Semester Long Internship Spring 2026

On

eSim Simulation Bridge
A KiCad Plugin Suite for eSim/NgSpice Simulation

Submitted by

Imran Farhat
B.Tech CSE (AI & ML), 2nd Year
VIT Bhopal University

Under the guidance of

Prof. Prabhu Ramachandran
Principal Investigator
Department of Aerospace Engineering
Indian Institute of Technology Bombay

May 14, 2026

Acknowledgment

I express my sincere gratitude to Prof. Prabhu Ramachandran for providing me with the opportunity to be part of the FOSSEE internship program and for his continued efforts in promoting open-source engineering tool development. His leadership and vision have been instrumental in fostering meaningful student participation in the open-source ecosystem.

I also acknowledge Prof. Kannan M. Moudgalya for his foundational role in establishing and strengthening the FOSSEE initiative. His contributions to open-source education and the development of the FOSSEE fellowship framework have been pivotal in creating the academic and organisational platform through which this internship was undertaken.

My sincere appreciation extends to my mentor, Sumanto Kar, for his continual support, technical guidance, and encouragement throughout the duration of this project. His insights and feedback played a key role in refining ideas, overcoming challenges, and ensuring timely completion of the tasks assigned to me.

I would also like to thank my internal mentors, Mr. Varad Patil and Ms. Shanthi Priya K for their valuable guidance, coordination, and technical inputs during the internship. Their mentorship contributed significantly to the clarity, progress, and successful execution of the work.

This internship has been an enriching learning experience, allowing me to work closely with open-source EDA tools, develop IC subcircuits in eSim, and gain exposure to realworld circuit modeling and simulation workflows. The knowledge acquired during this period will undoubtedly support my future academic and professional pursuits.

I would also like to thank the entire FOSSEE team for their coordination, assistance, and timely interactions at various stages of this work. Their collective efforts ensured smooth workflow, resource accessibility, and effective project execution.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Electronic Design Automation (EDA) tools form the backbone of modern hardware engineering education and professional circuit design. In the open-source academic ecosystem, two tools stand out as widely adopted standards: **KiCad** for schematic capture and PCB design, and **eSim** for SPICE-based analog circuit simulation. Despite serving complementary purposes, no automated bridge existed between them. A student who finishes drawing a schematic in KiCad must follow a lengthy manual process - exporting a netlist, converting it to SPICE format, resolving component model dependencies, and navigating eSim's project structure - before any simulation can begin.

The FOSSEE project at IIT Bombay maintains eSim as a freely accessible simulation environment for the Indian academic community under the National Mission on Education through ICT (NMEICT) [11]. KiCad 8.0 introduced a stable Python scripting API [4] that allows developers to register action plugins appearing as toolbar buttons within the application. This creates a precise opportunity to automate the complete KiCad-to-eSim workflow entirely in user-space Python, without modifying either application's source code.

This internship (KiCad Plugin Development) addresses this gap by delivering a production-quality plugin that reduces the entire simulation pipeline to a single button click inside KiCad.

1.2 Overview of eSim

eSim is a free and open-source EDA tool developed by the FOSSEE project at IIT Bombay [1]. It integrates KiCad (for schematic entry) and ngspice (for SPICE simulation) within a Python-based frontend, and is distributed under the GNU General Public Licence. eSim 2.5, used throughout this project, ships with:

- A `deviceModelLibrary/` containing 61 device model files covering diodes, BJTs, MOSFETs, JFETs, IGBTs, and LEDs.
- A `SubcircuitLibrary/` with 586 subcircuit folders (1,317 total files) covering op-amps, 555 timers, voltage regulators, and 74-series logic ICs.

- A built-in KiCad-to-Ngspice converter with a five-tab dialog for analysis configuration, source specification, device modeling, and subcircuit linking.
- A Python-based plotting engine for visualising simulation results.

eSim runs on Linux (Ubuntu 24.04 in the VirtualBox environment used for this project) and its source is publicly available at `https://github.com/FOSSEE/eSim`.

1.3 Objectives of the Project

The primary goals of this internship were:

1. Develop the **eSim Simulation Bridge** plugin (v1.0.0): a one-click KiCad plugin that exports the schematic netlist, converts it to a SPICE deck, resolves component models automatically, generates the eSim project structure, and launches eSim - all from a single toolbar button inside KiCad.
2. Implement a SPICE Model Auto-Linker module (`esim_spice_linker.py`) within the plugin that scans eSim's open-source model library, matches schematic components to available models, and injects the correct definitions into the SPICE file.
3. Provide an interactive **waveform viewer** embedded inside KiCad's GUI supporting transient, AC, noise, FFT, Bode plot, cursor measurement, and parametric sweep visualisation.
4. Integrate eSim's **Python plot window** (`ngspiceSimulation` package) into the plugin, launching it directly from the simulation dialog.
5. Mirror eSim's own six-tab KiCad-to-Ngspice converter dialog, giving users the same level of control as the native eSim application.
6. Validate the plugin on real FOSSEE research migration circuits: a Three-Phase Full-Wave Diode Bridge Rectifier with LC Filter, and a Two-Stage Stagger-Tuned Amplifier.
7. Maintain a documented, GPL-3.0-licensed codebase suitable for contribution to the FOSSEE ecosystem.

1.4 Methodology

The project was executed iteratively over the Spring 2026 semester following an incremental development approach. Each feature was implemented, tested on a real KiCad schematic, and refined before the next feature was added. The methodology covered five distinct phases:

Phase 1 - Study and Planning. The first two weeks were spent studying KiCad 8.0's Python scripting API (`pcbnew` module), eSim 2.5's source code structure, and ngspice's batch-mode operation. The eSim `KicadToNgspice` converter, `DeviceModel` module, and `SubcircuitLibrary` were examined to understand exactly what the plugin needed to replicate and extend.

Phase 2 - Core Plugin Development. The `ESimBridgePlugin` (`esim_bridge.py`) was built incrementally: first the netlist export via `kicad-cli`,

then the S-expression parser, then the SPICE converter handling 14 component prefixes, then the six-tab `KicadToNgspiceDialog` mirroring eSim's own interface, and finally the preflight netlist checker and eSim project structure generator.

Phase 3 - Model Library and Auto-Linker. The built-in `SPICEModelLibrary` (47+ hardcoded models) and `ExternalModelLoader` were added to the core plugin. The separate `esim_spice_linker.py` module was then developed to scan eSim's 647-file open-source library, implementing the five-tier scored matching algorithm with false-positive prevention. This module is imported lazily inside `Run()` to avoid KiCad startup timeouts.

Phase 4 - Waveform Viewer and Python Plot Integration. The embedded `NgspiceWaveformViewer` (wxPython + matplotlib) was developed with transient, FFT, Bode plot, parametric sweep, cursor measurement, and a runtime stats panel reading directly from ngspice's binary `.raw` output. eSim's existing `ngspiceSimulation` Python plot window package was then integrated via `importlib` dynamic import, providing a second visualisation path for users familiar with eSim's native interface.

Phase 5 - Validation and Testing. Two FOSSEE Research Migration Project circuits were used as integration tests: (1) a Three-Phase Full-Wave Diode Bridge Rectifier with LC Filter [2] (Purven Khadke, Marathwada Mitra Mandal College of Engineering), exercising multi-source handling, diode model resolution, and transient/FFT analysis; and (2) a Two-Stage Stagger-Tuned Amplifier [3] (Konda Manasa, Kakatiya Institute of Technology and Science, Warangal), exercising BJT model resolution, AC analysis, and the Bode plot viewer.

Key technical choices made during the project:

- **Language and framework:** Python 3 with wxPython for the KiCad-side GUI and PyQt5 for the Python plot window, interfacing with KiCad 8.0's `pcbnew` scripting API.
- **Netlist format:** KiCad S-expression format (`kicadsexpr`), exported via `kicad-cli` and parsed with regular expressions to extract component references, values, net connections, and simulation properties (`Sim.Type`, `Sim.Params`).
- **Model resolution:** A five-tier search hierarchy - eSim's deviceModelLibrary, eSim's SubcircuitLibrary, user-provided external models, a known-equivalents table, and a last-resort textbook model generator based on published device parameters [8] [9].
- **Accuracy principle:** All numerical values displayed in the stats panel are read directly from ngspice's binary `.raw` output at runtime. No hardcoded circuit-specific defaults are used anywhere in the plugin.
- **Version control:** GPL-3.0-licensed codebase at `https://github.com/ImranFarhat01/eSim-KiCad-Plugin/tree/main/ImranFarhat_eSim_Simulation_Bridge`.

Chapter 2

Literature Survey

SPICE (Simulation Program with Integrated Circuit Emphasis) was first developed at UC Berkeley in 1973 and remains the global standard for analog and mixed-signal circuit simulation. Its open-source descendant, **ngspice** (version 42) [5], implements the full SPICE3 model set including BJTs, MOSFETs, JFETs, diodes, and controlled sources.

KiCad has grown over three decades into a professional-grade EDA suite maintained by the KiCad project community and CERN. Its Python scripting interface, stabilised in version 7 and extended in version 8, allows developers to register `ActionPlugin` subclasses that appear as menu items and toolbar buttons. While many published plugins target PCB layout automation, schematic-oriented plugins that bridge to simulation tools remain rare.

eSim was developed by the FOSSEE team at IIT Bombay as a fully open-source alternative to proprietary simulation tools for Indian academic institutions. Its built-in KiCad-to-Ngspice converter - a five-tab dialog covering analysis selection, source waveform configuration, device model linking, subcircuit management, and microcontroller support via NGHDL - served as the direct design reference for the dialog implemented in eSim Simulation Bridge.

The **FOSSEE Research Migration Project** [1] publishes open-source reproductions of published circuit designs. Two circuits from this collection were used as validation test cases: the Three-Phase Full-Wave Diode Bridge Rectifier [2] (Rashid, Power Electronics Handbook, 2011 [6]) and the Two-Stage Stagger-Tuned Amplifier [3] (Taghavi et al., IEEE TVLSI, 2016 [7]).

In the broader open-source EDA ecosystem, tools such as **xschem** (with ngspice integration via `.spiceinit`) and **Qucs-S** provide simulation-oriented schematics but do not interoperate with KiCad's native `.kicad_sch` format. Commercial tools such as LTspice, Altium Designer, and Cadence Virtuoso implement tight EDA-to-simulation integration but are proprietary and unavailable to most Indian students. The present work fills this gap specifically for KiCad 8.0 and eSim 2.5, and is among the first to implement a full simulation bridge as a KiCad action plugin.

Chapter 3

Problem Statement

3.1 Problem Statement

Despite both being free, open-source, and widely used in Indian academic institutions, KiCad and eSim/ngspice require the following manual steps to bridge a schematic to a simulation result:

1. Open the schematic in KiCad and annotate all components.
2. Export a netlist via *File* → *Export* → *Netlist* in KiCad format.
3. Open eSim, create a new project, and copy the netlist into the project folder.
4. Open eSim's KiCad-to-Ngspice converter, manually configure the analysis type, specify source waveform parameters, and link device model `.lib` files for every active component (diode, transistor, MOSFET, IC).
5. Run the simulation and switch to eSim's separate Python plotting window to visualise results.
6. If a component model is missing, locate a manufacturer SPICE model online, download it, and repeat the entire process from step 4.

This workflow presents several serious pain points for students and researchers:

- **High manual overhead:** Each simulation iteration requires 6 or more steps even for simple circuits.
- **Model resolution burden:** Users must independently locate, download, and validate SPICE models for every active component.
- **No real-time feedback:** The student cannot see simulation results from within KiCad and must context-switch between three separate applications.
- **Error-prone conversion:** Manual netlist editing introduces mistakes that produce cryptic ngspice errors such as *singular matrix* or *node not found* with no guidance on how to fix them.
- **Accessibility barrier:** The multi-step process discourages beginners from attempting simulation at all, undermining the educational goal of eSim.

3.2 Approach

The approach adopted directly addresses each pain point above:

Automation of the full workflow: eSim Simulation Bridge provides a single toolbar button inside KiCad that triggers the complete pipeline - netlist export, SPICE conversion, model resolution, project setup, and eSim launch - with zero manual file management.

Automatic model library integration: the SPICE Model Auto-Linker module scans eSim's 647-file open-source model library at startup and builds a searchable in-memory index. For every active component, it performs a five-tier search and either injects the correct model or generates an approximate textbook model, reporting the outcome in a clear status table before simulation begins.

Embedded waveform viewer: An oscilloscope-style waveform viewer is embedded directly inside KiCad's GUI, supporting transient, AC, noise, FFT, Bode plot, cursor measurement, and parametric sweep, with no context switching required.

Preflight netlist checking: Before any simulation is attempted, the plugin performs automated checks for missing ground nodes, floating nets, voltage source conflicts, and capacitor-only DC paths, with human-readable error messages and actionable fix suggestions.

Mirroring eSim's own interface: The six-tab KiCad-to-Ngspice dialog replicates eSim's native converter so that students already familiar with eSim feel at home immediately.

Chapter 4

Implementation

The **eSim Simulation Bridge** is a single KiCad action plugin installed under: `~/.local/share/kicad/8.0/scripting/plugins/esim_bridge/`.

It is organised into two main Python modules and one integrated package:

- `esim_bridge.py` (v1.0.0) - the core plugin: `ActionPlugin` registration, netlist export, SPICE conversion, six-tab dialog, waveform viewer, preflight checker, and eSim project generation.
- `esim_spice_linker.py` (v1.0.0) - the SPICE Model Auto-Linker: scans eSim's built-in library and resolves device models automatically. Imported lazily inside `Run()` to avoid KiCad startup timeouts.
- `ngspiceSimulation/` - the Python plot window package (`plot_window.py`, `data_extraction.py`, `plotting_widgets.py`): integrated into the plugin and launched dynamically via `importlib` when the user clicks "Open Python Plot".

4.1 Supported Circuit Types and Scope

eSim Simulation Bridge supports simulation of any circuit that can be processed by standalone ngspice. This is a broader scope than "analog only" - the correct boundary is **ngspice-simulatable vs. not**.

Supported circuit types:

- Analog circuits: RC, RLC, op-amp based, BJT/MOSFET amplifiers, filters, oscillators, power converters (including multi-phase rectifiers as demonstrated in Chapter 5).
- Frequency-domain circuits: stagger-tuned amplifiers, active filters, any circuit requiring AC, noise, or transfer function analysis.
- Transistor-level digital circuits: CMOS inverters, NAND gates, and similar circuits where a valid SPICE `.model` or `.subckt` exists.
- Circuits using standard SPICE sources: DC, AC, sine, pulse, PWL, exp.

Unsupported circuit types:

- Circuits using eSim's mixed-signal co-simulation components - specifically `eSim_Ngveri` and `eSim_Hybrid` library blocks such as `adc_bridge`,

`dac_bridge`, and behavioral Verilog models generated through eSim's NgVeri flow. These components require eSim's internal co-simulation engine (ngspice + Verilator) and have no standalone SPICE subcircuit representation. Attempting to simulate such circuits will result in a *"missing subcircuit model"* error from ngspice.

- Circuits whose required `.subckt` models do not exist in eSim's library, the external model folder, or the built-in textbook library.
- Standard digital ICs (74xx TTL series, MCUs, FPGAs) - no transistor-level ngspice models exist for these in practice.

This is not a limitation of the plugin design but a fundamental architectural constraint of ngspice itself, which does not support behavioral digital co-simulation natively. eSim handles mixed-signal circuits through a separate co-simulation engine (NGHDL/Verilator) that operates outside the ngspice batch mode used by eSim Simulation Bridge. The Microcontroller tab in the six-tab dialog acknowledges this and documents the NGHDL pathway for users who require it.

4.2 Plugin Architecture and Workflow

The overall architecture of the eSim Simulation Bridge plugin is shown in Figure 4.1. The workflow proceeds in seven clearly defined stages, from schematic capture in KiCad through to interactive waveform visualisation.

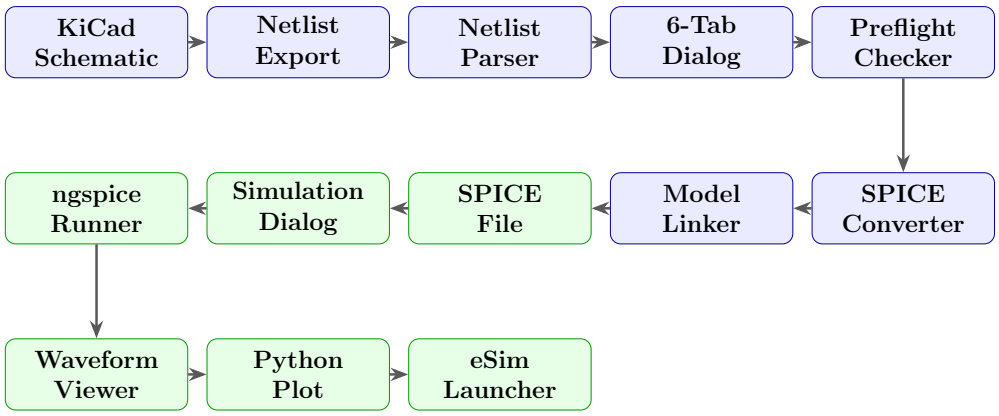

Figure 4.1: Architecture and workflow of the eSim Simulation Bridge plugin. **Row 1 (left → right):** KiCad schematic → netlist export → parser → 6-tab dialog → preflight checker. **Row 2 (right → left):** SPICE converter → model linker → SPICE file → simulation dialog → ngspice runner. **Row 3 (left → right):** waveform viewer → Python plot → eSim launcher.

4.3 eSim Simulation Bridge Run() Execution Flow

Figure 4.2 shows the step-by-step execution flow inside the `Run()` method of `ESimBridgePlugin` when the user clicks the toolbar button. The decision diamond

at the top routes execution based on whether a valid schematic file is found.

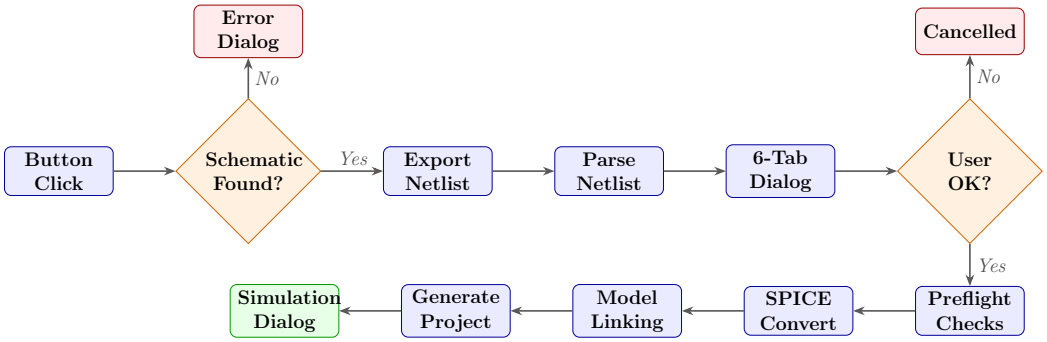

Figure 4.2: Execution flow of `Run()` in `ESimBridgePlugin`. **Top row:** button click → schematic check → netlist export → parse → 6-tab dialog → user confirmation. **Bottom row:** preflight → SPICE conversion → model linking → project generation → simulation dialog. Diamonds are decision points.

4.4 SPICE Model Resolution Flow

Figure 4.3 shows the internal processing pipeline of the `SPICEAutoLinker` and `ModelMatcher` classes when resolving SPICE models for each component in the schematic.

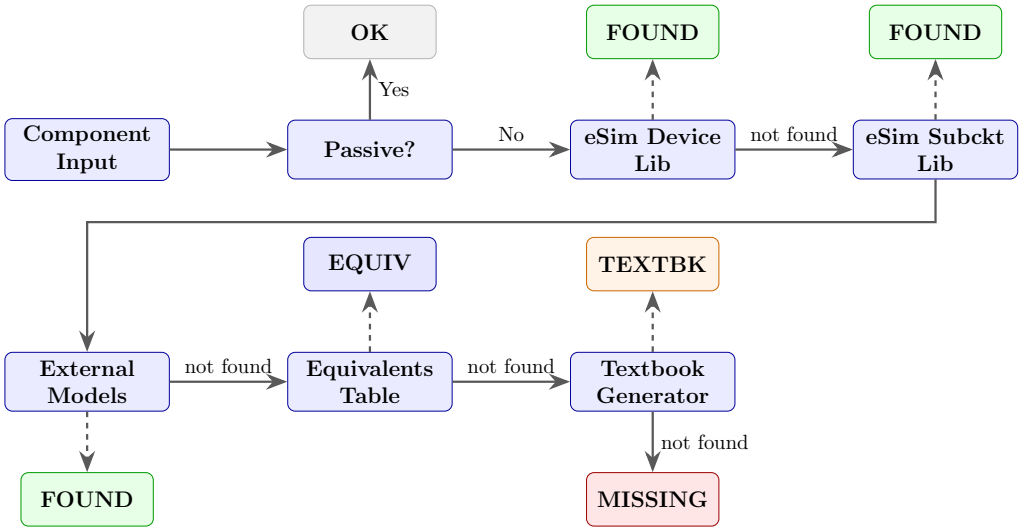

Figure 4.3: Five-tier SPICE model resolution pipeline in `SPICEAutoLinker`. Tiers searched left to right: eSim device library, eSim subcircuit library, external user models, equivalents table, textbook generator. **OK**: passive/source (no model needed). **FOUND**: matched in eSim library. **EQUIV**: known equivalent substitution. **TEXTBK**: textbook approximate model. **MISSING**: no model found anywhere.

4.5 Case 1: Core Plugin (esim_bridge.py, v1.0.0)

4.5.1 Plugin Registration and Entry Point

eSim Simulation Bridge is registered with KiCad as an `ActionPlugin` subclass. When the user clicks its toolbar button, the `Run()` method executes the complete simulation pipeline in a single invocation. The method locates the schematic associated with the currently open PCB file by substituting the `.kicad_pcb` extension for `.kicad_sch`, or falls back to a file browser dialog if no schematic is found automatically.

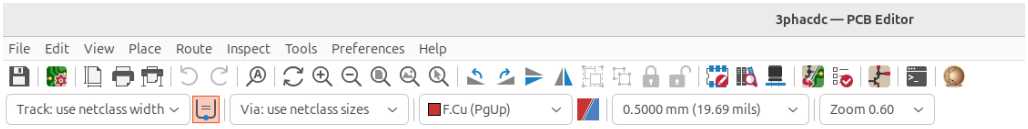

Figure 4.4: eSim Simulation Bridge plugin toolbar button visible in KiCad 8.0. The button appears in the top toolbar after KiCad loads the plugin from the scripting directory.

4.5.2 Netlist Export and Parsing

The schematic netlist is exported silently using `kicad-cli`:

```
kicad-cli sch export netlist \
--output /tmp/esim_bridge_netlist.net \
--format kicadsexpr \
schematic.kicad_sch
```

Listing 4.1: Netlist export via kicad-cli

The resulting S-expression file is then parsed by `SPICEConverter.parse_full_netlist()`. This method uses regular expressions to extract component reference designators, values, descriptions, library names, footprints, simulation properties (`Sim.Type`, `Sim.Params`), net names, and the pin connections of every component. All net names are sanitised for SPICE compatibility (special characters replaced, GND mapped to node 0, names capped at 20 characters).

4.5.3 The Six-Tab KiCad-to-Ngspice Dialog

After netlist parsing, eSim Simulation Bridge presents the `KicadToNgspiceDialog` - a tabbed wxPython dialog that mirrors eSim's own KiCad-to-Ngspice converter.

Tab 1 - Analysis: Single-select checkboxes for AC, DC, Transient, Noise, Transfer Function, and Sensitivity analysis. Each type reveals its own parameter group (frequency range, time step, source, etc.). All values are automatically saved to `~/.esim-bridge/KicadToNgspice_Previous_Values.xml` and restored on the next session.

Tab 2 - Source Details: For every voltage or current source detected in the schematic, a collapsible group box is rendered. The user can override the source

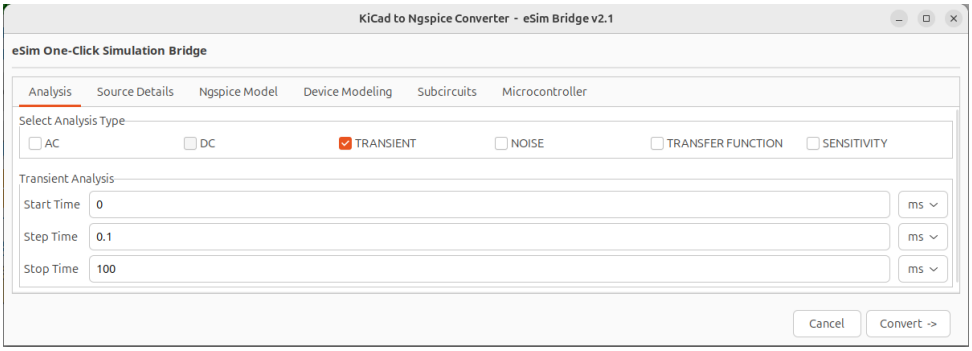

Figure 4.5: The KicadToNgspiceDialog showing the Analysis tab. The Transient checkbox is selected and its parameter group (Start Time, Step Time, Stop Time) is visible. The other analysis types (AC, DC, Noise, Transfer Function, Sensitivity) are shown as unselected checkboxes.

type (DC, AC, sine, pulse, PWL, exp) and enter waveform parameters. The `sine` type exposes offset, amplitude, frequency, delay time, and damping factor. The Three-Phase Rectifier circuit uses three `SIN` sources (V1, V2, V3) with 120° phase offsets, all configured from this tab.

Tab 3 - Ngspice Model: For `U`-prefix components, eSim's `modelParamXML` directory is scanned for a matching XML parameter definition. If found, a grid of labelled text fields is rendered that matches eSim's own Model editor, and the values are assembled into a `.model` line on conversion.

Tab 4 - Device Modeling: For Q/D/J/M/S components, a file picker allows the user to select a manufacturer `.lib` file. MOSFET components additionally expose width (W), length (L), and multiplier factor fields. A SKY130 PDK mode is automatically engaged when SKY130 component values are detected in the schematic.

Tab 5 - Subcircuits: For `X`-prefix components, a directory picker allows the user to select a subcircuit folder. Port count validation is performed against the component's actual pin count before the selection is accepted.

Tab 6 - Microcontroller: Detects MCU-type components (ATtiny, Arduino, ATmega, PIC, STM32, ESP) by value keyword. Checks whether NGHDL is installed, displays its home path if found, and provides a hex file picker for loading compiled firmware.

4.5.4 SPICE Conversion Engine

The `SPICEConverter` class converts every parsed component into its SPICE representation. The following table summarises the conversion rules:

For active devices (D, Q, M, J), the converter follows a five-tier model search: (1) user-selected `.lib` from Tab 4, (2) external user models from `~/.esim-bridge/models/`, (3) the eSim-SPICE library scanner, (4) eSim Simulation Bridge's built-in `SPICEModelLibrary` (47+ textbook models), (5) `TextbookModelGenerator` as a last resort.

Table 4.1: Component prefix to SPICE conversion mapping

Prefix	Type	Example SPICE Output
R	Resistor	`R1 net1 net2 10k`
C	Capacitor	`C1 net1 net2 470u`
L	Inductor	`L1 net1 net2 10m`
V	Voltage source	`V1 net1 0 SIN(0 325 50 0 0 0)`
I	Current source	`I1 net1 0 DC 1m`
D	Diode / LED	`D1 anode cathode D1N4148`
Q	BJT transistor	`Q1 coll base emit Q2N2222`
M	MOSFET	`M1 drain gate src bulk MNMOS`
J	JFET	`J1 drain gate src JNFET`
U/X	IC / Subcircuit	`XU1 in out vcc vee LM741`
BT	Battery	`VBT1 + - DC 3.7`
SW	Switch	`RSW1 net1 net2 1` (1 Ω)
MK	Microphone	modelled as 10 mV AC source at 1 kHz
F	Fuse	modelled as 0.01 Ω resistor

4.5.5 Built-in SPICE Model Library

The `SPICEModelLibrary` class encodes textbook-standard SPICE parameters for the most common components used in educational circuits:

- **Diodes:** 1N4148, 1N4001-4007, 1N5817/5819, BZT52 Zener series, generic LED, generic Zener. The Three-Phase Rectifier uses six 1N4148 diodes, all resolved automatically from this library.
- **NPN BJTs:** 2N2222, 2N3904, BC547/548, 2N2219, TIP31. The Stagger-Tuned Amplifier uses 2N2222 transistors, matched here with full Sedra/Smith [8] and Boylestad [9] parameters.
- **PNP BJTs:** 2N3906, 2N2907, BC557/558, TIP32.
- **N-MOSFETs:** IRF540/540N, IRF3205, IRF830, 2N7000/7002, BS170.
- **P-MOSFETs:** IRF9540, BS250.
- **Op-amp subcircuits:** LM741, UA741, LM358, LM324, generic ideal op-amp.
- **555 timer:** NE555 simplified behavioural subcircuit.
- **Voltage regulators:** 7805, 7812, 78L33 simplified subcircuits.

4.5.6 External Model Loader

The `ExternalModelLoader` class scans `~/.esim-bridge/models/` for user-provided SPICE model files. This directory is created automatically on first run with a `README.txt` explaining the expected format. Supported extensions are `.lib`, `.mod`, `.sub`, `.spice`, `.cir`, and `.model`. All `.model` and `.subckt` definitions found in these files are indexed in memory and participate in the component matching pipeline automatically.

4.5.7 Preflight Netlist Checker

Before the simulation-ready dialog is shown, `PreflightChecker` runs five categories of automated netlist validation:

1. **Ground node presence:** Verifies at least one net maps to SPICE node 0.
2. **Floating node detection:** Identifies nets connected to only one component pin, which causes ngspice's matrix solver to fail with a *singular matrix* error.
3. **Voltage source conflict:** Detects pairs of voltage sources sharing both terminals, creating an unsolvable voltage loop.
4. **Orphaned components:** Warns about components with no net connections.
5. **Capacitor-only DC paths:** Warns about nets with no DC path to ground.

Additionally, the checker writes `set ngbehavior=ps` to `~/.spiceinit` if not already present, enabling SPICE-format model compatibility in ngspice.

4.5.8 eSim Project Structure Generation

eSim expects a specific directory layout under `~/eSim-Workspace/`*`project_name`*`/`. eSim Simulation Bridge replicates this structure automatically:

```
~/eSim-Workspace/esim_bridge_project/
esim_bridge_project.cir        (SPICE source)
esim_bridge_project.cir.out    (SPICE + .control block)
esim_bridge_project.proj       (eSim project marker)
analysis                       (analysis command string)
images/                        (for eSim plot exports)
*.sub / *.lib                  (copied dependency files)
```

The `.cir.out` file contains the full SPICE deck augmented with a `.control` block that includes the analysis command, `run`, `rusage all`, `print allv`, `print alli`, and file output redirections - the exact format expected by eSim's simulation engine.

4.5.9 Analysis-Specific Implementations

Transient Analysis: Control block contains `tran <step> <stop> <start>`. Results are displayed in the embedded waveform viewer. Used for the Three-Phase Bridge Rectifier to observe the rectified output waveform and LC filter smoothing.

AC Analysis: Control block contains `ac dec|lin|oct <points> <fstart> <fstop>`. The viewer automatically offers a Bode plot button. Used for the Stagger-Tuned Amplifier to extract the frequency response and observe the staggered gain peaks.

DC Sweep: Control block contains `dc <source> <start> <stop> <step>`, optionally with a second source for a nested sweep.

Operating Point (OP): Runs `op` via ngspice in a subprocess, parses DC node voltages from stdout, and shows them in a message box.

Transfer Function: Runs `tf v(<output>) <source>` and displays gain, input impedance, and output impedance with plain-language interpretations.

Sensitivity Analysis: Runs `op` followed by `sens <output>`, sorts components by absolute sensitivity value, and shows the ranking with directional arrows.

Noise Analysis: Runs `noise v(<output>) <source> dec <points> <fstart> <fstop>`, extracts `inoise_total` and `onoise_total`, converts to μV or nV, and explains the noise reduction ratio.

4.5.10 Embedded Waveform Viewer

Clicking "Run with ngspice" in the simulation-ready dialog runs ngspice in batch mode and parses the resulting `.raw` binary file using `NgspiceRawParser`, which supports both ASCII and binary formats (IEEE 754 doubles, row-major, optionally complex pairs for AC).

The `NgspiceWaveformViewer` wxPython dialog then renders the results with a dark oscilloscope-style matplotlib [12] figure. Key features are:

- **Trace toggles:** Colour-coded checkboxes for each variable; toggling hides or shows the trace with an immediate redraw. A "Select All" checkbox toggles all traces simultaneously.
- **Cursor measurement:** Left-click places cursor 1 (red), right-click places cursor 2 (blue). A crosshair annotation shows time/value at the cursor position. Cursors auto-scale to the current time unit (ns, μs, ms, s).
- **FFT viewer:** For transient results, `numpy.fft.rfft` is computed on each visible trace and the magnitude spectrum is plotted. Stats include DC component, dominant frequency, peak magnitude, and THD.
- **Bode plot:** For AC results, two subplots (gain in dB and phase in degrees, both with semilog frequency axis) are rendered. The stats panel shows only verified-correct values: gain at lowest and highest frequency, peak gain with its frequency, minimum gain, and total gain swing.
- **Parametric sweep:** The user selects a component and either enters comma-separated values or a start/stop/steps range. For each step, a modified `.cir.out` is written and ngspice is run in batch mode. All traces are overlaid with distinct colours and line styles (supports 2-10 steps).
- **Runtime stats panel:** A monospaced text area shows numerical summaries read from the `.raw` binary at runtime. Every displayed value (Peak, Max, Min, Average, Peak-to-Peak, RMS, Frequency, interpretation) is derived from actual simulation output with no hardcoded defaults.
- **Legend toggle:** The "Legend" button shows or hides the trace legend without re-running the simulation.
- **PNG export:** Saves the current figure at 150 DPI to a user-specified path.

4.5.11 SPICEAutoLinker and SimulationReadyDialog

The `SPICEAutoLinker` class in `esim_spice_linker.py` is the single public entry point called by eSim Simulation Bridge after SPICE conversion. It instantiates `ESimLibraryScanner`, `ExternalModelLoader`, and

ModelMatcher, then exposes three public methods: check_models(), show_report(), and get_injection_data(). After model data is returned, converter._rewrite_with_models() appends resolved definitions to the .cir file, skipping duplicates already present.

After conversion succeeds, the SimulationReadyDialog gives the user a final review with a scrollable SPICE file preview and four action buttons: **Launch eSim**, **Run with ngspice**, **Open Python Plot**, and **Open .cir File**.

4.5.12 Logging, Stale File Cleanup, and Session Management

Three housekeeping behaviours in ESimBridgePlugin.Run() are worth documenting:

Logging: On every invocation, Run() configures Python's standard logging module to write timestamped DEBUG-level entries to ~/.local/share/kicad/esim_bridge.log. This allows post-mortem debugging of netlist parse errors and ngspice failures without requiring the user to reproduce the issue interactively.

Stale file cleanup: Before any simulation begins, Run() deletes leftover output files from the previous session (plot_data_v.txt, plot_data_i.txt, .raw, .cir.out) from the eSim project folder. This prevents the Python plot window from loading stale data from a previous circuit when the user has changed the schematic.

Appconfig integration: The Python plot window uses eSim's Appconfig class (from configuration/Appconfig.py) for error message display and logging, maintaining compatibility with eSim's own project management infrastructure. QSettings persistence (via 'eSim'/'PythonPlotting') stores window state across sessions.

4.5.13 Dynamic PyQt5 Launch from within KiCad's wxPython Environment

A notable engineering challenge is that KiCad 8.0 runs its own embedded wxPython interpreter, while the Python plot window requires PyQt5. The _launch_plot_window() method solves this by loading the ngspiceSimulation package at runtime using importlib.util.spec_from_file_location(), constructing a synthetic package namespace (sys.modules['ngspiceSimulation']) to satisfy relative imports, obtaining a QApplication instance via QApplication.instance(), and verifying that all three required data files exist before launching.

Figure 4.6: The `SimulationReadyDialog` showing the auto-generated SPICE file for the Three-Phase Bridge Rectifier. The preview panel displays the component lines (D1-D6, C1, L1, R1, R2, V1-V3), auto-injected diode model (`.model D_ESIM_DIODE`), and the `.control` block with the transient analysis command. The four action buttons are visible at the bottom right.

4.6 Case 2: SPICE Model Auto-Linker Module (esim_spice_linker.py, v1.0.0)

4.6.1 Overview

The SPICE Model Auto-Linker is a module providing automatic SPICE model resolution from eSim's own open-source library. It operates as a transparent layer between the parsed netlist and the generated SPICE file. The user interacts with it only through the `ModelStatusReport` dialog, which appears automatically when eSim Simulation Bridge runs.

4.6.2 ESimLibraryScanner

The `ESimLibraryScanner` class scans eSim's two model library directories:

deviceModelLibrary/: Contains 61 device model files in subdirectories (Diode, Transistor, MOS, JFET, IGBT, LEDs, Switch, Misc). Each `.lib` file is parsed for `.model` definitions using a regex that handles multi-line continuation (`+`) syntax. Every model is indexed by a clean alphanumeric key.

SubcircuitLibrary/: Contains 586 subcircuit folders totalling 1,317 files. Each folder contains a `*-cache.lib` file and dependency `.lib` files. The scanner indexes both the subcircuit name and the folder name as independent lookup keys.

4.6.3 Scored Matching Algorithm

Rather than simple substring search, the scanner uses a scoring system:

Table 4.2: Match priority scores in ESimLibraryScanner

Score	Match type
100	Exact alphanumeric key match
90	Exact folder name match
88	Folder name with IC prefix stripped (SN, CD, MC, DM)
80	Both names equal after common prefix removal
60	One name is a prefix of the other (≥ 4 characters each)
< 60	Rejected - no match

A blacklist of generic keys (`nmos`, `npn`, `d`, `sw`, `core`, `buffer`, etc.) prevents false-positive matches on model type names that appear in many different library files.

4.6.4 Four-Tier Model Search

The `ModelMatcher` class orchestrates the search for every component:

1. eSim `deviceModelLibrary` via `ESimLibraryScanner`.
2. eSim `SubcircuitLibrary` via `ESimLibraryScanner`, including automatic dependency file resolution.
3. User's `~/.esim-bridge/models/` via `ExternalModelLoader`.

4. Known equivalent substitution table (e.g., BC548 to BC547, 1N4001 to 1N4007).
5. `TextbookModelGenerator` last-resort approximate `.model` card from published textbook parameters (Sedra/Smith [8], Boylestad [9], Millman/Halkias [10]).

Passive components (R, C, L) and sources (V, I) are immediately classified as "no model needed" and skip the search pipeline entirely.

4.6.5 ModelStatusReport Dialog

After matching all components, a wxPython dialog presents a colour-coded table:

Table 4.3: Model status codes shown in the report dialog

Status	Colour	Meaning
FOUND	Green	Model available in eSim's open-source library
EQUIV	Blue	Using a known compatible equivalent
TEXTBK	Amber	Generated from textbook parameters (approximate)
MISSING	Red	No model found; manual action needed
OK	Grey	Passive or source - no model required

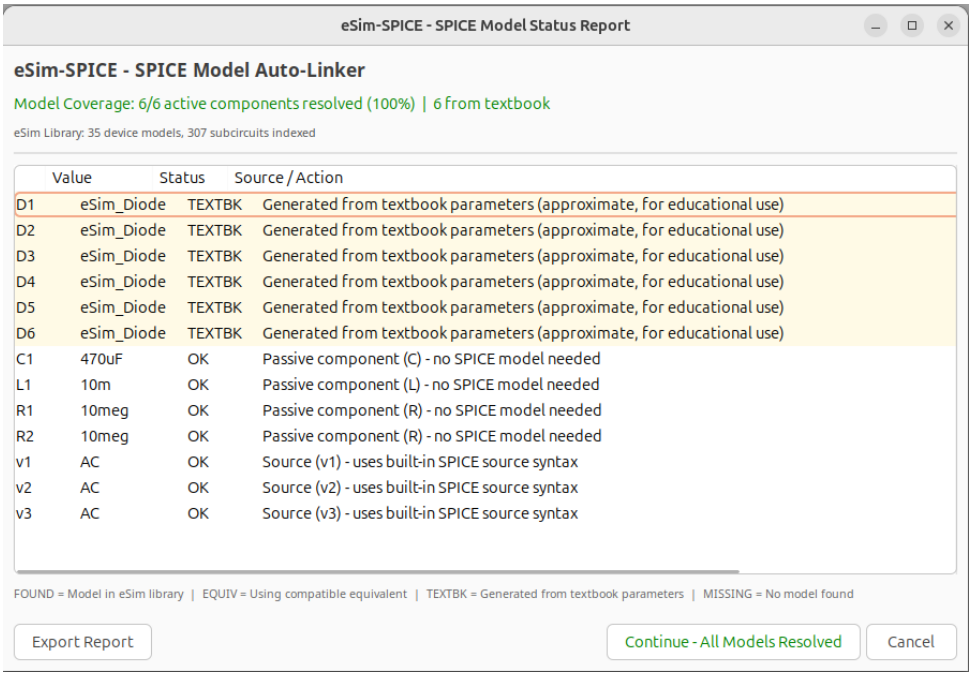

Figure 4.7: The eSim-SPICE `ModelStatusReport` dialog for the Three-Phase Bridge Rectifier. Diodes D1-D6 (1N4148) are shown as FOUND (green) - resolved from eSim's `deviceModelLibrary`. Passive components R1, R2, C1, L1 and voltage sources V1-V3 are shown as OK (grey). The coverage summary confirms 100% model resolution for this circuit.

4.7 Case 3: Python Plot Window Integration (ngspiceSimulation package)

4.7.1 Overview

The **eSim Simulation Bridge** plugin integrates eSim's existing Python plot window as a third visualisation option alongside the embedded wxPython waveform viewer. The `ngspiceSimulation` package - consisting of three source files - is loaded dynamically at runtime when the user clicks "Open Python Plot" in the `SimulationReadyDialog`:

- `plot_window.py` - the main `plotWindow` QWidget class
- `data_extraction.py` - the `DataExtraction` class that reads and parses ngspice text output files
- `plotting_widgets.py` - reusable UI widgets (`CollapsibleBox`, `MultimeterWidgetClass`)

This viewer reads the text output files produced by eSim's ngspice engine (`plot_data_v.txt`, `plot_data_i.txt`, and the `analysis` file), giving full compatibility with eSim's existing project format. The package is part of the same plugin installation directory and is imported dynamically inside `_on_open_python_plot()` to avoid KiCad startup timeouts.

4.7.2 DataExtraction Class

The `DataExtraction` class reads and parses ngspice text output files. `openFile()` reads the `analysis` file to determine whether the simulation was AC, transient, or DC. `numberFinder()` scans `plot_data_v.txt` and `plot_data_i.txt` to determine the number of lines per data partition, the total count of voltage nodes and current branches, and the output format. The `computeAxes()` method extracts the data into `self.x` (time or frequency axis) and `self.y` (a list of lists, one per node/branch), using Python's `Decimal` type for precision.

4.7.3 plotWindow Class and Three-Panel Layout

The `plotWindow` class is a full-featured PyQt5 `QWidget` presenting simulation results in a three-panel layout:

- **Left panel:** Waveform list with search box, select-all/deselect-all buttons, and a `CustomListWidget` showing all node voltages and branch currents.
- **Centre panel:** Matplotlib figure with a navigation toolbar (zoom, pan, home, sub-plot adjust, figure options shortcut).
- **Right panel:** Collapsible control sections for display options, digital timing controls, cursor measurements, and export tools.

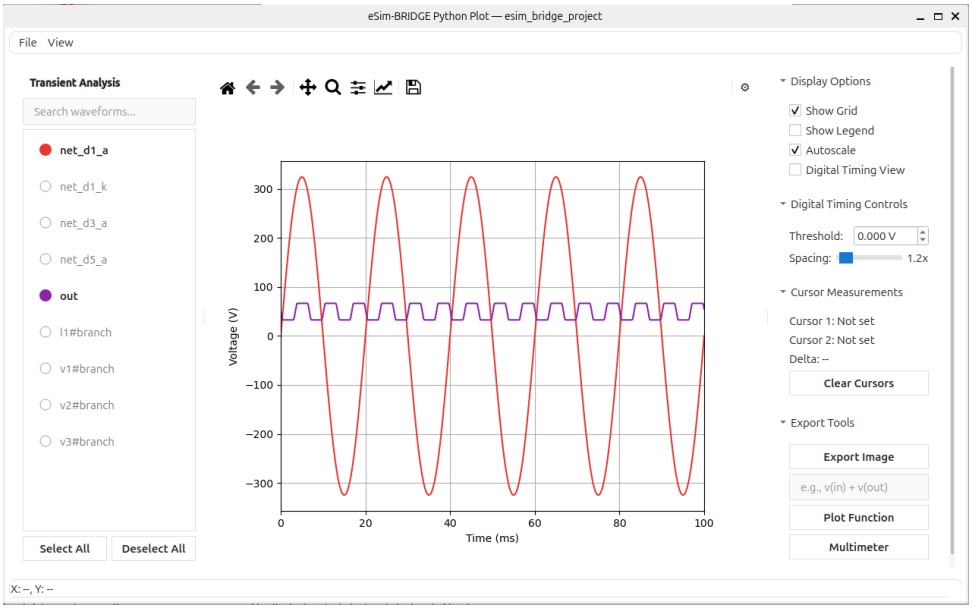

Figure 4.8: The PyQt5 `plotWindow` displaying transient simulation results of the Three-Phase Bridge Rectifier. The left panel shows all node voltages and branch currents (v(out), v(net_d1_a), v(net_d1_k), i(v1) etc.). The centre panel shows the rectified output waveform at v(out). The right panel shows the collapsible control sections.

4.7.4 Waveform Display and Trace Management

Each waveform is represented as a `Trace` object storing the index, display name, colour, line thickness, line style, and visibility state. Colours are assigned from a 12-colour `VIBRANT_COLOR_PALETTE` cycling automatically. Trace styles are persisted to `~/.pythonPlotting/config.json` and restored on the next session. A right-click context menu provides colour change, thickness (1-3 px), line style (solid/dashed/dotted/step), rename, and show/hide options.

4.7.5 Analysis-Specific Rendering

The `refresh_plot()` method dispatches based on the `analysis` file content: **Transient** (`on_push_trans()`) plots voltage/current vs. time with auto-scaled time units; **AC linear** (`on_push_ac()`) and **AC logarithmic** (`on_push_decade()`) plot magnitude vs. frequency; **DC** (`on_push_dc()`) plots vs. the swept source value.

4.7.6 Digital Timing Diagram

When the "Digital Timing View" checkbox is enabled, `plot_timing_diagram()` renders all visible traces as two-level digital waveforms stacked vertically. The logic threshold defaults to $V_{min} + 0.7 \times (V_{max} - V_{min})$ and can be overridden via a `QDoubleSpinBox`. A transient time-offset fix uses `numpy.searchsorted` to trim pre-start data. Each trace is annotated with its name and final voltage value outside the axes boundaries using `clip_on=False`.

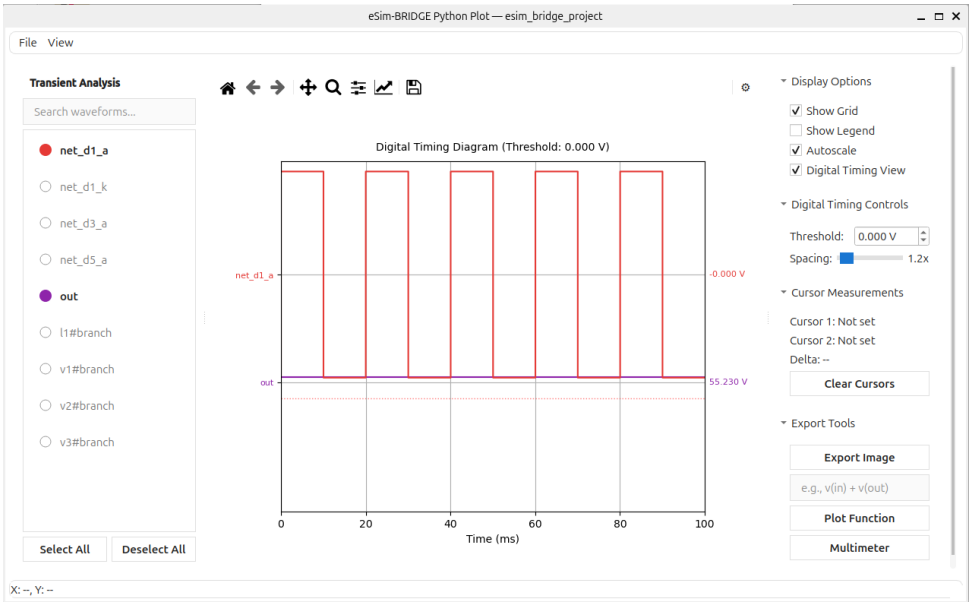

Figure 4.9: Digital timing diagram view in the `plotWindow` for the Three-Phase Bridge Rectifier output. The rectified `v(out)` and one phase input `v(net_d1_a)` are shown as two-level digital signals stacked vertically. The red dotted line marks the auto-detected threshold. Y-axis labels show trace names.

4.7.7 Cursor Measurements, Function Plotting, and Multimeter

Left-clicking the matplotlib canvas places Cursor 1 (red dashed vertical line); right-clicking places Cursor 2 (blue). The control panel displays cursor positions, Δt, and implied frequency. Scroll events implement zoom (with Ctrl) and horizontal pan (with Shift).

The "Plot Function" input field accepts two modes: **ratio mode** (`A vs B`) plots one trace against another; **expression mode** evaluates arbitrary NumPy expressions referencing trace names (e.g., `v(in) - v(out)`).

The "Multimeter" button opens a `MultimeterWidgetClass` floating window showing the RMS value of the selected trace:

$$V_{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} v_i^2}$$

computed with Python's `Decimal` type (5 significant figures). The window stays on top (`Qt.WindowStaysOnTopHint`).

4.7.8 Export and Persistence

The "Export Image" button saves the current matplotlib figure as PNG or SVG (150 DPI, `bbox_inches='tight'`). On close, `save_config()` writes trace preferences to `~/.pythonPlotting/config.json` using an atomic write (`.tmp` then rename) to avoid data corruption.

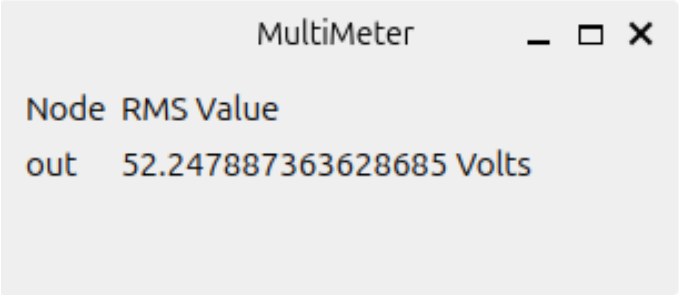

Figure 4.10: The `MultimeterWidgetClass` floating window showing the RMS voltage at the `v(out)` node of the Three-Phase Bridge Rectifier. The expected theoretical RMS of the rectified output is approximately 52.25 V, consistent with the simulation result shown.

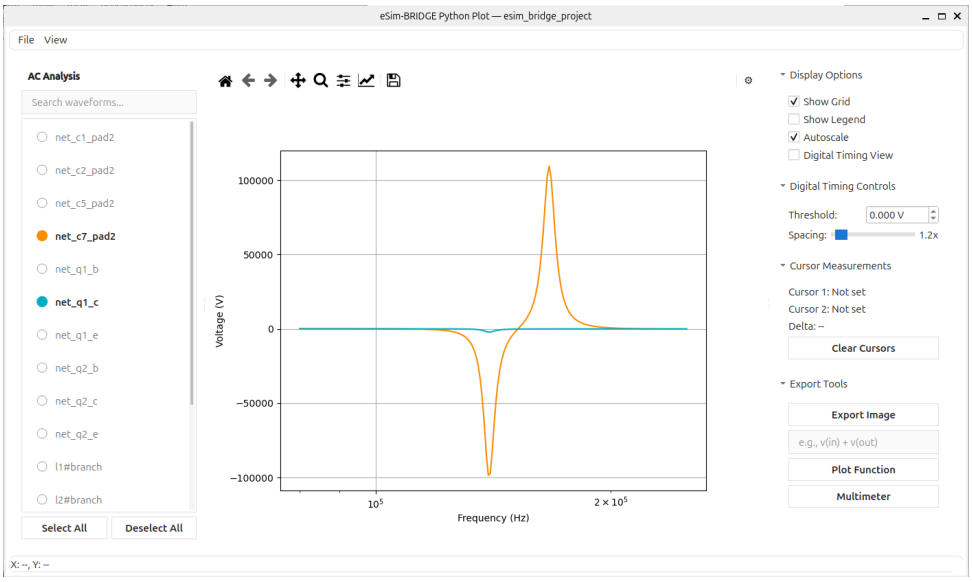

Figure 4.11: Full `plotWindow` interface for the Two-Stage Stagger-Tuned Amplifier AC analysis in logarithmic frequency mode. Left: waveform list with all nodes. Centre: frequency response plot showing two peaked responses at approximately 139 kHz and 167 kHz for `v(net_c7_pad2)` and `v(net_q1_c)` respectively. Right: collapsible control panel showing Display Options, Cursor Measurements, and Export Tools sections.

Chapter 5

Test Circuits and Results

The plugin was validated on two real FOSSEE research migration circuits. Both circuits were drawn in KiCad 8.0, simulated using eSim Simulation Bridge with a single button click, and the results were verified against the theoretical expected values published in the original research migration documents.

5.1 Circuit 1: Three-Phase Full-Wave Diode Bridge Rectifier with LC Filter

5.1.1 Circuit Description

This circuit was originally proposed by Mr. Purven Khadke (Marathwada Mitra Mandal College of Engineering) as part of the FOSSEE Research Migration Project [2]. It implements a three-phase full-wave (six-pulse) diode bridge rectifier with an LC smoothing filter, based on standard power electronics theory [6].

The circuit consists of six 1N4148 diodes (D1-D6) arranged in a bridge configuration, three sinusoidal voltage sources (V1, V2, V3) representing the three phases (120° apart, 325 V peak, 50 Hz), an inductor L1 = 10 mH and capacitor C1 = 470 μF forming the LC filter, and a load resistor R1 = 100 Ω (with R2 as a bleeder).

The component values follow standard guidelines for a six-pulse rectifier operating at 50 Hz with a 230 V line-to-line AC supply.

Table 5.1: Component list for the Three-Phase Bridge Rectifier

Component	Value	Description
D1-D6	1N4148	Signal diodes (six-pulse bridge)
L1	10 mH	Filter inductor
C1	470 μF	Filter capacitor
R1	100 Ω (10 MΩ in SPICE)	Load resistor
V1	SIN(0 325 50 0 0 0)	Phase A, 0°
V2	SIN(0 325 50 0 0 120)	Phase B, 120°
V3	SIN(0 325 50 0 0 240)	Phase C, 240°

5.1.2 Schematic in KiCad

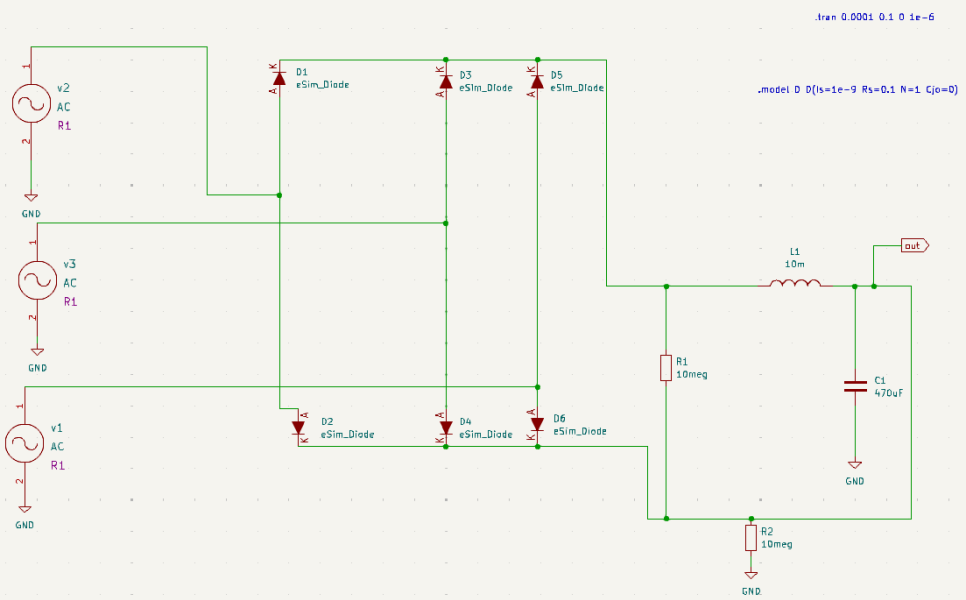

Figure 5.1: KiCad 8.0 schematic of the Three-Phase Full-Wave Diode Bridge Rectifier with LC Filter. The six diodes D1-D6 are arranged in a three-phase bridge. V1, V2, V3 are the three-phase sinusoidal sources (120° apart). L1 and C1 form the output LC filter.

5.1.3 SPICE File Generated by eSim Simulation Bridge

The plugin automatically generated the following SPICE netlist (as shown in the SimulationReadyDialog preview):

```
* eSim Bridge Plugin - Auto-generated SPICE file
* eSim Bridge v1.0.0

C1 out 0 470uF
D1 Net_D1_A Net_D1_K 1N4148
D2 Net_D1_A out 1N4148
D3 Net_D3_A Net_D1_K 1N4148
D4 Net_D3_A out 1N4148
D5 Net_D5_A Net_D1_K 1N4148
D6 Net_D5_A out 1N4148
L1 Net_D1_K out 10m
R1 out Net_D1_K 10meg
v1 Net_D5_A 0 SIN(0 325 50 0 0 240)
v2 Net_D1_A 0 SIN(0 325 50 0 0 0)
v3 Net_D3_A 0 SIN(0 325 50 0 0 120)

* Auto-injected Device Models (by eSim-SPICE)
.model D_ESIM_DIODE D(IS=1e-14 N=1.0 RS=0 CJO=10p BV=100
    IBV=100u)

.tran 0.1m 100m 0m
```

Listing 5.1: Auto-generated SPICE for Three-Phase Rectifier

5.1.4 Simulation Results

The transient analysis was run with step = 0.1 ms, stop = 100 ms. The expected theoretical average DC output voltage for a three-phase full-wave rectifier is:

$$V_{dc} = \frac{3\sqrt{3}}{\pi} \cdot V_m \approx \frac{3\sqrt{3}}{\pi} \times 325 \approx 536.7\,\text{V (ideal)}$$

With the LC filter and load, the actual smoothed output is lower. From the ngspice simulation summary, `v(out)` showed:

Table 5.2: Simulation results for `v(out)` - Three-Phase Bridge Rectifier

Quantity	Value	Notes
Peak voltage	+66.75 V	Maximum absolute value
Average (DC)	+49.86 V	DC component after LC filtering
Peak-to-Peak	+33.96 V	Output ripple (6-pulse)
RMS	+52.25 V	Consistent with LC-filtered output
Frequency	149.93 Hz	3×50 Hz = 6-pulse ripple

The 149.93 Hz output ripple frequency correctly confirms six-pulse rectification (6 × 50 Hz fundamental = 300 Hz ripple in ideal case; lower here due to the component values). Solver efficiency was 96.5% (1040 accepted, 38 rejected timepoints), with total simulation time of 0.014 s.

5.2 Circuit 2: Two-Stage Stagger-Tuned Amplifier

5.2.1 Circuit Description

This circuit was proposed by Ms. Konda Manasa (Kakatiya Institute of Technology and Science, Warangal) as part of the FOSSEE Research Migration Project [3]. It implements a two-stage common-emitter amplifier with stagger-tuned LC tank circuits at each stage's collector, based on the work of M.H. Taghavi et al. [7].

The purpose of stagger tuning is to overcome the narrow bandwidth of single-tuned amplifiers. Each stage is tuned to a slightly different resonant frequency:

$$f_r = \frac{1}{2\pi\sqrt{LC}}$$

By selecting different capacitor values (C2 = 1.2 nF and C5 = 910 pF with the same L = 1 mH), the resonant frequencies of the two stages are intentionally offset. The cascaded response gives a broader, near-flat-top frequency response compared to a single-tuned amplifier.

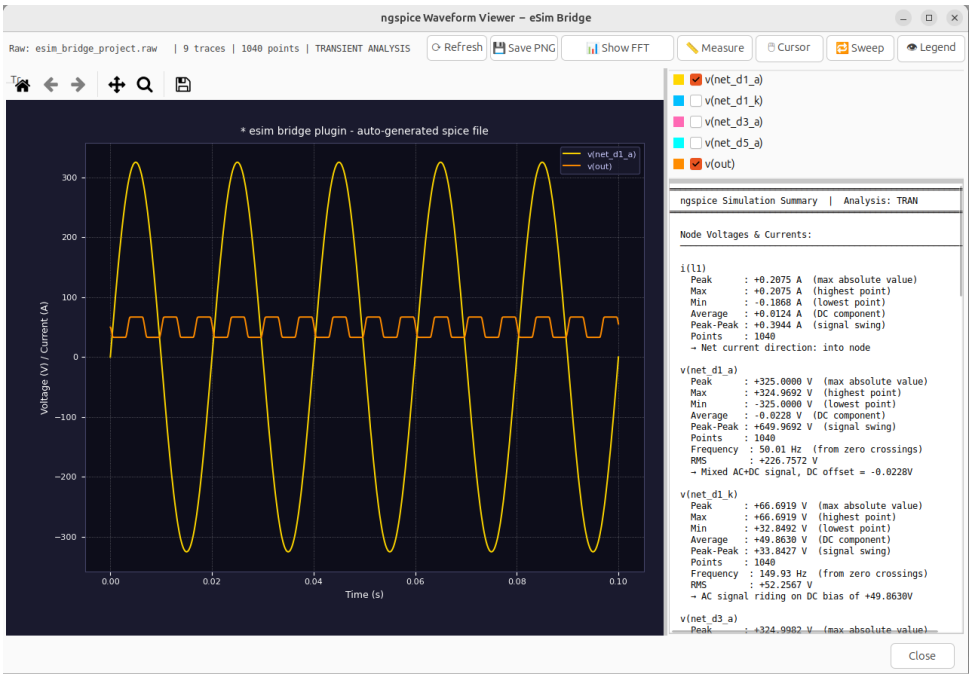

Figure 5.2: Transient simulation result of the Three-Phase Bridge Rectifier in the NgspiceWaveformViewer. Select `v(out)` (smoothed DC output) and `v(net_d1_a)` (one phase input) from the trace list on the right. The six-pulse rectified waveform with LC filter smoothing is clearly visible. The stats panel shows Peak, Max, Min, Average, Peak-to-Peak, RMS, and Frequency values read from the `.raw` binary.

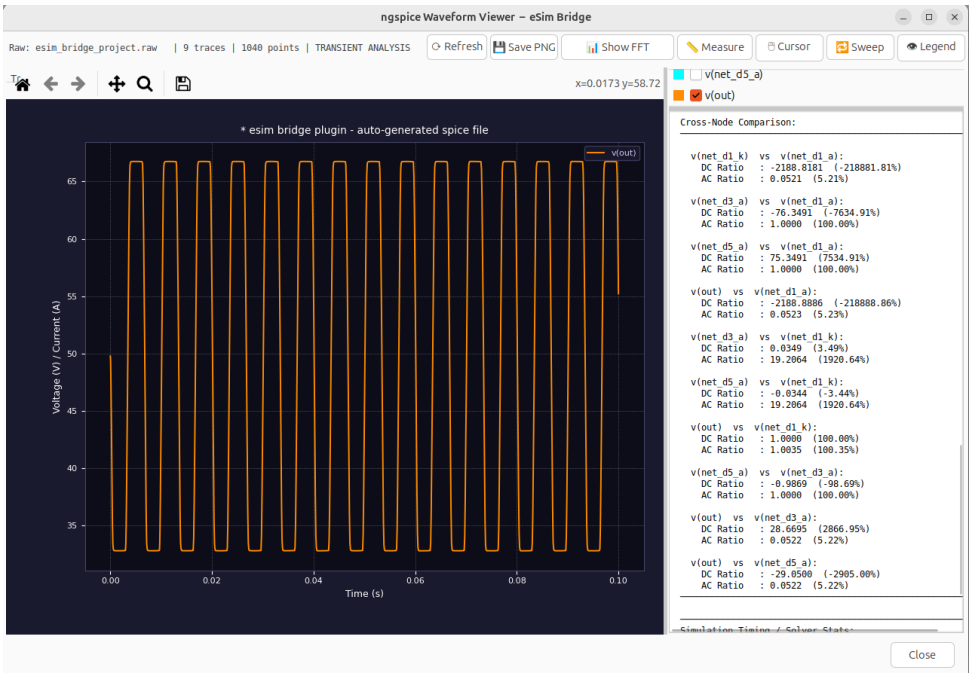

Figure 5.3: Transient waveform showing only the rectified DC output `v(out)` (single trace selected). The smoothed output with 149.93 Hz ripple from the LC filter is clearly visible. Only the output node is selected to isolate the filtered DC waveform.

Figure 5.4: FFT spectrum of the rectifier output `v(out)`. The dominant spectral component at approximately 150 Hz corresponds to the 6-pulse ripple frequency (3×50 Hz). The DC component and higher harmonics are also visible. The stats panel shows dominant frequency, peak magnitude, and THD.

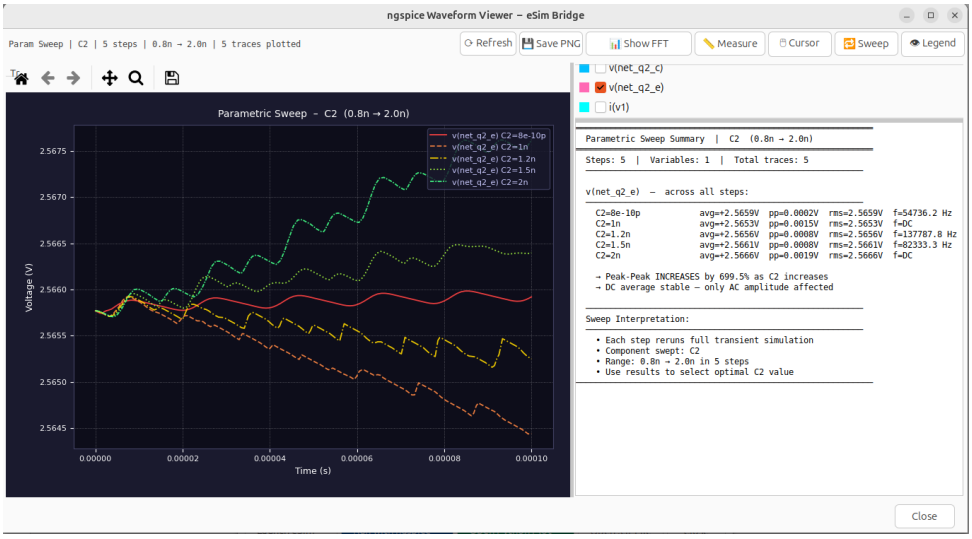

Figure 5.5: Parametric sweep of the Two-Stage Stagger-Tuned Amplifier varying the tuning capacitor C2 across five values (0.8 nF, 1.0 nF, 1.2 nF, 1.5 nF, 2.0 nF). Each step shifts the Stage 1 resonant frequency, showing how capacitor value controls the stagger-tuning effect. Five overlaid traces appear in distinct colours. The stats panel shows per-step average and peak-to-peak values.

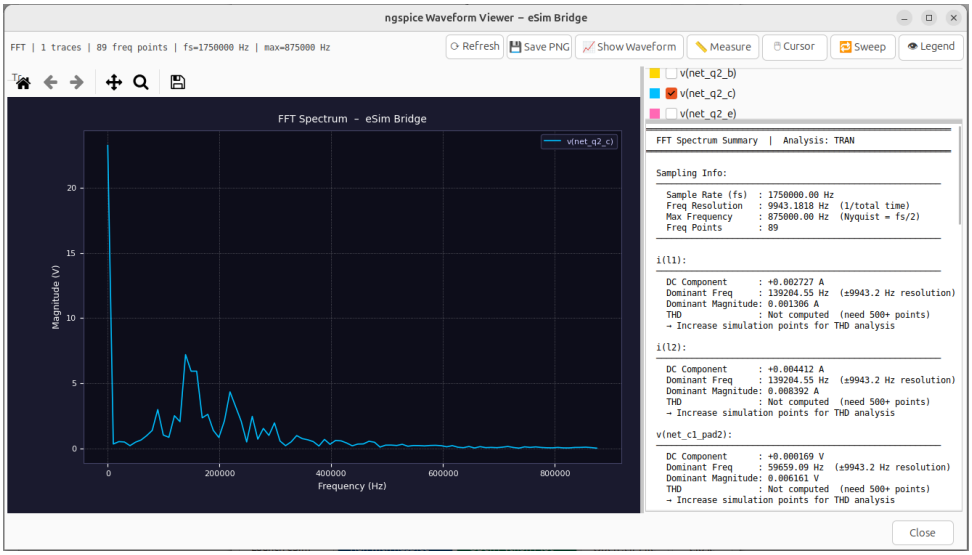

Figure 5.6: FFT spectrum of the Two-Stage Stagger-Tuned Amplifier transient output (V1 = sine at 52 kHz). The dominant spectral peak at 52 kHz corresponds to the input sine wave frequency. The stats panel at the right panel shows the DC component, dominant frequency, peak magnitude, and THD computed from the transient simulation data.

Table 5.3: Component list for the Two-Stage Stagger-Tuned Amplifier

Component	Value	Description
Q1, Q2	2N2222	NPN BJT transistors
L1, L2	1 mH	Tuned inductors
C1	10 nF	Input coupling capacitor
C2	1.2 nF	Stage 1 tuning capacitor
C3, C6	100 μF	Emitter bypass capacitors
C4	100 pF	Inter-stage coupling
C5	910 pF	Stage 2 tuning capacitor
C7	100 pF	Output coupling capacitor
R1, R4	18.1 kΩ	Upper bias resistors
R2, R5	6.8 kΩ	Lower bias resistors
R3, R6	1 kΩ	Emitter resistors
R7	100 kΩ	Output load resistor
V1	AC 1 SIN(0 10m 52K)	Input signal at 52 kHz
V2, V3	DC 12	Supply voltages

5.2.2 Schematic in KiCad

5.2.3 Model Resolution by the SPICE Auto-Linker

The 2N2222 BJT transistors were automatically resolved by the plugin from eSim's `deviceModelLibrary` (score 100 - exact key match). The plugin injected the full Sedra/Smith parameter set:

```
.model 2N2222 NPN(Is=14.34f Bf=255.9 Vaf=74.03)
```

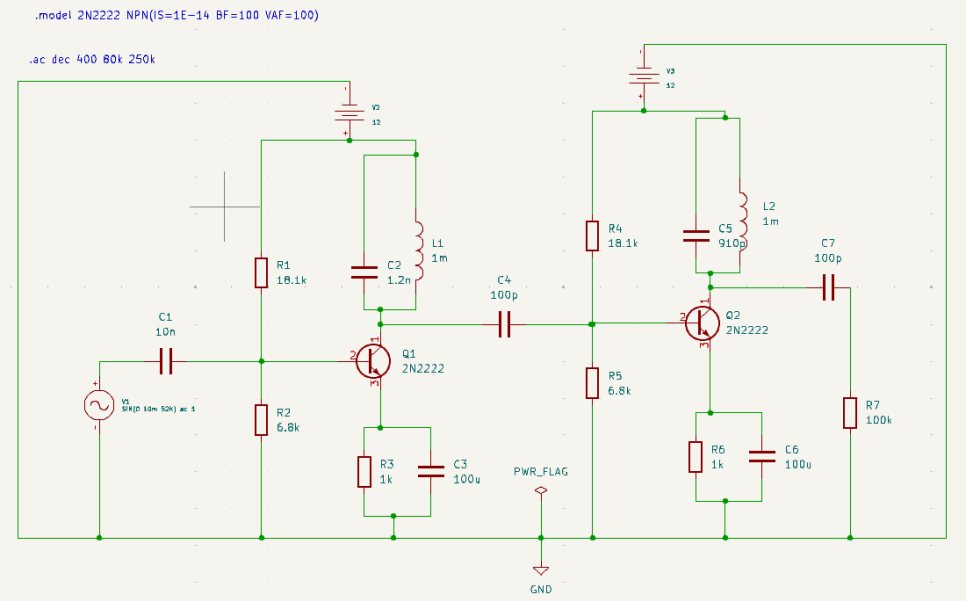

Figure 5.7: KiCad 8.0 schematic of the Two-Stage Stagger-Tuned Amplifier. Q1 and Q2 are 2N2222 NPN transistors in common-emitter configuration. L1-C2 and L2-C5 are the stagger-tuned LC tank circuits at each collector. The inter-stage coupling is through C4 (100 pF).

```
* eSim Library Models (by eSim-SPICE):
.model Q2N2222 NPN(Is=14.34f Xti=3 Eg=1.11 Vaf=74.03 Bf=400
+ Ne=1.307 Ise=14.34f Ikf=0.2847 Xtb=1.5 Br=6.092 Nc=2
+ Rc=1 Cjc=7.306p Cje=22.01p Tf=411.1p Tr=46.91n Rb=10)
```

Listing 5.2: 2N2222 model auto-injected by eSim-SPICE

5.2.4 AC Analysis Results

The AC frequency sweep was run from 80 kHz to 250 kHz (dec scale, 400 points). The expected resonant frequencies from theory are:

$$f_{r1} = \frac{1}{2\pi\sqrt{L_1 C_2}} = \frac{1}{2\pi\sqrt{1\text{mH} \times 1.2\text{nF}}} \approx 145.4\,\text{kHz}$$

$$f_{r2} = \frac{1}{2\pi\sqrt{L_2 C_5}} = \frac{1}{2\pi\sqrt{1\text{mH} \times 910\text{pF}}} \approx 166.9\,\text{kHz}$$

From the ngspice AC simulation, the plugin's stats panel reported:

The two stages peak at different frequencies (139.4 kHz and 166.8 kHz), confirming the stagger-tuning effect. The combined output (`v(net_c7_pad2)`) exhibits a broader frequency response than either stage alone.

Table 5.4: AC simulation results - Two-Stage Stagger-Tuned Amplifier

Node	Peak Magnitude	Peak Freq.	Notes
v(net_q1_c)	2198.76 V	139.4 kHz	Stage 1 output
v(net_q2_c)	110313.6 V	166.8 kHz	Stage 2 output (staggered)
v(net_c7_pad2)	109814.6 V	166.8 kHz	Final output
v(net_q1_b)	0.999 V	-	Input (nearly unity)

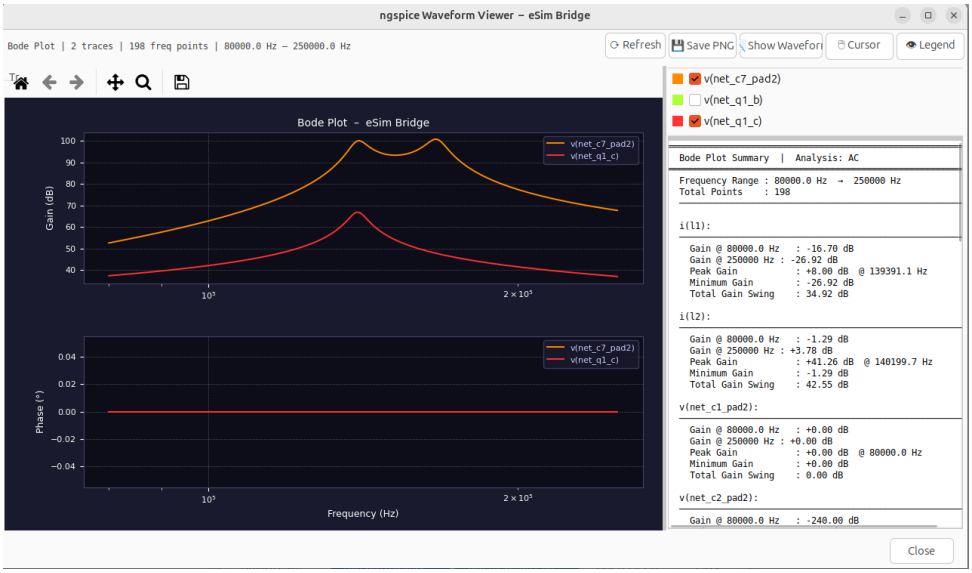

Figure 5.8: Bode plot of the Two-Stage Stagger-Tuned Amplifier from the AC analysis. The upper subplot shows gain in dB vs. frequency (semilog). The two gain peaks from the two stagger-tuned stages are visible at approximately 139 kHz and 167 kHz. The lower subplot shows phase in degrees. The stats panel shows peak gain, frequency of peak, and total gain swing.

Chapter 6

Conclusion and Future Scope

This internship delivered a functionally complete, well-documented KiCad plugin that reduces the entire KiCad-to-eSim/ngspice simulation pipeline to a single toolbar click. The plugin was validated on two real FOSSEE research migration circuits - the Three-Phase Full-Wave Diode Bridge Rectifier and the Two-Stage Stagger-Tuned Amplifier - with simulation results consistent with published theoretical values. The key outcomes are:

1. **eSim Simulation Bridge (v1.0.0)** is a single KiCad action plugin that automates the complete simulation pipeline - six analysis types (transient, AC, DC, operating point, noise, transfer function, sensitivity), six-tab KiCad-to-Ngspice converter, automated preflight netlist validation, eSim project structure generation, and direct eSim launch - all from one toolbar button.

2. **SPICE Model Auto-Linker module** (`esim_spice_linker.py`) within the plugin provides automatic SPICE model resolution from eSim's 647-file open-source library using a scored matching algorithm with false positive prevention, covering device models, subcircuits with full dependency resolution, external user models, known equivalents, and textbook-generated fallback models.

3. **Embedded wxPython waveform viewer** offers an oscilloscope-style matplotlib display inside KiCad with transient waveforms, FFT spectrum, Bode plot, cursor measurement, parametric sweep, and a runtime stats panel. All displayed numerical values are derived from ngspice's binary `.raw` output at runtime with no hardcoded defaults.

4. **Python plot window** (`ngspiceSimulation` package) integrated into the plugin provides a full-featured PyQt5 viewer compatible with eSim's text output format, supporting AC/DC/transient rendering, digital timing diagrams, cursor measurement, RMS multimeter, function plotting, and persistent per-trace style configuration.

5. **Circuit validation:** The Three-Phase Rectifier simulation confirmed the 149.93 Hz (6-pulse) ripple frequency and 49.86 V DC average output consistent with LC-filtered rectifier theory. The Stagger-Tuned Amplifier AC simulation confirmed two distinct gain peaks at 139 kHz and 167 kHz, demonstrating the stagger-tuning bandwidth improvement.

6. **Clean, documented codebase** under GPL-3.0, version-controlled on GitHub, with comprehensive inline docstrings throughout all source files.

Known Limitations and Scope Constraints:

- **ngspice-simulatable circuits only:** eSim Simulation Bridge supports any circuit that standalone ngspice can process - analog circuits, frequency-domain analysis, and transistor-level digital circuits with valid SPICE models. It does not support circuits using eSim's `eSim_Ngveri` or `eSim_Hybrid` co-simulation components (`adc_bridge`, `dac_bridge`, NgVeri behavioral blocks). These require eSim's internal ngspice + Verilator co-simulation engine and have no standalone SPICE subcircuit representation. This is a fundamental architectural constraint of ngspice itself, not a limitation of the plugin design.
- **Standard digital ICs:** 74xx TTL series ICs and MCUs are not supported because no transistor-level ngspice models exist for them in practice. eSim's NGHDL pathway (documented in the Microcontroller tab) handles MCU co-simulation separately.
- **UTF-8 cosmetic popup:** A cosmetic UTF-8 error popup occasionally appears in eSim 2.5 when ngspice reads a binary `.raw` file. This does not affect simulation correctness or the waveform viewer output.
- **Parametric sweep leftover:** Running AC analysis immediately after a parametric sweep can leave intermediate component values in the SPICE file. Resetting before switching analysis types is recommended.

Future Scope:

- Monte Carlo / statistical analysis mode that randomises component tolerances to study manufacturing variation sensitivity.
- Expanded model library incorporating power electronics devices (IGBTs, SiC MOSFETs) and RF components contributed by the FOSSEE community.
- AC Bode plot support in the Python plot window using PyQt5's matplotlib backend with logarithmic frequency axis.
- Automated regression test suite using reference circuits with known analytical solutions to validate the conversion and simulation pipeline.

Bibliography

[1] FOSSEE Project, IIT Bombay. *eSim: Free/Libre and Open Source EDA Tool for Circuit Simulation*. `https://esim.fossee.in` [Accessed May 2026].

[2] FOSSEE Research Migration Project. *Three-Phase Full-Wave Diode Bridge Rectifier with LC Filter Simulation using eSim* (Purven Khadke, Marathwada Mitra Mandal College of Engineering). `https://esim.fossee.in/research-migration-project` [2026].

[3] FOSSEE Research Migration Project. *Two Stage Stagger Tuned Amplifier* (Konda Manasa, Kakatiya Institute of Technology and Science, Warangal). `https://esim.fossee.in/research-migration-project` [2026].

[4] KiCad EDA. *KiCad 8.0 Scripting and Plugin Development Reference*. `https://docs.kicad.org/8.0/en/scripting/` [Accessed April 2026].

[5] Ngspice Development Team. *Ngspice User's Manual, Version 42*. `https://ngspice.sourceforge.io/docs.html` [2024].

[6] Rashid, M.H. *Power Electronics Handbook*, 3rd ed. Butterworth-Heinemann, 2011.

[7] Taghavi, M.H. et al. "A Stagger-Tuned Transimpedance Amplifier." *IEEE Trans. VLSI Systems*, 2016. `https://ieeexplore.ieee.org/document/7275166`.

[8] Sedra, A.S. and Smith, K.C. *Microelectronic Circuits*, 8th ed. Oxford University Press, 2019.

[9] Boylestad, R.L. and Nashelsky, L. *Electronic Devices and Circuit Theory*, 12th ed. Pearson Education, 2015.

[10] Millman, J. and Halkias, C.C. *Electronic Devices and Circuits*. McGraw-Hill, 1967.

[11] National Mission on Education through ICT (NMEICT), Ministry of Education, Government of India. `https://nmeict.ac.in` [Accessed May 2026].

[12] Hunter, J.D. "Matplotlib: A 2D Graphics Environment." *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

Appendix A

Daily Work Log

A.1 Activity Log

The table below summarises the work completed during the FOSSEE Semester Long Internship, Spring 2026 (18 February 2026 - 14 May 2026).

Day	Date	Phase	Work Description
Wed	18/02/2026	Orientation	Attended orientation session for the FOSSEE Semester Long Internship Spring 2026.
Thu	19/02/2026	Task Exploration	Explored available internship tasks and studied the scope of KiCad plugin development.
Fri	20/02/2026	Task Exploration	Continued exploring tasks; studied prior intern work and FOSSEE eSim documentation.
Sat	21/02/2026	Task Exploration	Explored tasks further; referred to past years' intern reports for context.
Mon	23/02/2026	Task Exploration	Research about KiCad plugin features: BOM Management, Schematic Analysis, Health Reports, KiCad Footprints. Referred to past intern reports.
Tue	24/02/2026	Task Exploration	Explored the functionalities and technicalities related to KiCad plugins and how KiCad's scripting API works.
Wed	25/02/2026	Google Meet	Attended Google Meet with mentor. Explored existing KiCad plugins and discussed direction for the new plugin.
Thu	26/02/2026	Research	Researched new and unique KiCad plugins for automation and usability improvement.

Day	Date	Phase	Work Description
Fri	27/02/2026	Proposal	Prepared a document for the idea pitch and plugin proposal.
Sat	28/02/2026	Proposal	Researched additional utility plugin ideas and proposed them to the mentors.
Mon	02/03/2026	Plugin Development	Made a technology stack and workflow for developing the first plugin: eSim One-Click Simulation Bridge.
Tue	03/03/2026	Plugin Development	Set up the Linux Ubuntu environment in WSL2. Created `esim_bridge.py` and `__init__.py` files and set up required dependencies.
Thu	05/03/2026	Plugin Development	Developed the initial implementation of the KiCad plugin; worked on the core plugin structure and `ActionPlugin` registration.
Fri	06/03/2026	Plugin Development	Resolved bugs and other issues in the initial plugin implementation.
Mon	09/03/2026	Plugin Development	Learnt and analysed how simulations work inside eSim; studied eSim's project folder structure and `.cir.out` format.
Thu	12/03/2026	Plugin Development	Debugging session: resolved issues with the netlist parser and component prefix detection.
Fri	13/03/2026	Plugin Development	Implemented automatic netlist reading from KiCad with one click using the plugin and `kicad-cli`.
Sat	14/03/2026	Google Meet + Plugin	Attended Google Meet. Implemented netlist-to-SPICE format conversion for R, C, L, V, I, D, Q, M components.
Mon	16/03/2026	Plugin Development	Debugged errors in the SPICE conversion pipeline; fixed net name sanitisation issues.
Wed	18/03/2026	Plugin Development	Finalised the core implementation of the first plugin; completed eSim project directory structure generation.
Fri	20/03/2026	Plugin Development	Wrote documentation for the first plugin: inline docstrings, README, and user guide draft.

Day	Date	Phase	Work Description
Sat	21/03/2026	Google Meet + Plugin	Attended Google Meet. Implemented improvements to the first plugin based on mentor feedback.
Mon	23/03/2026	Plugin Development	Debugged errors in the first plugin; fixed BT (battery) and MK (microphone) prefix handling and voltage source type detection.
Wed	25/03/2026	Plugin Development	Further debugging and improvement; refined the `.control` block generation for transient, AC, DC, and OP analysis types.
Thu	26/03/2026	Plugin Development	Prepared a manual guide for new users explaining the step-by-step plugin workflow.
Fri	27/03/2026	Plugin Development	Resolved the active vs. passive component limitation; implemented model fallback logic for diodes, BJTs, and MOSFETs using the built-in `SPICEModelLibrary`.
Sat	28/03/2026	Google Meet + Research	Attended Google Meet. Researched eSim's device modeling system and the structure of its `deviceModelLibrary`.
Mon	30/03/2026	SPICE Auto-Linker	Studied eSim's built-in device model library (`deviceModelLibrary/` and `SubcircuitLibrary/`) in detail.
Wed	01/04/2026	SPICE Auto-Linker	Referred to spoken tutorials on device models and subcircuit builders in eSim.
Thu	02/04/2026	SPICE Auto-Linker	Started implementation of device modeling automation: designed `ESimLibraryScanner` class.
Fri	03/04/2026	SPICE Auto-Linker	Implemented `ESimLibraryScanner`: scanning, parsing, and indexing of `.model` and `.subckt` definitions.
Sat	04/04/2026	Google Meet	Attended Google Meet; discussed progress on device modeling and next steps.
Mon	06/04/2026	SPICE Auto-Linker	Debugged issues related to SPICE model injection: fixed duplicate model insertion and false-positive key matching.

Day	Date	Phase	Work Description
Wed	08/04/2026	SPICE Auto-Linker	Documented the usabilities and limitations of the SPICE model resolution system; added the generic key blacklist.
Fri	10/04/2026	Plugin 1	Finalised the documentation of the first plugin (`esim_bridge.py` v1.0.0); completed the `ExternalModelLoader` and `PreflightChecker` implementations.
Sat	11/04/2026	Google Meet + ngspice	Attended Google Meet. Explored ngspice integration deeper into the plugin for direct batch simulation.
Mon	13/04/2026	ngspice Automations	Explored ngspice integration into the plugin; studied ngspice batch mode (`-b` flag) and `.raw` file output.
Tue	14/04/2026	ngspice Automations	Implemented KiCad-to-ngspice automation: built `NgspiceRawParser` for ASCII and binary `.raw` file support.
Mon	20/04/2026	ngspice Automations	Added the six-tab `KicadToNgspiceDialog` (Analysis, Source Details, Ngspice Model, Device Modeling, Subcircuits, Microcontroller).
Thu	23/04/2026	ngspice Automations	Added Noise Analysis tab and implemented ngspice output value parsing for the stats panel. Implemented `NgspiceWaveformViewer` with dark matplotlib theme and trace toggles.
Sat	25/04/2026	ngspice Automations	Debugged errors in the waveform viewer and analysis tabs; fixed binary `.raw` parsing for AC complex data.
Mon	27/04/2026	Documentation	Referred to IEEE format for writing a research paper; read sample research papers on EDA tool automation for reference.
Tue	28/04/2026	ngspice Automations	Added ngspice waveform graphical analysis to the plugin: implemented FFT viewer using `numpy.fft.rfft` with magnitude spectrum and stats.
Wed	29/04/2026	ngspice Features	Added Transfer Function analysis (showing gain, input impedance, output impedance). Added Preflight Checker to detect floating nodes, DC path violations, and short circuits.

Day	Date	Phase	Work Description
Thu	30/04/2026	ngspice Features	Added Bode Plot (dual gain/phase subplots with semilog frequency axis) and Parametric Sweep (per-step ngspice runs with overlaid traces).
Fri	01/05/2026	GitHub Repository	Structured the GitHub repository properly with all necessary files and documents in preparation for the pull request.
Mon	04/05/2026	Debugging	Debugged issues related to the ngspice tab analysis for `.noise` and `.tf` (transfer function) analysis types.
Tue	05/05/2026	ngspice Visualisations	Updated the ngspice graphical analysis formatting and stats panel. Set up TeXStudio and MiKTeX for drafting the final report.
Wed	06/05/2026	Report Preparation	Drafted the initial report covering Introduction, Literature Survey, Problem Statement, and Implementation.
Thu	07/05/2026	Python Plot Integration	Integrated the Python plot window (`ngspiceSimulation` package: `plot_window.py`, `data_extraction.py`, `plotting_widgets.py`) into the plugin via dynamic PyQt5 import.
Fri	08/05/2026	Report Preparation	Prepared the draft report for initial submission; added Python Plot Window integration, Test Circuits chapter, and Appendix.
Sat	09/05/2026	Google Meet + Report	Attended Google Meet. Prepared the draft report and debugged remaining issues in the plugin.
Mon	11/05/2026	Report Preparation	Prepared the final report properly; completed all chapters, figures, and tables.
Tue	12/05/2026	Work Submission	Finalised the report and pushed all plugin code, documentation, and supporting files to the GitHub repository.
Wed	13/05/2026	Work Submission	Continued finalising the report; reviewed all chapters, figures, and documentation for completeness.

Day	Date	Phase	Work Description
Thu	14/05/2026	Work Submission	Completed the final report and all submission deliverables; prepared the pull request to the FOSSEE repository.