Summer Fellowship Report

On

Enhancing eSim's GUI and Functionality

Submitted by

Arpita Kukreja
B.Tech in CSE(AIML), DCE

Under the guidance of

Sumanto Kar
IIT Bombay

July 16, 2025

Acknowledgment

I would like to express my heartfelt gratitude to FOSSEE, IIT Bombay, for giving me the opportunity to contribute to the development of the open-source project eSim. My deepest thanks to my mentor, Mr. Sumanto Kar, whose continuous guidance, motivation, and support have helped me grow throughout this fellowship.

This fellowship has been a remarkable learning experience, allowing me to gain valuable insights into open-source development, GUI design, and bug fixing. It has truly been an enriching and rewarding journey.

Contents

1 Introduction **4**
- 1.1 About FOSSEE . . . 4
- 1.2 About eSim . . . 4
- 1.3 Tools and Technologies Used . . . 4
- 1.4 Components Integrated with eSim . . . 5

2 Problem Statement **6**
- 2.1 Problem Definition . . . 6
- 2.2 Objectives . . . 6
- 2.3 Approach . . . 6
- 2.4 Impact . . . 7
- 2.5 Skills Gained . . . 7

3 Resolution of Console Area Scrollbar Issue(Issue #247) **8**
- 3.1 Overview . . . 8
- 3.2 Bug Description . . . 8
- 3.3 Expected Behavior . . . 8
- 3.4 Approach and Solution . . . 9
- 3.5 Before and After Comparison . . . 11
- 3.6 Outcome . . . 11

4 GUI Enhancements in eSim **12**
- 4.1 Theoretical Background . . . 12
  - 4.1.1 Human-Computer Interaction and Visual Ergonomics . . . 12
  - 4.1.2 Design Systems and Consistency . . . 13
- 4.2 System Architecture . . . 13
  - 4.2.1 Theme State Management . . . 13
  - 4.2.2 Implementation Components . . . 13
  - 4.2.3 Previous eSim Interface Limitations . . . 13
- 4.3 Implementation Details . . . 14
  - 4.3.1 Dark Theme Implementation . . . 15
  - 4.3.2 Light Theme Implementation . . . 16
  - 4.3.3 Dynamic Text Editor Styling . . . 18
  - 4.3.4 Color Theory and Palette Selection . . . 19
- 4.4 Dynamic Theme Implementation for eSim Components . . . 19
  - 4.4.1 KiCad to NgSpice Converter . . . 19
  - 4.4.2 Model Editor Component . . . 21
  - 4.4.3 Schematic Converter . . . 24
  - 4.4.4 Simulation . . . 26
  - 4.4.5 Subcircuit Component . . . 27
  - 4.4.6 Modelica Converter . . . 29
- 4.5 User Experience Considerations . . . 31
  - 4.5.1 Accessibility . . . 31
  - 4.5.2 Performance Optimization . . . 32
  - 4.5.3 Advanced Features . . . 32

4.6 Keyboard Shortcuts Implementation in eSim . . . 32
4.6.1 Font Size Adjustment System . . . 33
4.6.2 Functional Shortcuts Implementation . . . 33
4.6.3 Keyboard Shortcuts Reference . . . 35
4.7 Plot GUI Enhancements . . . 35
4.7.1 Class Structure . . . 36
4.7.2 Theme Implementation . . . 37
4.7.3 User Interface Components . . . 37
4.7.4 Multimeter Integration . . . 39

5 Project Section Redesign 41
5.1 Introduction . . . 41
5.2 Previous File Structure and Challenges . . . 41
5.2.1 Navigation Complexity and User Experience Issues . . . 41
5.2.2 Scalability and Performance Limitations . . . 41
5.2.3 Sequential Access Limitations . . . 41
5.3 System Architecture and Design . . . 42
5.3.1 Design Goals . . . 42
5.3.2 Component Architecture . . . 42
5.4 Implementation of Tab-Based File Handling . . . 43
5.4.1 Core Architecture and Design Philosophy . . . 43
5.4.2 Spatial Efficiency and Screen Real Estate Optimization . . . 43
5.5 File Management Features . . . 43
5.5.1 Advanced Tab Closure Mechanisms . . . 43
5.5.2 Integrated Save Functionality and State Management . . . 44
5.6 Implementation Details . . . 44
5.6.1 ProjectExplorer Modifications . . . 44
5.6.2 Event Handling Implementation . . . 45
5.6.3 Original openProject Method Enhancement . . . 45
5.6.4 MainView Integration . . . 47
5.6.5 Tabbed Interface Implementation . . . 47
5.7 User Interface Enhancements . . . 49
5.7.1 Tab Management Features . . . 49
5.7.2 Visual Styling . . . 49

6 Conclusion 50
6.0.1 Personal Growth and Development . . . 50
6.0.2 Contribution to Open Source Community . . . 50
6.0.3 Future Aspirations . . . 51
6.1 Future Scope for GUI Enhancement . . . 51
6.1.1 Advanced Theme Customization . . . 51
6.1.2 Enhanced Collaboration Features . . . 51
6.1.3 Advanced Simulation and Analysis . . . 51

7 Bibliography 52

Chapter 1

Introduction

1.1 About FOSSEE

FOSSEE (Free and Open Source Software for Education) is an initiative by IIT Bombay under the National Mission on Education through ICT, funded by the Ministry of Education, Government of India. Its primary goal is to reduce dependency on proprietary software in academic and research institutions by promoting the adoption of free and open-source alternatives. FOSSEE supports various domains including electronics, simulation, system modeling, mathematics, and engineering design through tools such as eSim, Scilab, and others. These tools provide cost-effective, high-quality alternatives that are accessible to students, educators, and professionals.

To fulfill its mission, FOSSEE provides comprehensive materials such as tutorials, documentation, workshops, hands-on training, and teaching aids to accomplish its objectives. It works together with individuals and academic institutions to enhance the open source culture in education. The projects allow users not only to access and use these tools, but also motivate users to build and enhance the tools which are available to them. Through its inclusive and collaborative approach, FOSSEE plays a vital role in democratizing technology, enhancing innovation, and building a self-reliant academic ecosystem in India and beyond.

1.2 About eSim

eSim is a powerful, open-source Electronic Design Automation (EDA) tool developed by the FOSSEE, IIT Bombay. It integrates several Free/Libre and Open Source Software (FLOSS) tools including NgSpice for circuit simulation, KiCad for schematic capture and PCB layout, GHDL and Verilator for VHDL/Verilog simulation, and Scilab for numerical computation. It also supports mixed-signal simulation and system-level modeling through integration with OpenModelica and NGHDL. These integrations make eSim a versatile and comprehensive tool for electronics education and research.

Some of its key features include schematic design, simulation of analog and digital circuits, PCB layout generation, microcontroller support, and the ability to add custom device models. The Subcircuit feature enables hierarchical circuit design, allowing users to manage complex designs by integrating smaller functional blocks. eSim is cross-platform, supporting both Linux (Ubuntu) and Windows, and is released under the GPL license, ensuring full accessibility and modifiability. As part of the FOSSEE initiative, eSim not only empowers students, educators, and researchers but also promotes collaborative development and the long-term sustainability of open-source EDA tools in academic and professional environments.

1.3 Tools and Technologies Used

- Ubuntu OS

- Python
- PyQt5
- Git GitHub
- NgSpice
- KiCad

1.4 Components Integrated with eSim

eSim brings together multiple open-source tools and custom-built modules to provide a unified platform for circuit design, simulation, and PCB layout. These components work in coordination to offer a complete EDA solution, particularly suitable for students, educators, and researchers in electronics.

Major Components of eSim:

- **KiCad:** An open-source tool used for schematic capture and PCB layout. It provides essential features such as component libraries, auto-routing, and Gerber file generation for PCB fabrication.
- **NgSpice:** A general-purpose circuit simulator based on the SPICE engine. It supports DC, AC, and transient analysis for analog and mixed-signal circuits.
- **GHDL:** A VHDL simulator integrated into eSim to verify digital circuit designs. It enables event-driven digital simulations and supports mixed-mode operation with analog tools.
- **Verilator:** A fast and widely used Verilog simulator, allowing high-performance simulation and debugging of digital circuits.
- **NGHDL Module:** Enables mixed-signal circuit simulation by combining NgSpice for analog and GHDL for digital simulations, providing support for co-simulation.
- **KiCad-to-NgSpice Converter:** Converts schematic netlists from KiCad into a format compatible with NgSpice, allowing users to simulate directly without rewriting designs.
- **Model Builder:** Allows users to create or edit models of semiconductor devices like diodes, BJTs, MOSFETs, JFETs, and IGBTs. Enhances flexibility in component-level simulation.
- **Subcircuit Builder:** Supports creation and management of modular subcircuits such as Op-Amps, IC-555 timers, and UJTs, enabling hierarchical and reusable circuit designs.
- **Makerchip Integration:** Interfaces eSim with the Makerchip platform for browser-based Verilog/SystemVerilog simulation and debugging through a user-friendly environment.

Chapter 2

Problem Statement

2.1 Problem Definition

As part of the internship under the eSim project at FOSSEE, IIT Bombay, the primary focus was on enhancing the graphical user interface (GUI) to provide a more intuitive, visually consistent, and user-friendly experience for circuit design and simulation. The work involved designing and implementing new features that improved usability to ensure a seamless and intuitive interface for users, identifying existing limitations in the GUI and systematically adding greater visual customization for users.

2.2 Objectives

The main objective of this internship was to improve the user interface and usability of eSim through the following key goals:

- Introduce a toggle feature for switching between dark and light themes.
- Redesign the project section layout using a tab-based interface.
- Enhance the plotting and console sections for improved clarity.
- Improve navigation through keyboard shortcuts and interface responsiveness.
- Add save buttons within the file editor for seamless editing.
- Integrate font-size control and improve file name visibility.
- Align all UI components with a unified theme palette.

2.3 Approach

The development process began with analyzing the existing user interactions across the platform and identifying opportunities for refinement. The first step involved improving the visual consistency of the interface by adding a light/dark theme toggle. The project section was redesigned with a tab-based system that allowed users to open and manage multiple files more efficiently.

To further streamline the workflow, save buttons were embedded into editable file tabs, while shortcut keys were introduced for frequently used actions like theme switching, opening tools on left toolbar and text zooming. Visual enhancements were also made to the plotting interface,welcome section, model editor window, KiCad-to-NgSpice converion window and console layout to improve clarity and responsiveness across the GUI.

2.4 Impact

These enhancements significantly improved the overall usability and user experience of eSim. The redesigned tab-based file handling system enabled more efficient navigation, while the new theme options and interface adjustments contributed to a modern, accessible, and visually appealing environment. Collectively, these contributions helped advance eSim as a more developer-friendly and educationally impactful tool for the open-source electronics community.

2.5 Skills Gained

During the internship, I gained valuable experience in user interface design, debugging, and open-source development. Working with PyQt helped me strengthen my skills in GUI development, especially in implementing features like tab navigation, theme toggles, and shortcut keys.

It also strengthened my understanding of software architecture, and collaborating in a structured development environment. This experience also enhanced my problem-solving abilities and gave me a deeper understanding of how user experience plays a key role in software design.

Contributing to a live project like eSim not only improved my technical skills but also taught me the importance of clean design, clear documentation, and teamwork in open-source contributions.

Chapter 3

Resolution of Console Area Scrollbar Issue(Issue #247)

3.1 Overview

Issue #247 was one of the key issues addressed, a graphical bug in the console area—specifically, the visibility and alignment of the scrollbar. This bug, present as Issue #247 on GitHub, impacted user experience by making the scroll arrows partially hidden and the scrollbar rod overlap its boundary.

3.2 Bug Description

The scrollbar in the console output section the vertical scrollbar was not properly styled and lacked clarity in both appearance and interaction.

- The scrollbar arrows (up/down) were barely visible and difficult to distinguish due to poor contrast and small size.
- The scroll rod (handle) appeared squished and misaligned within the scroll track, leading to confusion when scrolling large output.
- The edges of the scrollbar overlapped with the console container boundary, resulting in a cluttered and unpolished appearance.

3.3 Expected Behavior

The expected behavior of the console scrollbar in eSim is described below:

- **Visible Scroll Arrows:**
  The up and down arrows of the scrollbar should be fully visible and easily accessible to the user, without being clipped or hidden by the container edges.
- **Proper Scroll Rod Alignment:**
  The scroll rod (handle) must remain entirely within the scroll track and should not overlap or extend beyond the boundaries of the scrollbar container.
- **No Overlapping Elements:**
  All scrollbar components should stay within their assigned layout areas and should not overlap with the console's text area or other UI elements.
- **Smooth Scrolling:**
  Users should be able to scroll through console output seamlessly, with no lag or abrupt jumps, enabling easy review of simulation logs.

3.4 Approach and Solution

To resolve the issue, I made use of Python's PyQt framework to apply a custom scrollbar style using the `setStyleSheet()` method. This lead to proper appearance and behavior of scrollbar components.

Steps Taken:

1. **Set Explicit Scrollbar:**
   The vertical and horizontal scrollbars for the console's QTextEdit widget were defined.This ensured that scrollbars appear only when necessary, optimizing space and usability.

2. **Applied Custom Scrollbar Styling with `setStyleSheet()`:**
   Applied enhanced custom CSS styling for scrollbar using setStyleSheet() in Python (Qt).Fixed the layout issue where scrollbar components (especially the rod/handle) were overlapping the container boundaries by setting appropriate margins and min-height/min-width value.

   - Styled both vertical and horizontal scrollbars with consistent themes.
   - Enhanced visibility of scroll arrows and scroll rod.
   - Added proper border-radius, padding, hover effects and background-color for scrollbar components for better integration with the overall UI.
   - Used `border-radius` and spacing to create smooth corners and avoid overlap.

3. **Improved Layout and Alignment:**
   `margin`, `padding`, `min-height`, and `min-width` values were carefully adjusted to ensure:

   - Scrollbar elements fit properly within the container.
   - The scroll rod no longer overlapped the console area.
   - A clean, uncluttered user interface was maintained.

4. **Pull Request Submitted and Merged:**
   The fix was submitted as a pull request on GitHub referencing Issue #247, and after review, it was successfully merged into the main codebase.

```
# Set explicit scrollbar policy
self.noteArea.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAsNeeded)
self.noteArea.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAsNeeded)

 # Enhanced CSS with proper scrollbar styling
        self.noteArea.setStyleSheet("""
        QTextEdit {
            border-radius: 15px;
            border: 1px solid gray;
            padding: 5px;
            background-color: white;
        }

        QScrollBar:vertical {
            border: 1px solid #999999;
            background: #f0f0f0;
            width: 16px;
            margin: 16px 0 16px 0;
            border-radius: 3px;
        }

        QScrollBar::handle:vertical {
            background: #606060;
            min-height: 20px;
            border-radius: 3px;
```

```
            margin: 1px;
        }

        QScrollBar::handle:vertical:hover {
            background: #505050;
        }

        QScrollBar::add-line:vertical {
            border: 1px solid #999999;
            background: #d0d0d0;
            height: 15px;
            width: 16px;
            subcontrol-position: bottom;
            subcontrol-origin: margin;
            border-radius: 2px;
        }

        QScrollBar::sub-line:vertical {
            border: 1px solid #999999;
            background: #d0d0d0;
            height: 15px;
            width: 16px;
            subcontrol-position: top;
            subcontrol-origin: margin;
            border-radius: 2px;
        }

        QScrollBar::add-line:vertical:hover,
        QScrollBar::sub-line:vertical:hover {
            background: #c0c0c0;
        }

        QScrollBar::add-page:vertical,
        QScrollBar::sub-page:vertical {
            background: none;
        }

        QScrollBar::up-arrow:vertical {
            width: 8px;
            height: 8px;
            background-color: #606060;
        }

        QScrollBar::down-arrow:vertical {
            width: 8px;
            height: 8px;
            background-color: #606060;
        }

        QScrollBar:horizontal {
            border: 1px solid #999999;
            background: #f0f0f0;
            height: 16px;
            margin: 0 16px 0 16px;
            border-radius: 3px;
        }

        QScrollBar::handle:horizontal {
            background: #606060;
            min-width: 20px;
            border-radius: 3px;
```

```
            margin: 1px;
        }

        QScrollBar::handle:horizontal:hover {
            background: #505050;
        }

        QScrollBar::add-line:horizontal,
        QScrollBar::sub-line:horizontal {
            border: 1px solid #999999;
            background: #d0d0d0;
            width: 15px;
            height: 16px;
                border-radius: 2px;
        }

        QScrollBar::add-line:horizontal:hover,
            QScrollBar::sub-line:horizontal:hover {
                background: #c0c0c0;
            }
        """)
```

Listing 3.1: Custom Scrollbar Styling in PyQt

3.5 Before and After Comparison

Figure 3.1: Before Fix – Overlapping scroll rod and hidden arrows

Figure 3.2: After Fix – Proper scrollbar with visible arrows and aligned layout

3.6 Outcome

After implementing the above fixes, the scrollbar now appears as expected in the console area. The bug fix was tested successfully and merged into the main codebase, resolving GitHub Issue #247.

Chapter 4

GUI Enhancements in eSim

Introduction

This chapter presents a comprehensive overview of the graphical user interface (GUI) enhancements implemented in eSim, an open-source Electronic Design Automation (EDA) tool. The enhancements focus on improving user experience through modern design patterns, accessibility features, and streamlined workflows. Key improvements include the implementation of a light/dark mode toggle, improved plotting interfaces, and comprehensive keyboard shortcut integration.The GUI enhancement focused on modernizing the visual design, streamlining workflows, and increasing accessibility of eSim.

4.1 Theoretical Background

4.1.1 Human-Computer Interaction and Visual Ergonomics

The concept of adaptive user interfaces stems from fundamental principles of human-computer interaction (HCI) and visual ergonomics. Research in visual perception has demonstrated that different lighting conditions require different interface designs to maintain optimal readability and reduce eye strain.

Light Theme Characteristics

Light themes, characterized by light backgrounds and dark text, are traditionally preferred in well-lit environments. The high contrast between dark text and light backgrounds maximizes readability under bright ambient lighting conditions. Light themes typically feature:

- High luminance backgrounds
- Low luminance text
- Subtle color accents for interactive elements
- Minimal use of shadows and depth cues

Dark Theme Characteristics

Dark themes, conversely, utilize dark backgrounds with light text and are particularly beneficial in low-light environments. The reduced overall screen brightness helps prevent eye strain and can contribute to better sleep patterns by minimizing blue light exposure. Dark themes typically feature:

- Low luminance backgrounds
- High luminance text

- Vibrant color accents for emphasis
- Enhanced use of gradients and depth effects

4.1.2 Design Systems and Consistency

A robust theme system requires careful consideration of design consistency across all interface elements. The implementation follows modern design system principles, ensuring that theme changes affect all components uniformly while maintaining visual hierarchy and user experience quality.

4.2 System Architecture

4.2.1 Theme State Management

The theme management system is built around a centralized state management approach, where a single boolean flag (`is_dark_theme`) controls the entire application's appearance. This design pattern ensures consistency and simplifies maintenance.

4.2.2 Implementation Components

The system comprises several key components:

1. **Theme State Manager**: Maintains the current theme state
2. **Style Sheet Generator**: Creates appropriate CSS/QSS styles
3. **Component Updater**: Applies styles to UI components
4. **Toggle Interface**: Provides user control over theme switching

4.2.3 Previous eSim Interface Limitations

Prior main window interface delivered essential circuit design capabilities through a functional but dated layout and lack of theme support eSim.

Key Characteristics:

- **Classic Layout Structure**: Left-aligned project explorer and vertical toolbars, central welcome tab.
- **Toolbar Simplicity**: Minimal icons for project and help actions, lacking visual grouping or styling enhancements.
- **Basic Docking**: Rigid layout with minimal customization or dynamic docking behavior.
- **Text Presentation**: Default system fonts and simple hyperlink presentation for documentation.
- **Limited Modern UI Features**: Absence of flat design, hover effects, or dark/light theme switching.

While fully functional, the legacy version lacked modern design aesthetics and accessibility enhancement.

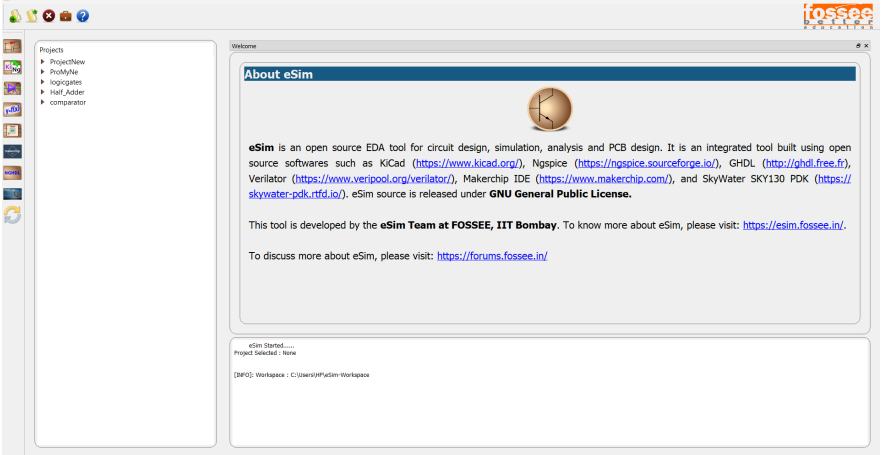

Figure 4.1: Previous eSim GUI

4.3 Implementation Details

Theme Toggle Mechanism

The core theme switching functionality is implemented through a centralized toggle system:

```
def toggle_theme(self):
    """Toggle between light and dark themes."""
    self.is_dark_theme = not self.is_dark_theme
    if self.is_dark_theme:
        self.apply_dark_theme()
        self.theme_toggle.setIcon(QtGui.QIcon(init_path + 'images/sun.png')
            )
        self.theme_toggle.setToolTip('Switch to Light Mode (Ctrl+T)')

        # Update component themes
        self.update_schematic_converter_theme(is_dark=True)
        self.update_model_editor_theme(is_dark=True)
        self.update_makerchip_ngveri_theme(is_dark=True)

        # Update terminal UI theme
        if hasattr(self, 'current_terminalui_widget') and self.
            current_terminalui_widget is not None:
            self.current_terminalui_widget.set_theme(True)
    else:
        self.apply_light_theme()
        self.theme_toggle.setIcon(QtGui.QIcon(init_path + 'images/dark_mode
            .png'))
        self.theme_toggle.setToolTip('Switch to Dark Mode (Ctrl+T)')

        # Update component themes
        self.update_schematic_converter_theme(is_dark=False)
        self.update_model_editor_theme(is_dark=False)
        self.update_makerchip_ngveri_theme(is_dark=False)

        # Update terminal UI theme
        if hasattr(self, 'current_terminalui_widget') and self.
            current_terminalui_widget is not None:
            self.current_terminalui_widget.set_theme(False)
```

Listing 4.1: Theme Toggle Implementation

4.3.1 Dark Theme Implementation

The dark theme implementation focuses on creating a premium, modern appearance with carefully selected color palettes and gradient effects. The system uses Qt Style Sheets (QSS) to define the visual appearance of all interface elements.

```
def apply_dark_theme(self):
    """Applies a premium, modern dark theme everywhere using only supported QSS properties."""
    premium_dark_stylesheet = """
    QMainWindow, QDialog, QMessageBox {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
            stop:0 #0a0e1a, stop:0.3 #1a1d29, stop:0.7 #1e2124, stop:1 #0f1419);
        color: #e8eaed;
        font-family: 'Inter', 'Segoe UI', 'Roboto', 'Arial', sans-serif;
        font-size: 12px;
        font-weight: 500;
    }
    QWidget {
        background: transparent;
        color: #e8eaed;
    }
    QWidget#centralWidget {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
            stop:0 #0a0e1a, stop:0.3 #1a1d29, stop:0.7 #1e2124, stop:1 #0f1419);
    }
    QDockWidget {
        background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
            stop:0 #23273a, stop:1 #181b24);
        color: #e8eaed;
        border: 1px solid #23273a;
        border-radius: 14px;
    }
    QDockWidget::title {
        text-align: left;
        background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
            stop:0 #23273a, stop:1 #181b24);
        color: #40c4ff;
        border-radius: 14px 14px 0 0;
        padding: 8px 12px;
        font-weight: 700;
        font-size: 13px;
    }
    QToolButton {
        background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
            stop:0 #23273a, stop:1 #181b24);
        border: 1px solid #23273a;
        color: #ffffff;
        padding: 4px;
        margin: 2px;
        border-radius: 6px;
        font-weight: 600;
        min-width: 120px;
        max-width: 160px;
    }
    QToolButton:hover {
        background: #1976d2;
        color: #ffffff;
        border: 1.5px solid #1976d2;
    }
```

```
    QToolButton:pressed, QToolButton:checked {
        background: #1976d2;
        color: #fff;
        border: 1.5px solid #1976d2;
    }
    QTabBar::tab {
        background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
            stop:0 #2d3748, stop:1 #1a202c);
        color: #e2e8f0;
        border: 1px solid #4a5568;
        border-bottom: none;
        border-top-left-radius: 12px;
        border-top-right-radius: 12px;
        padding: 12px 28px;
        font-weight: 600;
        min-width: 180px;
        max-width: 400px;
    }
    QTabBar::tab:selected {
        background: #667eea;
        color: #1a202c;
        border: 1px solid #667eea;
        border-bottom: 3px solid #667eea;
        font-weight: 700;
    }
    """
    self.setStyleSheet(premium_dark_stylesheet)
```

Listing 4.2: Dark Theme Implementation

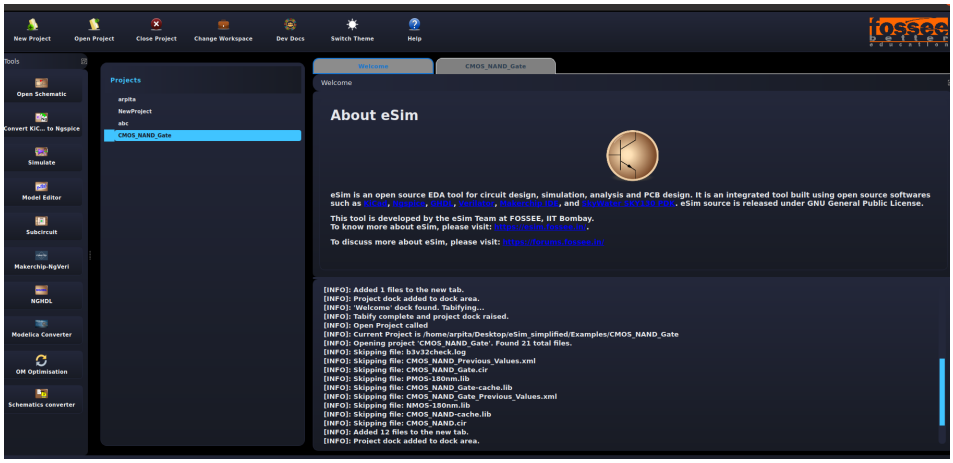

Figure 4.2: Dark Theme of eSim

4.3.2 Light Theme Implementation

The light theme implementation emphasizes clarity and readability with clean, minimalist design elements. The color scheme is carefully balanced to provide sufficient contrast while maintaining visual comfort.

```
def apply_light_theme(self):
    """Apply light theme to the application."""
    premium_light_stylesheet = """
    QMainWindow, QDialog, QMessageBox {
        background: #ffffff;
```

```
        color: #2c3e50;
        font-family: 'Inter', 'Segoe UI', 'Roboto', 'Arial', sans-serif;
        font-size: 12px;
        font-weight: 500;
    }
    QWidget {
        background: transparent;
        color: #2c3e50;
    }
    QWidget#centralWidget {
        background: #ffffff;
    }
    QDockWidget {
        background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
            stop:0 #ffffff, stop:1 #f8f9fa);
        color: #2c3e50;
        border: 1px solid #e1e4e8;
        border-radius: 14px;
    }
    QDockWidget::title {
        text-align: left;
        background: qlineargradient(x1:0, y1:0, x2:1, y2:0,
            stop:0 #ffffff, stop:1 #f8f9fa);
        color: #1976d2;
        border-radius: 14px 14px 0 0;
        padding: 8px 12px;
        font-weight: 700;
        font-size: 13px;
    }
    QToolButton {
        background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
            stop:0 #ffffff, stop:1 #f8f9fa);
        border: 1px solid #e1e4e8;
        color: #2c3e50;
        padding: 4px;
        margin: 2px;
        border-radius: 6px;
        font-weight: 600;
        min-width: 120px;
        max-width: 160px;
    }
    QToolButton:hover {
        background: #1976d2;
        color: #ffffff;
        border: 1.5px solid #1976d2;
    }
    QTabBar::tab {
        background: #ffffff;
        color: #2c3e50;
        border: 1px solid #e1e4e8;
        border-bottom: none;
        border-top-left-radius: 12px;
        border-top-right-radius: 12px;
        padding: 6px 24px;
        font-weight: 600;
        min-width: 180px;
        max-width: 400px;
    }
    QTabBar::tab:selected {
        background: #1976d2;
        color: #ffffff;
```

```
        border: 1px solid #1976d2;
    }
    QTabBar::tab:hover:!selected {
        background: #e8eaed;
        color: #2c3e50;
    }
    """
    self.setStyleSheet(premium_light_stylesheet)
```

Listing 4.3: Light Theme Implementation

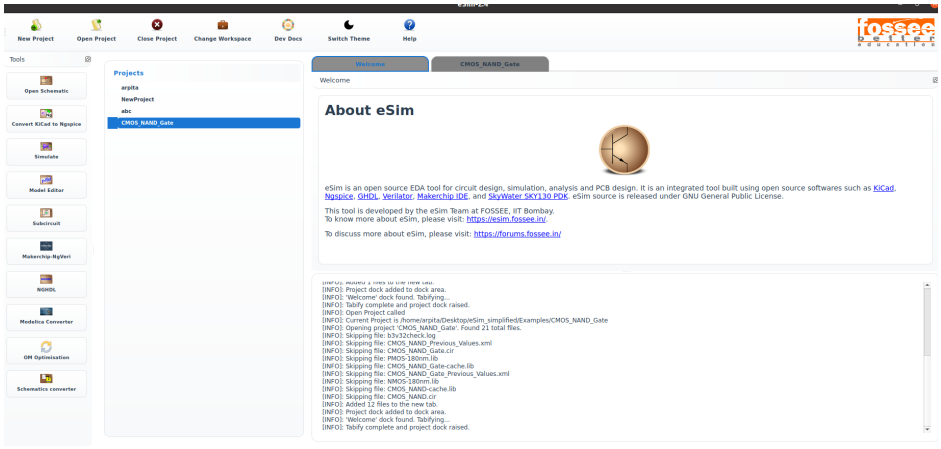

Figure 4.3: Light theme of eSim

4.3.3 Dynamic Text Editor Styling

The text editor component receives special attention to ensure optimal readability in both themes, with appropriate font selection and sizing for different use cases.

```
welcome_font_size = self.text_font_size + 2  # Slightly larger for welcome
    message

if self.is_dark_theme:
    self.obj_Mainview.noteArea.setStyleSheet(f"""
        QTextEdit {{
            font-family: 'Fira Code', 'JetBrains Mono', 'Consolas',
                monospace;
            line-height: 1.4;
            padding: 16px;
            background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
                stop:0 #23273a, stop:1 #181b24);
            color: #e8eaed;
            border: 1px solid #23273a;
            border-radius: 12px;
            font-size: {welcome_font_size}px;
        }}
    """)
else:
    self.obj_Mainview.noteArea.setStyleSheet(f"""
        QTextEdit {{
            font-family: 'Fira Code', 'JetBrains Mono', 'Consolas',
                monospace;
            line-height: 1.4;
            padding: 16px;
```

```
            background: #ffffff;
            color: #2c3e50;
            border: 1px solid #e1e4e8;
            border-radius: 12px;
            font-size: {welcome_font_size}px;
        }}
    """)
```

Listing 4.4: Text Editor Theme Implementation

4.3.4 Color Theory and Palette Selection

Dark Theme Color Palette

The dark theme employs a sophisticated color palette based on blue-gray tones with cyan accents:

Component	Color Code	Usage
Primary Background	#0a0e1a	Main window background
Secondary Background	#23273a	Component backgrounds
Accent Color	#40c4ff	Interactive elements
Text Primary	#e8eaed	Main text content
Text Secondary	#b0b3b8	Secondary text
Border Color	#23273a	Element borders

Table 4.1: Dark Theme Color Palette

Light Theme Color Palette

The light theme uses a clean, professional palette with blue accents:

Component	Color Code	Usage
Primary Background	#ffffff	Main window background
Secondary Background	#f8f9fa	Component backgrounds
Accent Color	#1976d2	Interactive elements
Text Primary	#2c3e50	Main text content
Text Secondary	#6c757d	Secondary text
Border Color	#e1e4e8	Element borders

Table 4.2: Light Theme Color Palette

4.4 Dynamic Theme Implementation for eSim Components

4.4.1 KiCad to NgSpice Converter

Initial Light Mode Implementation

The KiCad to NgSpice converter component initially presented a conventional light-themed interface that prioritized functional clarity and user familiarity. The original implementation featured:

- **Traditional Window Framework**: Standard light gray window backgrounds with conventional border styling
- **Analysis Type Selection**: Clearly defined checkboxes for AC, DC, and TRANSIENT analysis options with traditional black text on white backgrounds
- **Parameter Configuration Panel**: Organized input fields for critical simulation parameters including Start Time, Step Time, and Stop Time

- **Unit Specification System**: Dropdown menus with light backgrounds for time unit selection (seconds, milliseconds)
- **Tabbed Navigation**: Light-themed tabs for different functional areas within the converter interface
- **Convert Button**: Standard gray-toned action button following conventional desktop application design patterns

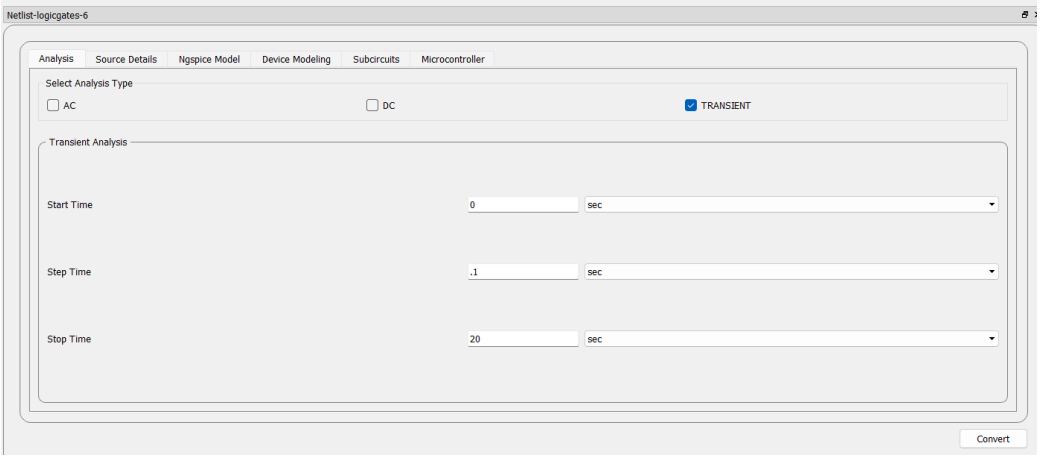

Figure 4.4: Initial KiCad-to-NgSpice GUI

The initial light mode design emphasized clarity and accessibility, providing users with a familiar environment that aligned with traditional CAD software conventions. However, the high contrast white backgrounds could contribute to eye strain during extended circuit analysis sessions.

Enhanced Light Mode Evolution

The intermediate phase of the converter interface introduced several refinements while maintaining the light theme foundation:

- **Improved Layout Organization**: Enhanced spacing and visual hierarchy between different parameter groups
- **Refined Input Field Design**: Better text field styling with clearer boundaries and improved readability
- **Enhanced Tab Structure**: More sophisticated tab design with better visual distinction between active and inactive states
- **Optimized Parameter Display**: Improved alignment and organization of timing parameter controls
- **Professional Aesthetics**: Cleaner overall appearance with subtle design improvements

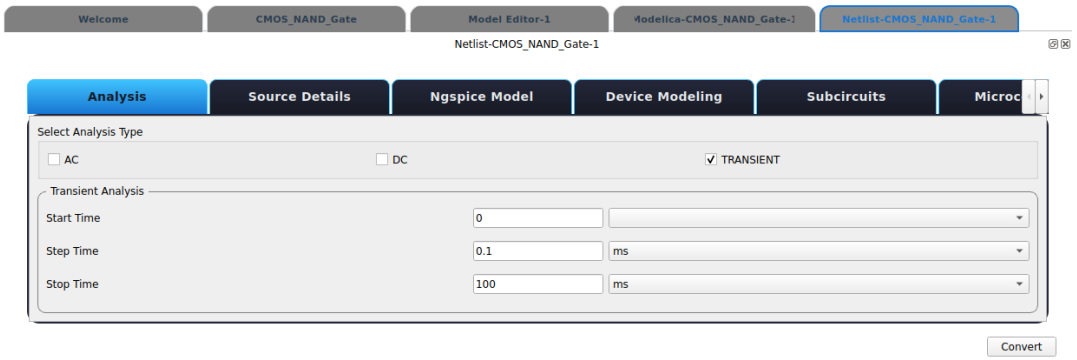

Figure 4.5: KiCad-to-NgSpice:Light theme

This evolutionary phase demonstrated incremental improvements in user interface design while preserving the fundamental light theme approach.

Dark Mode Transformation

The final dark mode implementation represents a comprehensive transformation of the converter interface, introducing significant visual and functional enhancements:

- **Background Transformation**: Complete transition from bright white backgrounds to sophisticated dark blue-gray color schemes
- **Text Optimization**: Conversion of all text elements to high-contrast white and light colors for optimal visibility against dark backgrounds
- **Input Field Redesign**: Dark-themed input fields with light text providing better visual comfort during extended usage
- **Button Enhancement**: Transformation of the Convert button to vibrant blue accent color, creating clear visual hierarchy and improved user interaction feedback

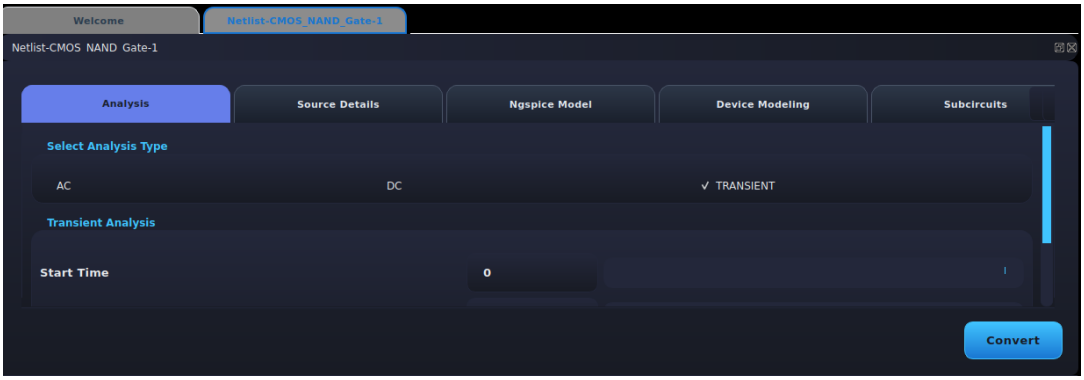

Figure 4.6: KiCad-to-NgSpice:Dark theme

4.4.2 Model Editor Component

Comparative Analysis

The Model Editor component began with a traditional light-themed interface that embodied the conventional design principles of early EDA tools.The Model Editor's evolution from light to dark mode demonstrates significant improvements across multiple dimensions:

Professional Interface Standards

- **Initial State**: Traditional, functional interface with standard desktop application appearance
- **Final State**: Modern, sophisticated interface design reflecting contemporary software development standards

User Interaction Feedback

- **Initial State**: Standard gray buttons with minimal visual hierarchy and limited interaction feedback
- **Final State**: Vibrant blue accent buttons providing clear visual feedback and improved user interaction guidance

Component Selection Experience

- **Initial State**: Basic radio button list with standard light theme styling
- **Final State**: Enhanced component selection interface with improved visibility and selection state indication

Accessibility and Customization

- **Initial State**: Fixed light theme without accommodation for different visual preferences
- **Final State**: Adaptive theming system supporting user preferences and accessibility requirements with real-time switching capability

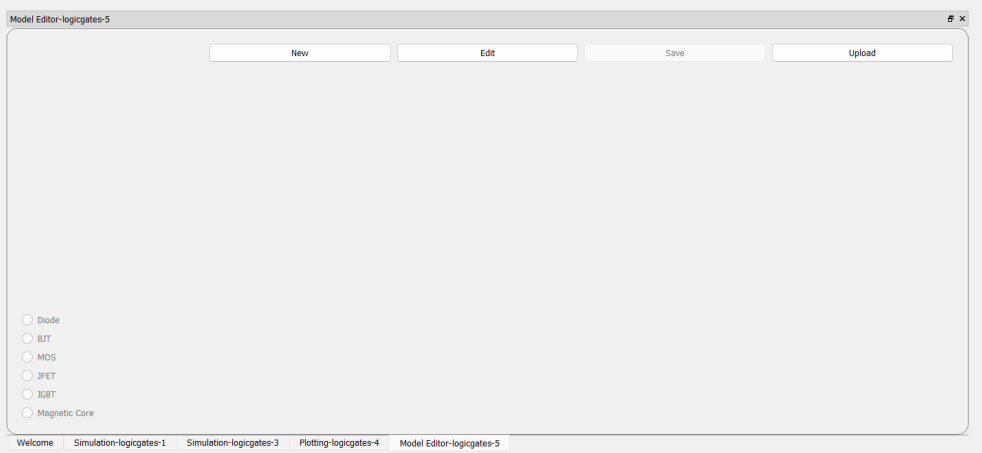

Figure 4.7: Initially Model Editor

Centralized Light and Dark Theme

The implementation utilizes advanced theme control mechanisms:

- **Unified Theme Control**: Central theme management system ensuring consistency across all Model Editor components
- **Efficient Update Processes**: Targeted widget updates minimizing system resource usage during theme transitions
- **Component State Preservation**: Maintenance of all component selections and interface states during theme switching

- **Performance Optimization**: Selective styling updates avoiding unnecessary full interface reloads

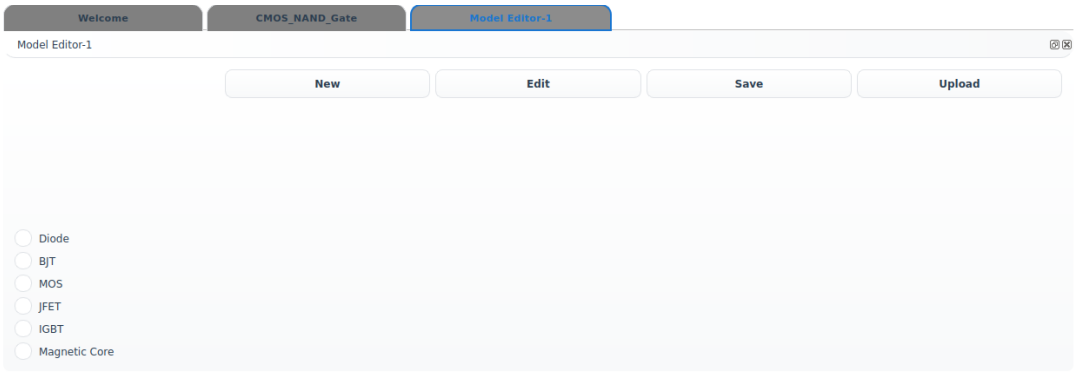

Figure 4.8: Model Editor:Light theme

Dark Theme Implementation

The model editor dark theme emphasizes readability and reduces eye strain:

```
def update_model_editor_theme(self, is_dark):
    """Update model editor theme based on current theme."""
    if is_dark:
        model_editor_style = """
            QGroupBox {
                border: 2px solid #40c4ff;
                border-radius: 14px;
                margin-top: 1em;
                padding: 15px;
                background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
                    stop:0 #23273a, stop:1 #181b24);
                color: #e8eaed;
            }
            QRadioButton {
                color: #e8eaed;
                font-weight: 600;
                font-size: 13px;
            }
            QRadioButton::indicator {
                width: 16px;
                height: 16px;
                border: 2px solid #40c4ff;
                border-radius: 8px;
                background: #23273a;
            }
            QRadioButton::indicator:checked {
                background: #40c4ff;
                border: 2px solid #40c4ff;
            }
            QTableWidget {
                background: #23273a;
                color: #e8eaed;
                border: 1px solid #40c4ff;
                border-radius: 8px;
                gridline-color: #40c4ff;
                font-size: 12px;
            }
            QTableWidget::item {
```

```
                padding: 8px;
                border-bottom: 1px solid #181b24;
            }
            QTableWidget::item:selected {
                background: #40c4ff;
                color: #181b24;
            }
            QHeaderView::section {
                background: #181b24;
                color: #40c4ff;
                border: 1px solid #40c4ff;
                padding: 8px;
                font-weight: 700;
            }
        """
```

Listing 4.5: Model Editor Dark Theme Configuration

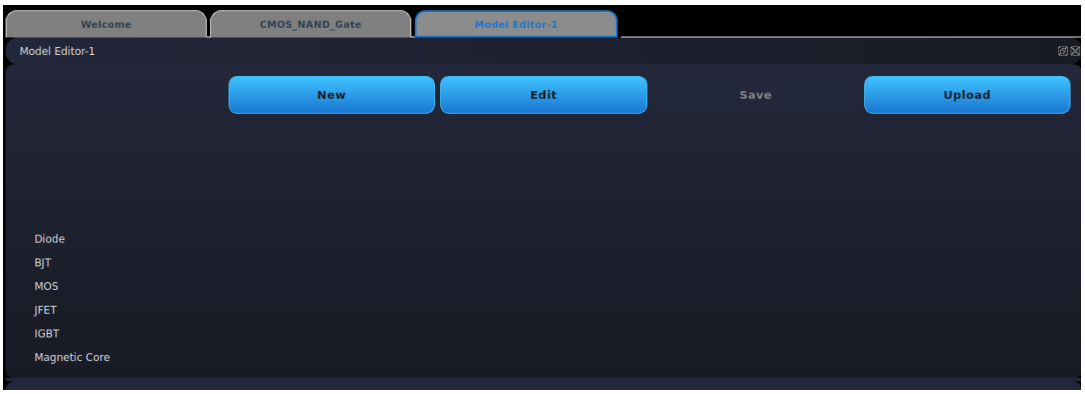

Figure 4.9: Model Editor:Dark theme

4.4.3 Schematic Converter

The schematic converter component utilizes CSS-like styling through Qt's stylesheet system:

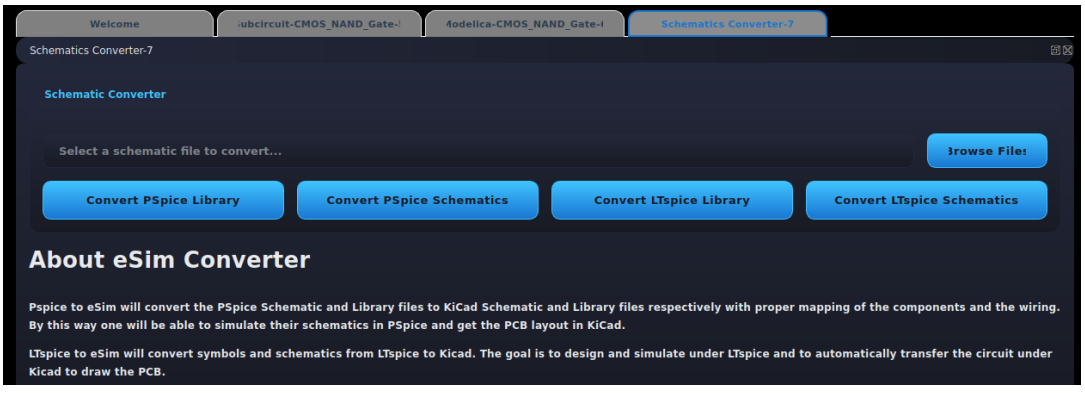

Figure 4.10: Schematic Converter:Dark theme

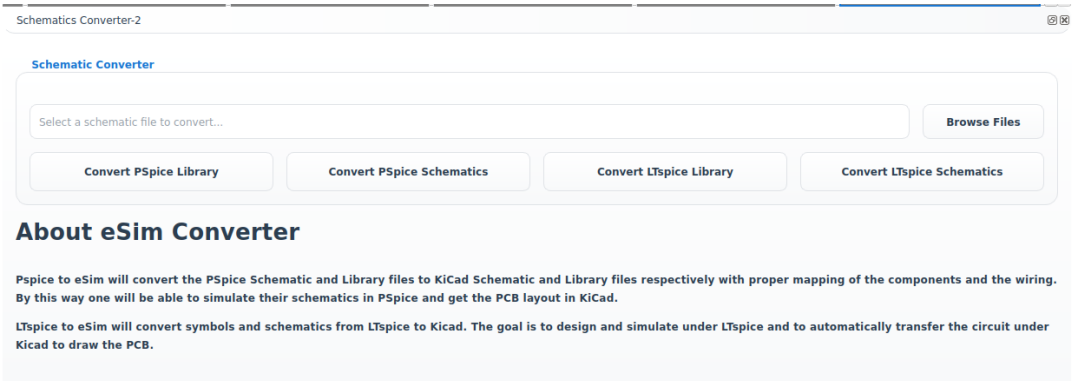

Figure 4.11: Schematic Converter:Light theme

```
def update_schematic_converter_theme(self, is_dark):
    """Update schematic converter theme based on current theme."""
    if is_dark:
        # Dark theme configuration
        converter_style = """
            QGroupBox {
                border: 2px solid #40c4ff;
                border-radius: 14px;
                margin-top: 1em;
                padding: 15px;
                background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
                    stop:0 #23273a, stop:1 #181b24);
                color: #e8eaed;
            }
            QGroupBox::title {
                subcontrol-origin: margin;
                left: 15px;
                padding: 0 5px;
                color: #40c4ff;
                font-weight: bold;
                font-size: 14px;
            }
            QLineEdit {
                background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
                    stop:0 #23273a, stop:1 #181b24);
                color: #e8eaed;
                border: 2px solid #40c4ff;
                border-radius: 10px;
                padding: 8px 15px;
                font-weight: 500;
                font-size: 12px;
            }
            QLineEdit:focus {
                border: 2px solid #1976d2;
                background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
                    stop:0 #2a2f3a, stop:1 #1e2128);
            }
            QPushButton {
                background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
                    stop:0 #40c4ff, stop:1 #1976d2);
                color: #181b24;
                border: 1px solid #40c4ff;
                border-radius: 10px;
                padding: 10px 20px;
                font-weight: 700;
```

```
                font-size: 12px;
            }
            QPushButton:hover {
                background: #1976d2;
                color: #fff;
                border: 1.5px solid #1976d2;
            }
            QPushButton:pressed {
                background: #23273a;
                color: #40c4ff;
                border: 1.5px solid #40c4ff;
            }
        """
```

Listing 4.6: Schematic Converter Dark Theme Styling

4.4.4 Simulation

The Terminal Simulation Console evolution demonstrates exceptional user interface design and technical implementation. The component successfully balances professional functionality with modern aesthetic standards, providing users with an optimal environment for circuit simulation monitoring and analysis.

The sophisticated theming system, comprehensive information display, and advanced technical architecture establish the Terminal Simulation Console as an exemplary component in the `eSim` EDA suite. The seamless integration of dark and light modes, combined with robust functionality and professional appearance, creates an outstanding user experience that supports efficient electronic design workflows.

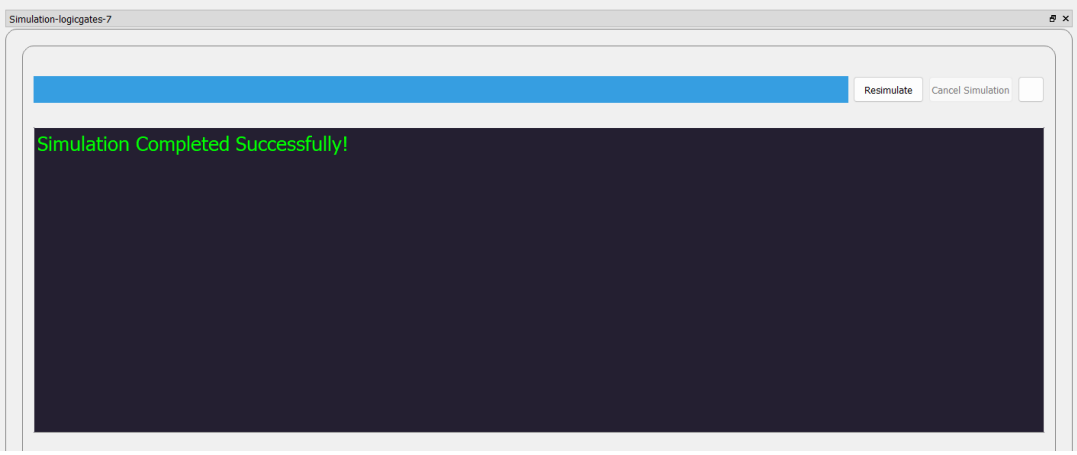

Figure 4.12: Simulate in eSim: Previous GUI

Dynamic Theme Management

- **Real-Time Updates**: Seamless transition between dark and light modes without data loss.
- **Component Consistency**: Uniform theming across all interface elements.
- **Performance Optimization**: Efficient theme switching without simulation interruption.
- **State Preservation**: Complete maintenance of simulation progress and console output during theme changes.

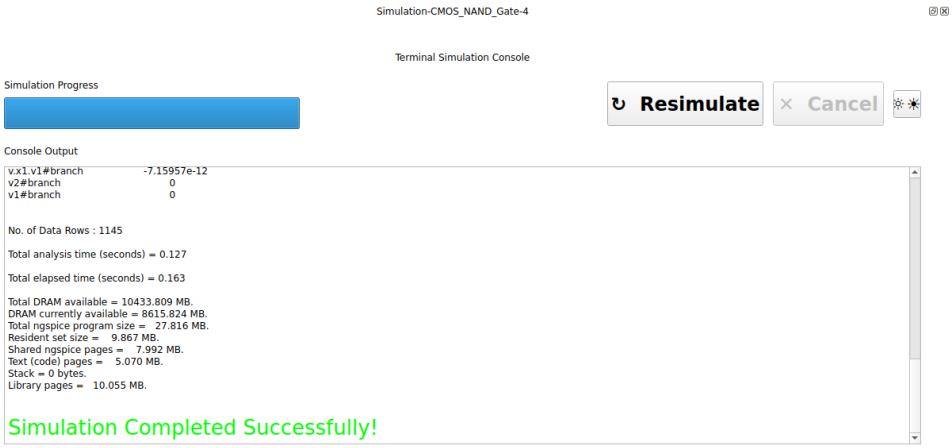

Figure 4.13: Simulate in eSim: Light Mode

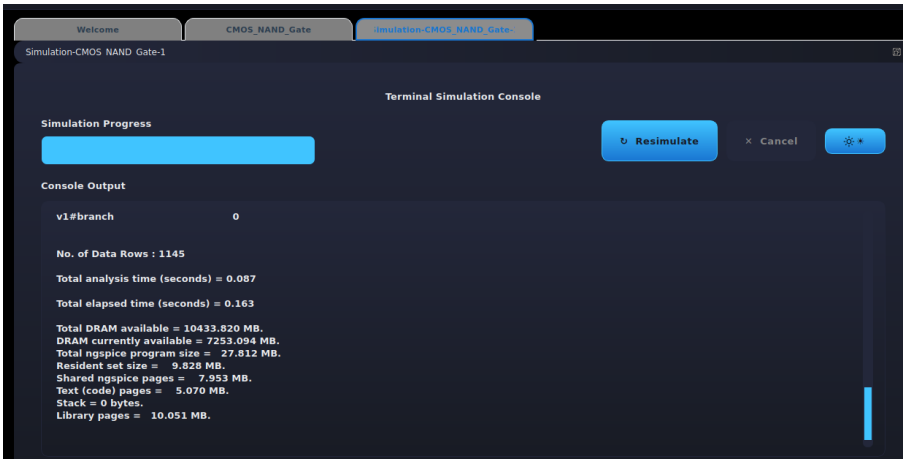

Figure 4.14: Simulate in eSim:Dark Mode

4.4.5 Subcircuit Component

Initial Light Mode Implementation

The Subcircuit Management interface represents a critical component in the eSim EDA environment, serving as the primary control center for hierarchical circuit design and subcircuit integration. The initial light mode design emphasized functional accessibility and user familiarity, providing a clean workspace environment that aligned with conventional CAD software design patterns. However, the extensive white background areas Subcircuit interface featured a minimalist yet comprehensive design structure:

- **Traditional Window Framework**: Standard light gray window chrome with conventional title bar displaying "Subcircuit-logicgates-10"
- **Minimalist Color Scheme**: Predominantly white background with subtle gray button styling and black text elements
- **Standard Desktop Aesthetics**: Traditional flat button design with minimal visual hierarchy and conventional interaction patterns

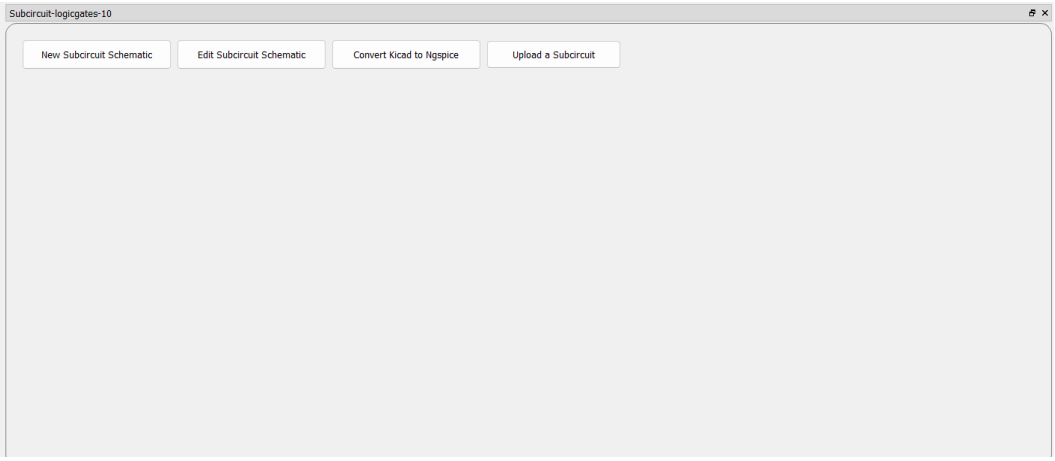

Figure 4.15: Subcircuit in eSim:Initial

Enhanced Light and Dark Mode Evolution

The intermediate evolution phase introduced subtle improvements while maintaining the fundamental light and dark theme approach

The Subcircuit Management interface evolution demonstrates exceptional user interface design and technical implementation. The component successfully balances professional functionality with modern aesthetic standards, providing users with an optimal environment for hierarchical circuit design and subcircuit management.

The sophisticated theming system, comprehensive button functionality, and advanced technical architecture establish the Subcircuit Management interface as an exemplary component in the eSim EDA suite. The seamless integration of dark and light modes, combined with robust functionality and professional appearance, creates an outstanding user experience that supports efficient electronic design workflows.

- **Light Mode Buttons**: Flat, gray-toned buttons with subtle borders and minimal visual impact
- **Dark Mode Buttons**: Vibrant blue buttons with enhanced visual prominence and improved user interaction feedback
- **Consistent Functionality**: Complete preservation of all subcircuit management capabilities across both themes
- **Improved Accessibility**: Better visual distinction and enhanced user interface navigation
- **Background Transformation**: Complete transition from bright white backgrounds to sophisticated dark blue-gray color schemes (approximately #2B3E50)
- **Button Design Excellence**: Transformation of traditional gray buttons to vibrant blue accent buttons (approximately #3498DB) with enhanced visual prominence
- **Text Optimization Strategy**: Conversion of all text elements to high-contrast white and light colors ensuring optimal readability against dark backgrounds
- **Professional Color Palette**: Modern dark theme implementation reflecting contemporary software design standards

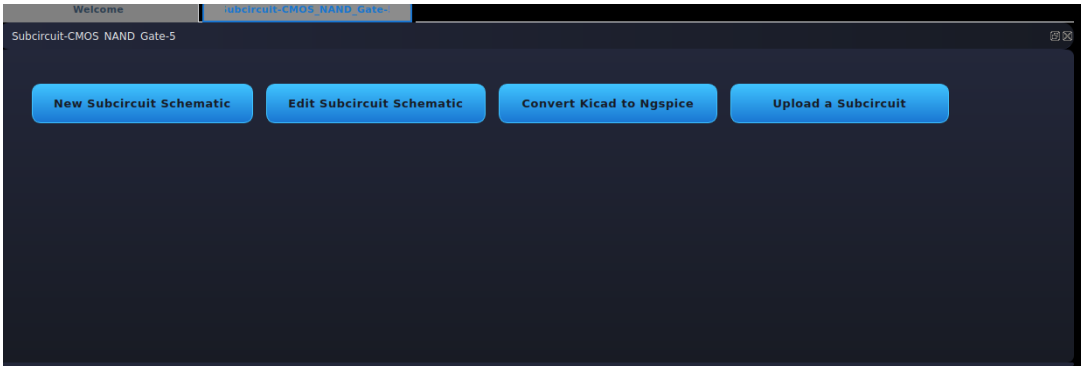

Figure 4.16: Subcircuit in eSim:Dark Mode

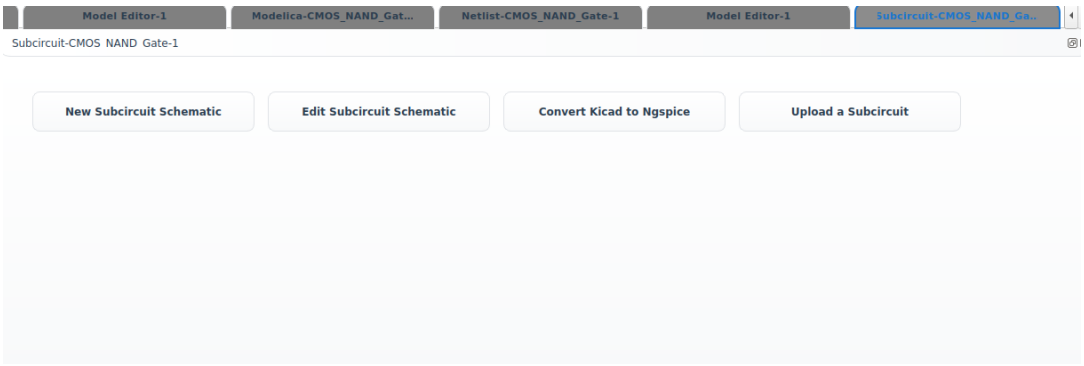

Figure 4.17: Subcircuit in eSim:Light Mode

4.4.6 Modelica Converter

The Modelica Converter is a crucial utility within the eSim suite, enabling the transformation of `.cir.out` netlist files into Modelica-compliant formats for simulation and visualization. It bridges circuit simulations with the OpenModelica environment via OMEdit integration. The evolution of this component reflects significant visual and usability enhancements while maintaining robust technical functionality.

Legacy Interface Design (Initial Version)

Figure 4.18: Initial Modelica Converter Interface

The original Modelica Converter featured a utilitarian and functional interface but lacked modern styling:

- **Basic File Input**: A single text input field with netlist file path display and a "Browse Netlist (*.cir.out)" button.
- **Action Controls**: Convert, Load OMEdit, and Browse OM buttons aligned vertically with minimal styling.
- **Monochrome Scheme**: Grayscale background with default widgets, contributing to a dull user experience.
- **Lack of Visual Hierarchy**: No color-coded indicators or emphasized interactive components.

This design emphasized basic functionality but did not cater to modern interface expectations or accessibility standards.

Enhanced Light Mode Implementation

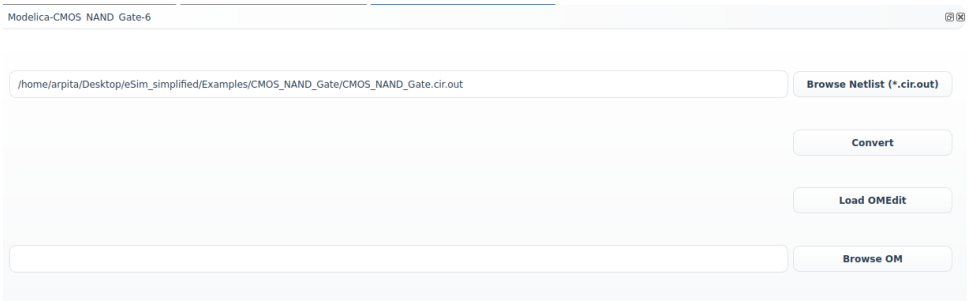

Figure 4.19: Modelica Converter – Light Mode UI

The light mode evolution introduced significant refinements:

- **Improved Alignment and Spacing**: Input and button elements are spaced evenly, contributing to a balanced layout.
- **Clean Aesthetic**: White backgrounds with soft shadows and updated fonts reflect a modern user interface design standard.
- **Improved Button Design**: Rounded buttons with subtle hover effects and consistent alignment across vertical panels.
- **Better Path Display**: Enhanced text visibility for file paths and user interaction clarity.

This version improved both usability and visual presentation, catering to users who prefer a professional and clean design environment.

Modern Dark Mode Implementation

The dark mode version provides an advanced and modern visual experience suitable for extended usage:

- **High-Contrast Theming**: Deep blue-gray backgrounds (#1B1F29 and #2C3440 range) combined with white and cyan-blue text.
- **Stylized Buttons**: Blue-toned action buttons with strong hover visibility and rounded corners, consistent with other dark-themed eSim modules.

Figure 4.20: Modelica Converter – Dark Mode UI

- **Accessible Typography**: Increased font contrast and bold file paths for better visual clarity.
- **Integrated UI Experience**: Dark mode aligns with other updated modules, delivering consistent user experience across tools.

This version exemplifies a polished EDA interface, prioritizing eye comfort, aesthetic appeal, and professional usability.

Comparative Analysis of UI Evolution

Visual Contrast and Accessibility

- **Legacy**: Flat, grayscale interface with no contrast differentiation.
- **Light Mode**: Clear visual separation of elements with balanced light tones.
- **Dark Mode**: High-contrast dark background with light text for reduced eye strain.

Button Design Evolution

- **Legacy**: Default OS-styled buttons with minimal design.
- **Light/Dark Modes**: Uniform button sizing, color-coded feedback, and modern round-cornered styles.

File Path Display

- **Legacy**: Low-contrast text and fixed-width box.
- **Updated Versions**: Full-width adaptive display with better visibility of long paths.

4.5 User Experience Considerations

4.5.1 Accessibility

The theme system adheres to the Web Content Accessibility Guidelines (WCAG) standards:

- **Contrast Ratio:** Both themes maintain a minimum 4.5:1 contrast ratio for normal text.
- **Color Independence:** Information is not conveyed through color alone.
- **Focus Indicators:** Clear visual indicators are provided for keyboard navigation.
- **Consistent Navigation:** Theme changes do not affect navigation patterns.

4.5.2 Performance Optimization

The theme system is optimized for performance through the following strategies:

- **Efficient Style Application:** A single stylesheet application reduces rendering overhead.
- **Minimal State Management:** A simple boolean flag is used, reducing memory usage.
- **Cached Styles:** Pre-compiled stylesheets allow instant switching.
- **Selective Updates:** Only affected components are updated during theme changes.

4.5.3 Advanced Features

Gradient Effects

Both themes utilize CSS gradients to create depth and visual interest:

```
/* Dark theme gradient */
background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
  stop:0 #0a0e1a, stop:0.3 #1a1d29,
  stop:0.7 #1e2124, stop:1 #0f1419);

/* Light theme gradient */
background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
  stop:0 #ffffff, stop:1 #f8f9fa);
```

Listing 4.7: Gradient Implementation

Interactive States

The system includes comprehensive styling for various interactive states:

1. **Hover States:** Provide immediate visual feedback.
2. **Active States:** Indicate current user interaction.
3. **Focus States:** Support keyboard navigation.
4. **Auto-Detection:** Automatically switches themes based on system preferences.
5. **Disabled States:** Clearly indicate unavailable actions.

4.6 Keyboard Shortcuts Implementation in eSim

The implementation of keyboard shortcuts in eSim represents a significant enhancement to the user experience, providing quick access to frequently used functions and improving overall productivity. This section details the comprehensive keyboard shortcut system implemented across various modules of the eSim application, including font size adjustment capabilities and functional shortcuts for all major tools.

The keyboard shortcuts system was designed with the following objectives:

- Improve user productivity through quick access to common functions
- Provide intuitive key combinations following standard conventions
- Enhance accessibility for power users
- Maintain consistency across the application interface

4.6.1 Font Size Adjustment System

Overview

The font size adjustment system allows users to dynamically modify the text size across the entire application interface. This feature is particularly beneficial for users working on different display sizes or those with visual accessibility requirements.

Implementation Details

The font size adjustment system is implemented through three main functions that work together to provide a seamless user experience:

```
# Add keyboard shortcuts for font size adjustment
QtWidgets.QShortcut(QtGui.QKeySequence("Ctrl++"), self, self.
    increase_font_size)
QtWidgets.QShortcut(QtGui.QKeySequence("Ctrl+-"), self, self.
    decrease_font_size)
QtWidgets.QShortcut(QtGui.QKeySequence("Ctrl+0"), self, self.
    reset_font_size)
```

Listing 4.8: Font Size Adjustment Shortcuts Registration

Font Size Control Functions

The system includes three primary control functions for font size manipulation:

```
def increase_font_size(self):
    """Increase font size for all widgets"""
    if self.toolbar_font_size < 20:  # Set maximum size
        self.toolbar_font_size += 1
        self.text_font_size += 1
        self.update_font_sizes()
        self.obj_appconfig.print_info("Font size increased")

def decrease_font_size(self):
    """Decrease font size for all widgets"""
    if self.toolbar_font_size > 8:  # Set minimum size
        self.toolbar_font_size -= 1
        self.text_font_size -= 1
        self.update_font_sizes()
        self.obj_appconfig.print_info("Font size decreased")

def reset_font_size(self):
    """Reset font sizes to default"""
    self.toolbar_font_size = 11
    self.text_font_size = 11
    self.update_font_sizes()
    self.obj_appconfig.print_info("Font size reset to default")
```

Listing 4.9: Font Size Control Functions

4.6.2 Functional Shortcuts Implementation

Theme Toggle Shortcut

The theme toggle functionality provides users with a quick way to switch between light and dark modes:

```
self.theme_toggle.setShortcut('Ctrl+T')
self.theme_toggle.setToolTip('Switch to Dark Mode (Ctrl+T)')
self.theme_toggle.triggered.connect(self.toggle_theme)
```

Listing 4.10: Theme Toggle Shortcut

Toolbar Shortcuts Configuration

The toolbar shortcuts provide quick access to all major eSim functions. Each shortcut is carefully chosen to avoid conflicts and follow intuitive patterns:

```
# KiCad Schematic Editor
self.kicad.setShortcut('Ctrl+K')
self.kicad.setToolTip('Open Schematic (Ctrl+K)')
self.kicad.triggered.connect(self.obj_kicad.openSchematic)

# KiCad to Ngspice Converter
self.conversion.setShortcut('Ctrl+G')
self.conversion.setToolTip('Convert KiCad to Ngspice (Ctrl+G)')
self.conversion.triggered.connect(self.obj_kicad.openKicadToNgspice)

# Ngspice Simulation
self.ngspice.setShortcut('Ctrl+M')
self.ngspice.setToolTip('Simulate (Ctrl+M)')
self.ngspice.triggered.connect(self.plotFlagPopBox)

# Model Editor
self.model.setShortcut('Ctrl+E')
self.model.setToolTip('Model Editor (Ctrl+E)')
self.model.triggered.connect(self.open_modelEditor)

# Subcircuit Editor
self.subcircuit.setShortcut('Ctrl+B')
self.subcircuit.setToolTip('Subcircuit (Ctrl+B)')
self.subcircuit.triggered.connect(self.open_subcircuit)

# NGHDL Interface
self.nghdl.setShortcut('Ctrl+H')
self.nghdl.setToolTip('NGHDL (Ctrl+H)')
self.nghdl.triggered.connect(self.open_nghdl)

# Makerchip-NgVeri
self.makerchip.setShortcut('Ctrl+V')
self.makerchip.setToolTip('Makerchip-NgVeri (Ctrl+V)')
self.makerchip.triggered.connect(self.open_makerchip)

# Modelica Converter
self.omedit.setShortcut('Ctrl+I')
self.omedit.setToolTip('Modelica Converter (Ctrl+I)')
self.omedit.triggered.connect(self.open_OMedit)

# OM Optimisation
self.omoptim.setShortcut('Ctrl+Y')
self.omoptim.setToolTip('OM Optimisation (Ctrl+Y)')
self.omoptim.triggered.connect(self.open_OMoptim)

# Schematics Converter
self.conToeSim.setShortcut('Ctrl+J')
self.conToeSim.setToolTip('Schematics converter (Ctrl+J)')
self.conToeSim.triggered.connect(self.open_conToeSim)
```

Listing 4.11: Comprehensive Toolbar Shortcuts Implementation

4.6.3 Keyboard Shortcuts Reference

Font Management Shortcuts

Table 4.3: Font Size Management Shortcuts

Shortcut	Function	Description
Ctrl++	Increase Font Size	Increases font size by 1 point (max: 20pt)
Ctrl+-	Decrease Font Size	Decreases font size by 1 point (min: 8pt)
Ctrl+0	Reset Font Size	Resets font size to default (11pt)

Application Function Shortcuts

Table 4.4: Application Function Shortcuts

Shortcut	Function	Module
Ctrl+T	Theme Toggle	Interface
Ctrl+K	Open Schematic	KiCad
Ctrl+G	Convert KiCad to Ngspice	KiCad Converter
Ctrl+M	Simulate	Ngspice
Ctrl+E	Model Editor	Model Management
Ctrl+B	Subcircuit	Subcircuit Editor
Ctrl+H	NGHDL	NGHDL Interface
Ctrl+V	Makerchip-NgVeri	Verilog Tools
Ctrl+I	Modelica Converter	Modelica Tools
Ctrl+Y	OM Optimisation	Optimization Tools
Ctrl+J	Schematics Converter	Conversion Tools

4.7 Plot GUI Enhancements

The plotting interface represents a critical component of the circuit simulation environment, providing users with comprehensive visualization capabilities for analyzing simulation results. This section presents the implementation of an advanced plotting system that supports both dark and light themes, offering enhanced user experience through modern UI design patterns and comprehensive data visualization tools.

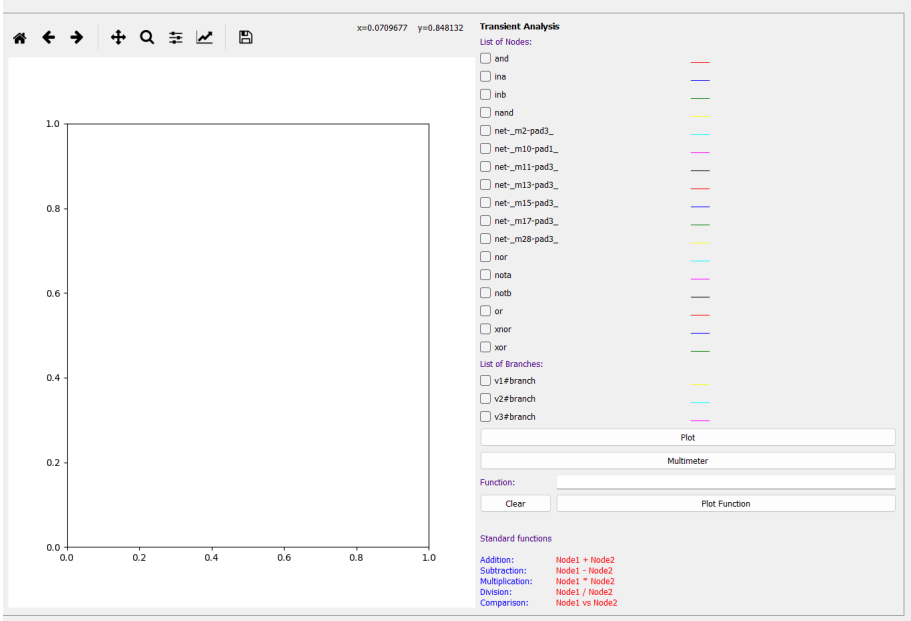

Figure 4.21: Previous Plot

- **Theme Management**: Dynamic switching between dark and light themes
- **Data Processing**: Integration with `DataExtraction` class for simulation data handling
- **Visualization Engine**: matplotlib-based plotting with custom styling
- **User Interface**: PyQt5-based controls with modern styling
- **Analysis Tools**: Built-in multimeter and mathematical functions

4.7.1 Class Structure

```
class plotWindow(QtWidgets.QMainWindow):
    """
    This class defines python plotting window, its features, buttons,
    colors, AC and DC analysis, plotting etc.
    """

    # Class variable to store the single instance
    instance = None

    @classmethod
    def add_output(cls, fpath, projectName, is_dark_theme=True):
        """Static method to manage plot window instances."""
        if cls.instance is None:
            cls.instance = cls(fpath, projectName, is_dark_theme)
        else:
            # Update existing instance with new data
            cls.instance.fpath = fpath
            cls.instance.projectName = projectName
            cls.instance.is_dark_theme = is_dark_theme
            cls.instance.obj_dataext = DataExtraction()
            cls.instance.plotType = cls.instance.obj_dataext.openFile(fpath
                )
            cls.instance.obj_dataext.computeAxes()
            cls.instance.a = cls.instance.obj_dataext.numVals()
            cls.instance.createMainFrame()
```

```
25
26          return cls.instance
```

Listing 4.12: Core Class Definition

4.7.2 Theme Implementation

Dynamic Theme Switching

The theme switching mechanism allows users to toggle between dark and light modes seamlessly:

```
def toggle_theme(self):
    """Toggle between light and dark themes."""
    self.is_dark_theme = not self.is_dark_theme
    self.setStyleSheet(DARK_STYLESHEET if self.is_dark_theme else
        LIGHT_STYLESHEET)
    self.update_plot_theme()

def update_plot_theme(self):
    """Update plot colors based on current theme."""
    if self.is_dark_theme:
        bg_color = DARK_BLUE
        text_color = TEXT_COLOR
        accent_color = ACCENT_BLUE
        grid_color = BORDER_COLOR
        function_color = TEXT_COLOR
        # Set a bright color cycle for plot lines in dark mode
        self.axes.set_prop_cycle(cycler('color',
            ['#00eaff', '#ff6b6b', '#ffe156', '#6bffb4',
             '#a55eea', '#fd79a8', '#ffb347', '#f9ca24',
             '#4ecdc4', '#45b7d1']))
    else:
        bg_color = LIGHT_BG
        text_color = LIGHT_TEXT
        accent_color = LIGHT_ACCENT
        grid_color = LIGHT_BORDER
        function_color = LIGHT_TEXT
        # Same color cycle for consistency
        self.axes.set_prop_cycle(cycler('color',
            ['#00eaff', '#ff6b6b', '#ffe156', '#6bffb4',
             '#a55eea', '#fd79a8', '#ffb347', '#f9ca24',
             '#4ecdc4', '#45b7d1']))
```

Listing 4.13: Theme Toggle Implementation

4.7.3 User Interface Components

Main Frame Creation

The main interface is constructed through the `createMainFrame` method, which establishes the primary layout and integrates all components:

```
def createMainFrame(self):
    self.mainFrame = QtWidgets.QWidget()
    self.dpi = 100

    # Create matplotlib figure with theme-appropriate background
    if self.is_dark_theme:
        self.fig = Figure((7.0, 7.0), dpi=self.dpi, facecolor="#000000")
    else:
        self.fig = Figure((7.0, 7.0), dpi=self.dpi)
```

```
10
11      self.canvas = FigureCanvas(self.fig)
12      self.canvas.setParent(self.mainFrame)
13      self.axes = self.fig.add_subplot(111)
14
15      # Configure axes for current theme
16      if self.is_dark_theme:
17          self.axes.set_facecolor("#000000")
18          self.axes.tick_params(colors="#f0f6fc", labelsize=12)
19          self.axes.xaxis.label.set_color("#f0f6fc")
20          self.axes.yaxis.label.set_color("#f0f6fc")
21          self.axes.title.set_color("#f0f6fc")
22          for spine in self.axes.spines.values():
23              spine.set_color("#1f6feb")
24              spine.set_linewidth(2)
```

Listing 4.14: Main Frame Construction

Custom Toolbar Implementation

The plotting interface features a custom toolbar that replaces the default matplotlib navigation toolbar with a theme-consistent design:

```
# Custom toolbar for both dark and light mode
self.navToolBar.hide()
custom_toolbar = QtWidgets.QWidget()
custom_toolbar_layout = QtWidgets.QHBoxLayout()
custom_toolbar_layout.setContentsMargins(0, 0, 0, 0)
custom_toolbar_layout.setSpacing(0)

for action in self.navToolBar.actions():
    if action.isSeparator() or action.icon().isNull():
        continue

    btn = QtWidgets.QToolButton()
    btn.setDefaultAction(action)
    btn.setIcon(action.icon())
    btn.setToolTip(action.toolTip())

    # Apply theme-appropriate styling
    if self.is_dark_theme:
        btn.setStyleSheet('''
            QToolButton {
                background-color: #23272e;
                border: 2px solid #30363d;
                border-radius: 8px;
                padding: 4px;
                margin: 0px;
                min-width: 28px;
                min-height: 28px;
                font-size: 13px;
                color: #f0f6fc;
            }
            QToolButton:hover {
                background-color: #1f6feb;
                border-color: #388bfd;
            }
        ''')
```

Listing 4.15: Custom Toolbar Creation

4.7.4 Multimeter Integration

The plotting interface includes integrated multimeter functionality for detailed signal analysis:

```
# Multimeter widget styles for dark theme
DARK_MULTIMETER_STYLE = f"""
    QWidget {{
        background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
            stop:0 {GRADIENT_START}, stop:1 {GRADIENT_END});
        color: {TEXT_COLOR};
        border: 2px solid {BORDER_COLOR};
        border-radius: 8px;
        padding: 10px;
    }}
    QLabel[class="value"] {{
        color: {ACCENT_BLUE};
        font-size: 16px;
        font-weight: bold;
    }}
"""

# Light theme multimeter styling
LIGHT_MULTIMETER_STYLE = f"""
    QWidget {{
        background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
            stop:0 {LIGHT_GRADIENT_START}, stop:1 {LIGHT_GRADIENT_END});
        color: {LIGHT_TEXT};
        border: 2px solid {LIGHT_BORDER};
        border-radius: 8px;
        padding: 10px;
    }}
    QLabel[class="value"] {{
        color: {LIGHT_ACCENT};
        font-size: 16px;
        font-weight: bold;
    }}
"""
```

Listing 4.16: Multimeter Styling

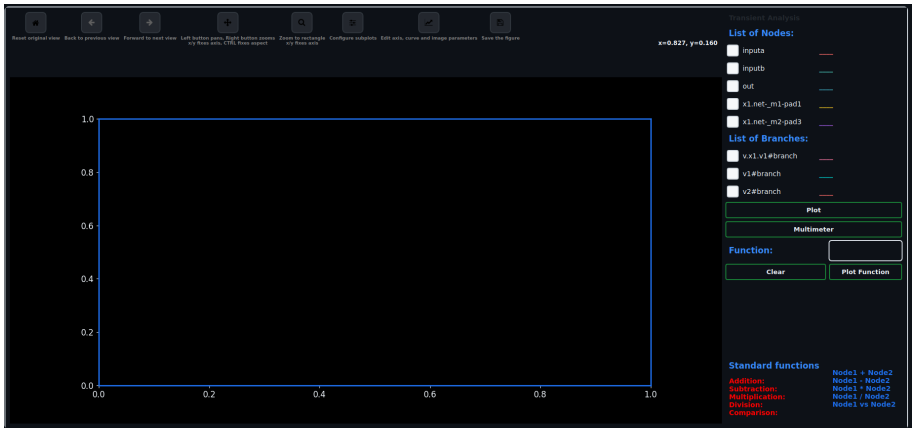

Figure 4.22: Dark Theme Interface - Optimized for low-light environments with high contrast elements

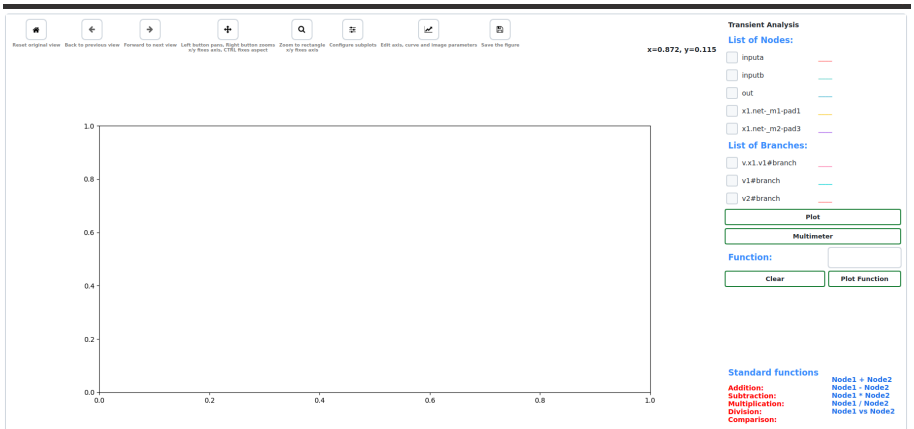

Figure 4.23: Light Theme Interface - Clean, professional appearance suitable for documentation

Feature Highlights

The plotting interface successfully implements:

- **Dynamic Theming**: Seamless switching between dark and light modes
- **Modern UI Design**: Color schemes with gradients and smooth transitions
- **Professional Styling**: Consistent visual design across all components
- **Visual Improvements**: Improved axis management and legend formatting, interactive plot features

Chapter 5

Project Section Redesign

5.1 Introduction

This chapter presents the implementation of a project file management system that enables users to open project files in a tabbed interface within the main application window. The system is designed to enhance user productivity by providing easy access to all files within a project through a single, organized interface.

5.2 Previous File Structure and Challenges

The initial implementation of the project management system utilized a traditional tree-based file structure interface, as illustrated in Figure. This hierarchical approach, while conceptually familiar to users accustomed to traditional file systems, presented several significant challenges that impacted user productivity and overall system efficiency in the context of modern software development workflows.

5.2.1 Navigation Complexity and User Experience Issues

The tree structure required users to navigate through multiple levels of nested directories, making file access time-consuming and cumbersome. Users had to expand and collapse various project folders to locate specific files, which became increasingly difficult as project complexity grew.

The interface displayed all files in a vertical list format, consuming substantial screen real estate and limiting the amount of visible content. As projects expanded to include numerous files and subdirectories, the interface became cluttered and difficult to navigate, with scroll bars frequently required to access files located at the bottom of long directory listings.

5.2.2 Scalability and Performance Limitations

The tree-based approach suffered from significant scalability issues that became more pronounced as project size increased. The vertical layout meant that users often had to scroll extensively to locate files, particularly when working with projects containing multiple components such as the `logicgates`, `Half_Adder`, and `comparator` directories shown in the original design.

5.2.3 Sequential Access Limitations

The lack of simultaneous file access was another critical limitation that significantly impacted development workflows. Users could only view one file at a time, forcing them to close current files before opening new ones. This sequential access pattern disrupted the natural workflow of software development, where developers often need to reference multiple files simultaneously for cross-referencing, debugging, and code comparison purposes.

The inability to maintain multiple file contexts simultaneously meant that developers had to rely heavily on memory when working with related files. This limitation was particularly

problematic when debugging complex issues that spanned multiple files or when implementing features that required coordination between different project components. The constant need to switch between files also increased the likelihood of losing track of important changes or forgetting to save modifications before switching contexts.

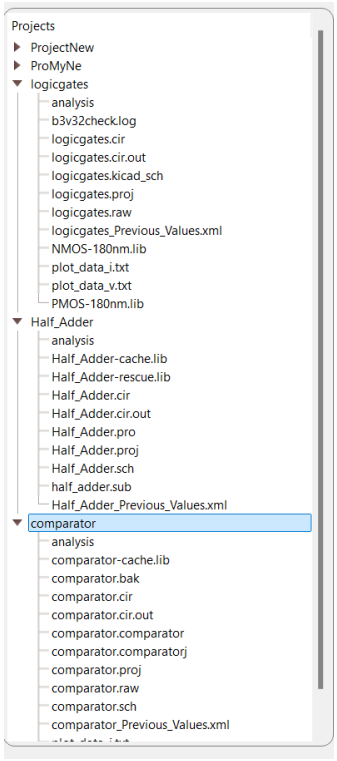

Figure 5.1: Previous project section

5.3 System Architecture and Design

5.3.1 Design Goals

The primary objectives of this implementation are:

- Provide intuitive project file navigation
- Enable simultaneous editing of multiple files within a project
- Maintain clean separation between project management and file editing
- Ensure seamless integration with existing application components

5.3.2 Component Architecture

The system consists of three main components:

1. **ProjectExplorer**: Manages project tree structure and handles user interactions
2. **MainView**: Coordinates between components and manages the tabbed interface
3. **DockArea**: Provides the docking framework for organizing tabs

5.4 Implementation of Tab-Based File Handling

To address the comprehensive limitations of the tree-based structure, a fundamental redesign was implemented featuring a modern tab-based file handling system, as demonstrated in Figure. This new approach represents a paradigm shift from traditional file management to a more contemporary, efficiency-focused interface design that aligns with modern software development practices and user expectations.

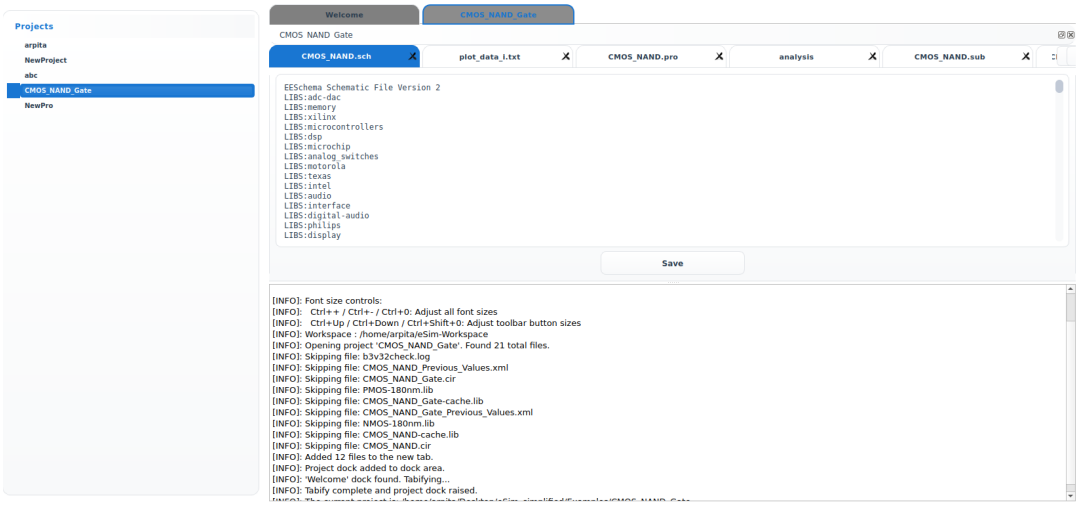

Figure 5.2: Redesigned Project Section in light mode

5.4.1 Core Architecture and Design Philosophy

The tab-based system fundamentally transforms the user experience by enabling simultaneous access to multiple files within a single, unified interface. Rather than replacing the current view with each new file selection, the system creates individual tabs for each opened file, maintaining all active contexts simultaneously. This approach eliminates the need for constant navigation between files and provides immediate access to previously opened documents through simple tab switching.

5.4.2 Spatial Efficiency and Screen Real Estate Optimization

The new tab-based interface represents a significant improvement in spatial efficiency compared to the previous design. The horizontal tab arrangement also scales better with project complexity. While the tree structure became increasingly unwieldy as projects grew, the tab system maintains a consistent, manageable interface regardless of the number of files in the project. Users can easily identify and access any open file through the tab interface, eliminating the need to navigate through complex directory hierarchies repeatedly.

5.5 File Management Features

The redesigned interface incorporates several enhanced file management features that significantly improve user experience and operational efficiency. These features were specifically designed to address common pain points identified in the previous system while introducing modern usability standards that align with contemporary software development practices.

5.5.1 Advanced Tab Closure Mechanisms

Each tab is equipped with a dedicated close button positioned at the right edge of the tab header, following established user interface conventions and providing users with intuitive control over

their workspace. The close button offers precise control over individual file closures without affecting other open tabs, allowing users to maintain their preferred set of active files while removing unnecessary ones from their workspace.

5.5.2 Integrated Save Functionality and State Management

A prominent "Save" button has been strategically positioned at the end of each file tab, providing immediate access to file preservation functionality. This placement ensures that users can quickly save their work whenever they edit something tthis button get enabled else disable help users to save and edit files easily.

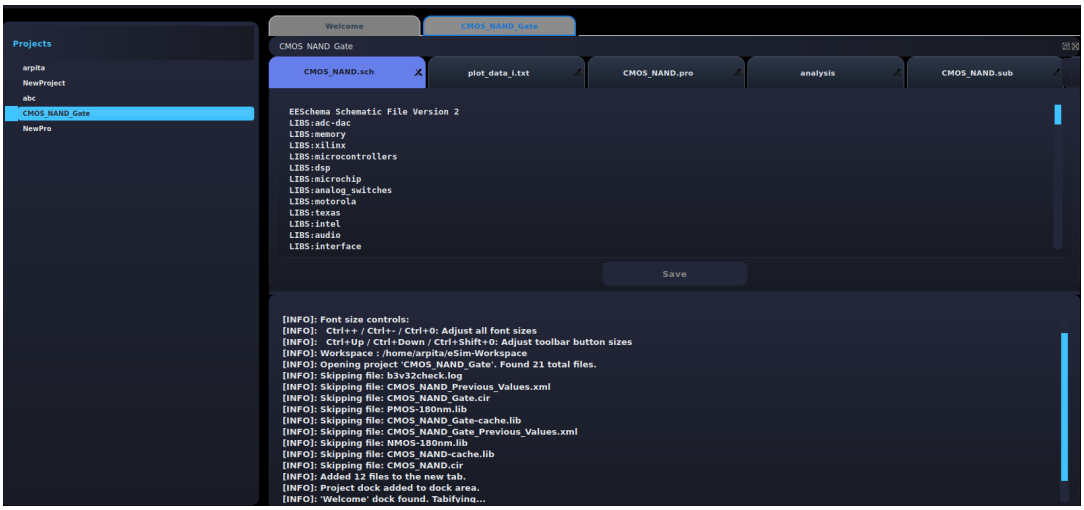

Figure 5.3: Redesigned Project Section in dark mode

5.6 Implementation Details

The implementation involves modifications to the following files:

- `src/frontEnd/ProjectExplorer.py`: Enhanced with signal emission
- `src/frontEnd/Application.py`: Added tab management functionality
- `src/frontEnd/DockArea.py`: Utilized for dock widget management

5.6.1 ProjectExplorer Modifications

The `ProjectExplorer` class was enhanced to emit a custom signal when a project is selected:

```
class ProjectExplorer(QtWidgets.QWidget):
    """
    Enhanced ProjectExplorer with project opening capabilities
    """
    projectOpened = QtCore.pyqtSignal(str, str, list)

    def __init__(self, is_dark_theme=False):
        # ... existing initialization code ...
        self.treewidget.itemClicked.connect(self.handleItemClicked)
```

Listing 5.1: ProjectExplorer Signal Declaration

5.6.2 Event Handling Implementation

The click event handler distinguishes between project folders and individual files:

```
def handleItemClicked(self, item, column):
    """
    Handle tree widget item clicks

    Args:
        item: QTreeWidgetItem that was clicked
        column: Column index of the click
    """
    # Check if the clicked item is a project (has children)
    if item.childCount() > 0:
        projectName = item.text(0)
        projectPath = item.text(1)
        files = [item.child(i).text(0) for i in range(item.childCount())]
        self.projectOpened.emit(projectName, projectPath, files)
    else:
        # If it's a file, use the existing behavior
        self.openProject()
```

Listing 5.2: Item Click Handler

5.6.3 Original openProject Method Enhancement

The existing `openProject` method was modified to support both file and project operations:

```
def openProject(self):
    """
    Open selected project or file with enhanced functionality
    """
    self.indexItem = self.treewidget.currentIndex()
    filename = str(self.indexItem.data())
    self.filePath = str(
        self.indexItem.sibling(self.indexItem.row(), 1).data()
    )

    if (os.path.isfile(str(self.filePath))):
        # Handle individual file opening
        self.fopen = open(str(self.filePath), 'r')
        lines = self.fopen.read()

        self.textwindow = QtWidgets.QWidget()
        self.textwindow.setMinimumSize(600, 500)
        self.textwindow.setGeometry(QtCore.QRect(400, 150, 400, 400))
        self.textwindow.setWindowTitle(filename)

        # Apply premium dark theme styling
        premium_dark_stylesheet = """
        QWidget {
            background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
                stop:0 #0a0e1a, stop:0.3 #1a1d29,
                stop:0.7 #1e2124, stop:1 #0f1419);
            color: #e8eaed;
            font-family: 'Fira Sans', 'Inter', 'Segoe UI',
                         'Roboto', 'Arial', sans-serif;
            font-size: 15px;
            font-weight: 500;
        }
        QTextEdit {
            background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
```

```
                stop:0 #23273a, stop:1 #181b24);
            color: #e8eaed;
            border: 1px solid #23273a;
            border-radius: 10px;
            padding: 16px 20px;
            font-weight: 500;
            font-size: 15px;
            selection-background-color: #40c4ff;
            selection-color: #181b24;
        }
        QScrollBar:vertical, QScrollBar:horizontal {
            background: #23273a;
            border-radius: 6px;
            margin: 0;
        }
        QScrollBar::handle:vertical, QScrollBar::handle:horizontal {
            background: #40c4ff;
            border-radius: 6px;
            min-height: 30px;
            min-width: 30px;
            margin: 2px;
        }
        QScrollBar::handle:vertical:hover,
        QScrollBar::handle:horizontal:hover {
            background: #1976d2;
        }
        QPushButton {
            background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
                stop:0 #40c4ff, stop:1 #1976d2);
            color: #181b24;
            border: 1px solid #40c4ff;
            padding: 12px 24px;
            border-radius: 10px;
            font-weight: 700;
            font-size: 15px;
            letter-spacing: 0.5px;
        }
        QPushButton:hover {
            background: #1976d2;
            color: #fff;
            border: 1.5px solid #1976d2;
        }
        QPushButton:pressed {
            background: #23273a;
            color: #40c4ff;
            border: 1.5px solid #40c4ff;
        }
        """
        self.textwindow.setStyleSheet(premium_dark_stylesheet)

        self.text = QtWidgets.QTextEdit()
        self.save = QtWidgets.QPushButton('Save and Exit')
        self.save.setDisabled(True)

        self.text.setText(lines)
        self.text.textChanged.connect(self.enable_save)

        vbox_main = QtWidgets.QVBoxLayout(self.textwindow)
        vbox_main.addWidget(self.text)
        vbox_main.addWidget(self.save)
        self.save.clicked.connect(self.save_data)
```

```
96
97          self.textwindow.show()
98      else:
99          # Handle project directory opening
100         self.refreshProject(self.filePath)
101         projectName = os.path.basename(self.filePath)
102         files = [f for f in os.listdir(self.filePath)
103                  if os.path.isfile(os.path.join(self.filePath, f))]
104         self.projectOpened.emit(projectName, self.filePath, files)
105         self.obj_appconfig.print_info(
106             'The current project is: ' + self.filePath
107         )
108
109         self.obj_appconfig.current_project["ProjectName"] = str(
110             self.filePath)
111         (
112             self.obj_appconfig.
113             proc_dict[self.obj_appconfig.current_project['ProjectName']]
114         ) = []
115         if (
116             self.obj_appconfig.current_project['ProjectName'] not in
117             self.obj_appconfig.dock_dict
118         ):
119             (
120                 self.obj_appconfig.
121                 dock_dict[
122                     self.obj_appconfig.current_project['ProjectName']]
123             ) = []
```

Listing 5.3: Enhanced openProject Method

5.6.4 MainView Integration

The `MainView` class was enhanced to handle the project opening signal:

```
1  def __init__(self, is_dark_theme=False, *args):
2      # ... existing initialization code ...
3      self.obj_projectExplorer = ProjectExplorer.ProjectExplorer(
4          self.is_dark_theme)
5      self.obj_dockarea = DockArea.DockArea(self.is_dark_theme)
6
7      # Connect the projectOpened signal to the handler
8      self.obj_projectExplorer.projectOpened.connect(self.openProjectInTabs)
9
10     # ... rest of initialization code ...
```

Listing 5.4: MainView Signal Connection

5.6.5 Tabbed Interface Implementation

The core functionality for creating tabbed interfaces is implemented in the `openProjectInTabs` method:

```
1  def openProjectInTabs(self, projectName, projectPath, fileList):
2      """
3      Create a tabbed interface for project files
4
5      Args:
6          projectName (str): Name of the project
7          projectPath (str): Path to the project directory
```

```
        fileList (list): List of file names in the project
    """
    # Create a dock widget for the project
    projectDock = QtWidgets.QDockWidget(projectName, self)
    projectDock.setAllowedAreas(QtCore.Qt.TopDockWidgetArea)

    # Create the tab widget
    tabWidget = QtWidgets.QTabWidget()
    tabWidget.setTabsClosable(True)
    tabWidget.tabCloseRequested.connect(
        lambda index: tabWidget.removeTab(index)
    )

    # Process each file in the project
    for f_name in fileList:
        filePath = os.path.join(projectPath, f_name)

        # Filter out binary or cache files
        if f_name.endswith(('.lib', '.sch-bak', '.kicad_pcb-bak',
                            '.net', '.xml', '.cir', '.log')):
            continue

        try:
            # Read file content
            with open(filePath, 'r', errors='ignore') as f:
                content = f.read()

            # Create text editor for the file
            editor = QtWidgets.QTextEdit()
            editor.setText(content)

            # Apply monospace font for code readability
            font = QtGui.QFont()
            font.setFamily("monospace")
            font.setStyleHint(QtGui.QFont.Monospace)
            editor.setFont(font)

            # Add tab for this file
            tabWidget.addTab(editor, f_name)

        except Exception as e:
            self.obj_appconfig.print_info(
                f"Error reading file {filePath}: {e}"
            )

    # Add the tab widget to the dock area if it has tabs
    if tabWidget.count() > 0:
        projectDock.setWidget(tabWidget)
        self.obj_dockarea.addDockWidget(
            QtCore.Qt.TopDockWidgetArea, projectDock
        )

        # Tabify with the welcome widget if it exists
        if 'Welcome' in DockArea.dock:
            self.obj_dockarea.tabifyDockWidget(
                DockArea.dock['Welcome'], projectDock
            )

        # Bring the new dock to the front
        projectDock.raise_()
```

Listing 5.5: Tabbed Interface Creation

5.7 User Interface Enhancements

5.7.1 Tab Management Features

The tabbed interface includes several user-friendly features:

- **Closable Tabs**: Users can close individual file tabs
- **Monospace Font**: Code files are displayed with appropriate formatting
- **Theme Integration**: Tabs inherit the application's theme
- **Dock Integration**: Tabs are properly integrated with the dock system

5.7.2 Visual Styling

The implementation maintains consistency with the application's visual design:

- Dark theme support with gradient backgrounds
- Consistent color schemes across components
- Professional typography choices
- Responsive UI elements

The implemented project file management system successfully provides an intuitive and efficient way to work with project files through a tabbed interface. The modular design ensures that the system can be easily extended and maintained while providing a solid foundation for future enhancements.

Chapter 6

Conclusion

During the internship period, a comprehensive set of features and improvements were successfully implemented across the eSim platform, significantly enhancing both functionality and user experience. The implemented features can be categorized into several key areas:

6.0.1 Personal Growth and Development

This internship experience has been transformative in multiple aspects of professional and personal development:

Technical Competence

The hands-on experience with GUI development, theme implementation, and bug fixing has significantly enhanced technical skills. Working with real-world software architecture, dealing with legacy code, and implementing modern design patterns has provided invaluable practical experience that complements theoretical knowledge.

Problem-Solving Skills

Encountering and resolving complex technical challenges, from theme integration conflicts to cross-platform compatibility issues, has developed strong analytical and problem-solving capabilities. The experience of debugging intricate GUI issues and implementing user-centric solutions has built confidence in tackling challenging technical problems.

Project Management

Managing multiple concurrent features, prioritizing tasks based on user impact, has provided insight into effective project management practices. The experience of balancing feature development with bug fixes while maintaining code quality has been particularly valuable.

6.0.2 Contribution to Open Source Community

Knowledge Sharing

Contributing to the eSim project has provided an opportunity to give back to the open source community. The implemented features will benefit researchers, students, and professionals using eSim for circuit simulation and analysis. The comprehensive documentation and clean code implementation ensure that future contributors can build upon this work.

Community Engagement

Active participation in issue discussions, feature planning, and code reviews has fostered a deeper understanding of collaborative software development. The experience of working with diverse contributors and incorporating community feedback has been enriching and educational.

6.0.3 Future Aspirations

This internship has solidified interest in user interface development and open source contribution. The experience has highlighted the importance of user-centered design in software development and the satisfaction derived from creating tools that enhance productivity and accessibility.

Continued Contribution

The intention is to continue contributing to the eSim project and other open source initiatives. The foundation built during this internship provides a solid base for tackling more complex features and architectural improvements.

Professional Development

The skills developed during this internship - from technical competencies to project management and community engagement - will be valuable in future professional endeavors. The experience has provided a comprehensive understanding of the software development lifecycle and the importance of balancing technical excellence with user experience.

6.1 Future Scope for GUI Enhancement

6.1.1 Advanced Theme Customization

Personalization Features

- **Custom Color Schemes**: Allow users to create and save personalized color themes
- **Theme Marketplace**: Platform for sharing and downloading community-created themes

Advanced Visual Options

- **Layout Customization**: Drag-and-drop interface elements for personalized layouts
- **Workspace Profiles**: Different interface configurations for different types of work
- **Visual Density Options**: Compact and spacious layout alternatives

6.1.2 Enhanced Collaboration Features

Multi-user Support

- **Real-time Collaboration**: Multiple users working on the same project simultaneously
- **Version Control Integration**: Built-in Git support with visual diff tools

6.1.3 Advanced Simulation and Analysis

Enhanced Visualization

- **3D Visualization**: Three-dimensional representation of circuits and results
- **Interactive Plots**: Dynamic and interactive plotting capabilities
- **Animation Support**: Animated visualization of signal flows and circuit behavior

The GUI enhancements implemented during this internship represent more than just technical improvements it is the journey from identifying user points to implementing comprehensive solutions has been both challenging and rewarding. The positive feedback from users and the visible impact of the improvements validate the effort invested in this project.

Chapter 7

Bibliography

- FOSSEE Official Website, Available at:
  `https://fossee.in/about`
- eSim Official Website, Available at:
  `https://esim.fossee.in/`
- GitHub - FOSSEE eSim Repository, Available at:
  `https://github.com/FOSSEE/eSim`
- NgSpice Official Website, Available at:
  `https://ngspice.sourceforge.net/`
- KiCad Official Website, Available at:
  `https://www.kicad.org/`
- I. Antcheva, R. Brun, F. Rademakers, CERN, Geneva, Switzerland *Guidelines For Developing a Good GUI.* Available at:
  `https://cds.cern.ch/record/865663/files/p613.pdf`
- Web Accessibility Initiative. Tips for Designing. Available at:
  `https://www.w3.org/WAI/tips/designing/`