess.
   - **Backend** parses the output, compares it to the target, and calculates the error (residual).
   - **SciPy** calculates the next set of component values.
5. **Feedback:** The Backend pushes status messages and plot data into a thread-safe `Queue`.
6. **Visualization:** The Frontend polls this Queue and updates the Real-time Graph and Logs.

4 Frontend Subsystem (UI)

The frontend uses `tkinter` but abandons the default OS look for a custom "Modern" theme.

4.1 Entry Point & Safety

The `main_app.py` script includes `multiprocessing.freeze_support()`. This is crucial for Windows compatibility if the application is packaged into an `.exe` using PyInstaller, ensuring background threads behave correctly.

4.2 Modern Theme Engine [XycLOps II Upgrade]

Located in `frontend/ui_theme.py`, this module enforces a cohesive design language.

- **Palette:** Defines semantic colors (`accent`: Blue #3b82f6, `success`: Green #10b981, `bg_secondary`: Slate #f8fafc).
- **Widget Factories:** Helper functions like `create_card` and `create_primary_button` ensure consistent padding, fonts ("Segoe UI"), and border radii across the entire application.

4.3 App Controller

The `AppController` manages the navigation stack. It allows the user to move non-linearly between the **Netlist Uploader**, **Parameter Selection**, and **Settings** windows while preserving state (e.g., keeping selected resistors active even if you go back to the upload screen).

4.4 Optimization Settings & Dialogs

This module handles the complex configuration required for multi-physics optimization.

4.4.1 Transient Analysis Controls [XycLOps II Upgrade]

While present in v1, v2 surfaces granular controls directly in the UI:

- **Timing:** Configurable `Stop Time`, `Time Step`, `Start Time`, and `Max Step`.
- **UIC:** Toggle for "Use Initial Conditions" to bypass the DC operating point calculation.

4.4.2 AC Analysis Settings [XycLOps II Upgrade]

Managed by `ac_settings_dialog.py`.

- **Sweep Types:** Supports Decade (DEC), Linear (LIN), and Octave (OCT).
- **Validation:** Prevents invalid inputs (e.g., Start Freq $\geq$ Stop Freq, or negative frequencies).

4.4.3 Noise Analysis Settings [XycLOps II Upgrade]

Managed by `noise_settings_dialog.py`.

- **Inputs:** Requires the user to select an **Output Node** (parsed from the netlist) and an **Input Source**.
- **Quantity:** Users can choose between Output Noise Density (`onoise`) or Input-Referred Noise (`inoise`), with optional dB conversion.

4.4.4 Visual Curve Editors [XycLOps II Upgrade]

Located in `visual_curve_editors.py`, this is a major usability enhancement. It embeds `matplotlib` directly into the settings window to allow interactive target definition.

- **Heaviside Editor:** A dedicated GUI for Step Responses. Users drag handles to set Amplitude, Start Time (t_0), and window width.
- **Piecewise Linear (PWL) Editor:** A point-and-click interface.
  - **Interaction:** Double-click to add points; drag points to adjust timing/voltage.
  - **Constraint Visualization:** The `LegacyConstraintsBar` allows users to visualize constraints (e.g., "Voltage must be < 5V") as shaded regions directly on the target graph.

4.5 Optimization Dashboard

Managed by `optimization_summary.py`, this window acts as the mission control center.

- **Real-time Plotting:** Embeds a Matplotlib figure that updates every time the backend produces a new simulation result.
- **Sidebar Logging [XycLOps II Upgrade]:** A collapsible sidebar streams text logs (e.g., "Xyce Run #45 complete") in real-time.
- **Convergence Window [XycLOps II Upgrade]:** A floating, "always-on-top" window displays the current convergence percentage:

$$\text{Improvement} = \frac{\text{Initial Cost} - \text{Current Cost}}{\text{Initial Cost}} \times 100\%$$

5 Backend Subsystem (Logic)

The backend is responsible for the "loop": Modify Netlist → Run Xyce → Measure Error → Adjust Parameters.

5.1 Netlist Parsing & Injection

The `Netlist` class in `netlist_parse.py` parses SPICE files. In XycLOps II, it was significantly expanded.

- **LTSpice Compatibility [XycLOps II Upgrade]:** The parser now correctly handles `.PARAM` definitions and nested logic common in LTSpice netlists.
- **Encoding Safety [XycLOps II Upgrade]:** Files are read with `utf-8-sig` to automatically strip Byte Order Marks (BOM), preventing syntax errors in Xyce.
- **AC Injection [XycLOps II Upgrade]:** The `writeAcCmdsToFile` method injects `.AC` commands.
- **Noise Injection [XycLOps II Upgrade]:** The `writeNoiseCmdsToFile` method handles the complex syntax of `.NOISE`.

5.2 Optimization Engine

This file (`curvefit_optimization.py`) contains the function which drives the entire process.

5.2.1 Session Management [XycLOps II Upgrade]

A new "Workspace-Aware" system organizes results.

- **Structure:** `runs/netlist-results/[NetlistName]/[SessionID]/`.
- **Logic:** The system scans the directory for existing session integers and automatically increments the ID.
- **History View:** A dedicated UI (`optimization_history.py`) allows users to browse past runs and open their log folders.

5.2.2 Multi-Physics Handling [XycLOps II Upgrade]

The solver now adapts its data processing based on the `analysis_mode`:

- **AC Analysis:** Utilizes Xyce's native `VDB` output to retrieve decibel values directly, eliminating the need for post-processing logarithmic conversion in Python.
- **Window Masking:** Constraint evaluation is strictly bound to the target time/frequency window to avoid false penalties from simulation artifacts outside the domain.

5.2.3 Data Downsampling [XycLOps II Upgrade]

To prevent the GUI from crashing when Xyce produces massive datasets (e.g., 100,000+ points), the `_downsample_pairs` function reduces the dataset to ≈5,000 points before sending it to the frontend. The internal optimizer, however, continues to use the full-resolution data for maximum accuracy.

5.3 Threading Model

- **Worker Thread:** The optimization runs in a dedicated daemon thread (`backend.optimization_process.optimizeProcess`).
- **IPC Queue:** Communication happens via a `queue.Queue`.
  - `("Log", msg)`: Text updates.
  - `("UpdateYData", (x, y))`: Plot updates.
- **Polling Loop:** The Frontend's `update_ui` method executes every 100ms to drain this queue.

6 Testing & Validation

6.1 Stress Testing Harness [XycLOps II Upgrade]

Located in `backend/manual_tests/`, these scripts bypass the UI to stress-test the solver logic directly.

- `long_instr_amp_test.py`: Optimizes a 7-variable Instrumentation Amplifier (Transient) with strict tolerances (10^{-14}) to force long runtimes.
- `long_instr_amp_ac_test.py`: The AC counterpart, sweeping 10Hz to 1MHz.
- **ConsoleQueue:** A lightweight mock of the UI queue used to stream results to the terminal during these headless tests.

7 User Workflows

7.1 Transient Optimization (Standard)

1. **Upload:** Load a `.cir` file.
2. **Parameters:** Select components (e.g., `R1`, `C1`).
3. **Settings:** Ensure "Transient" is selected. Configure Stop Time and Step size.
4. **Target:** Draw a Step Response (0V to 5V) using the **Heaviside Editor [XycLOps II Upgrade]**.
5. **Run:** The system tunes the parameters to match the step response.

7.2 AC Optimization [XycLOps II Upgrade]

1. **Settings:** Select "AC Analysis".
2. **Configure:** Click "Configure AC Sweep". Set Sweep to `DEC`, Start `10`, Stop `1Meg`.
3. **Response:** Select "Magnitude (dB)" or "Phase".
4. **Target:** Upload a CSV defining the desired Bode plot (Gain vs. Freq).
5. **Run:** The optimizer adjusts components to match the frequency response.

7.3 Noise Optimization [XycLOps II Upgrade]

1. **Settings:** Select "Noise Analysis".
2. **Configure:**
   - **Output Node:** `V(out)`
   - **Input Source:** `Vsrc`
   - **Quantity:** `onoise` (Output Noise Density)
3. **Target:** Define a flat noise floor target (e.g., $10nV/\sqrt{Hz}$).
4. **Run:** The system minimizes noise density to meet the target.

8 Troubleshooting

- **Xyce Not Found:** Ensure the Xyce binary location is added to your system's `PATH` environment variable. XycLOps II logs the detected path in the sidebar at startup.
- **Slow Startup:** Run `python analyze_startup_time.py` to check if a specific library (like Matplotlib) is causing delays **[XycLOps II Upgrade]**.
- **Convergence Issues:** If the graph flatlines, try enabling "Default Bounds" in Settings or widening the search space for your selected components.