eSim Winter Internship 2025 Report

On

Development of a Cross-Platform Dockerized Launcher for eSim

Submitted by

Barun Kumar Pattanaik

B.Tech in CSE , VIT Bhopal University

Under the Guidance of

Prof. Prabhu Ramachandran

Principal Investigator

Department of aerospace engineering

Indian Institute of Technology Bombay

January 12, 2026

Acknowledgment

I would like to express my deepest gratitude to **Prof. Prabhu Ramachandran** for providing me the opportunity to be a part of the FOSSEE internship program and for supporting open-source engineering tool development at IIT Bombay. His guidance and vision have encouraged students and researchers to actively contribute to the open-source ecosystem.

I would also like to acknowledge **Prof. Kannan M. Moudgalya** for his foundational role in establishing and nurturing the FOSSEE initiative. His contributions toward open-source education and the creation of the FOSSEE fellowship framework have directly shaped the platform through which this internship was undertaken.

My sincere appreciation extends to my mentor, **Mr. 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**, **Ms. Shanthi Priya K**, and **Mr. Aditya M**, 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 real-world 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.

Abstract

This report details the creation of a cross-platform, containerized deployment solution for eSim, an open-source Electronic Design Automation (EDA) tool. Historically, eSim installation presented challenges, including OS-specific failures and dependency conflicts. This project overcame these issues by utilizing Docker technology.

A key deliverable is a custom Python-based launcher that provides a "one-click" installation experience for users on Windows, Linux, and macOS. Technical innovations include a multi-stage Docker build to optimize size, dynamic port allocation to prevent socket conflicts, and the integration of VNC and X11 display protocols for seamless Graphical User Interface (GUI) access.

The overall goal was to enhance eSim's accessibility for the global academic community, which was achieved through the final deliverables: automated CI/CD pipelines and standalone executables.

Contents

Chapter 1 Introduction

To update the eSim installer script to install eSim 2.4 on Ubuntu 23.04 (Lunar Lobster) without errors.

1.1 FOSSEE: Promoting Open-Source Software in Education

The Free/Libre and Open Source Software for Education (FOSSEE) project is an initiative of IIT Bombay [2]. It operates under the National Mission on Education through Information and Communication Technology (ICT) and receives funding from the Ministry of Education, Government of India. The project's main goal is to reduce reliance on proprietary software within academic and research settings by encouraging the use of Free and Open Source Software (FOSS) alternatives.

FOSSEE pursues this objective through a variety of initiatives:

- **Development and Enhancement:** Creating new open-source tools and improving existing ones to meet the evolving needs of academia and industry.
- **Workshops and Training:** Organizing training programs and workshops to promote the adoption of open-source software.
- **Collaborations:** Partnering with institutions, researchers, and professionals to integrate FOSS into mainstream education.
- **Translation and Documentation:** Translating open-source software documentation into various regional languages to improve accessibility.

These efforts ensure that high-quality software remains freely accessible, thus removing the financial obstacles associated with proprietary tools and empowering users to learn, contribute, and innovate.

1.2 eSim: An Open-Source Electronic Design Automation (EDA) Tool

One of FOSSEE's most notable contributions to the open-source community is eSim. Developed by IIT Bombay, eSim is a free and open-source EDA tool that provides a comprehensive environment for circuit design, simulation, analysis, and PCB design [1]. It integrates several existing FLOSS tools to offer a complete solution for electrical and electronics engineers.

Core Features of eSim

eSim is designed to offer functionalities comparable to commercial EDA tools like Xpedition, OrCAD, and HSPICE, but without the licensing costs. Its key features include:

- **Schematic Capture:** Users can create detailed circuit schematics via an interactive graphical interface.
- **Simulation and Analysis:** Integration with NgSpice facilitates transient, AC, and DC circuit analysis.
- **Mixed-Mode Simulation:** Powered by GHDL and Verilator, it supports VHDL and Verilog simulation.
- **PCB Design:** Seamless integration with KiCad, a powerful open-source PCB design tool, is included.
- **Embedded Systems Support:** Features for designing embedded systems, including the integration of microcontrollers, are offered.
- **Open-Source License:** Released under the GPL, guaranteeing unrestricted access and modification.

Integrated Components

eSim combines several open-source tools to deliver a powerful EDA suite:

- **KiCad:** Used for both schematic capture and PCB layout.
- **NgSpice:** A general-purpose circuit simulator for AC, DC, and transient analysis.
- **GHDL:** A VHDL simulator for digital circuit design verification.
- **Verilator:** A fast Verilog simulator commonly used for hardware verification.

By leveraging these tools, eSim stands as an affordable, flexible, and powerful alternative to proprietary EDA software, making it a preferred choice for students, researchers, and professionals in the field of circuit design and simulation.

However, while eSim is a powerful tool, its installation across various Ubuntu versions has presented difficulties. The following section explores these challenges and the resulting need for improvements to ensure a smooth user experience.

1.3 The Need for Containerization

While eSim acts as a powerful alternative to proprietary EDA tools, its dependence on specific versions of KiCad, Ngspice, and Python libraries creates a significant barrier to entry for students. Traditional installation methods (shell scripts) often fail due to:

- **OS Fragmentation:** Differences between Ubuntu 20.04, 22.04, and 23.04.
- **Library Conflicts:** System-wide Python updates breaking older eSim dependencies (e.g., distutils removal in Python 3.10+).
- **Platform Lock-in:** Native support for Windows and macOS is complex to maintain.

To address these challenges, this project introduces a **Dockerized approach**. By packaging the entire eSim environment into a container, we ensure that "if it runs on my machine, it runs on yours," regardless of the underlying operating system.

Chapter 2: Problem Statement

In this chapter, we analyze the limitations of existing installation methods and define the objectives for the Dockerized solution.

2.1 Challenges in Traditional Installation

The existing `install-eSim.sh` installation scripts, which use system-level package managers (apt, pip), present several maintenance and compatibility challenges. As documented in previous internship reports, these issues require frequent updates with every new Ubuntu release.

Key problems include:

- **Dependency Conflicts:** eSim relies on outdated packages (e.g., hdlparse 1.0.4) that are incompatible with current Python environments.
- **Operating System Inflexibility:** Scripts often fail across different Ubuntu versions due to changes in repository configurations.
- **Barriers for Non-Linux Users:** Windows and Mac users must resort to resource-intensive Virtual Machines (consuming 20GB+ storage), which can be intimidating for new users.

2.2 Approach: The "Zero-Friction" Launcher

To resolve these issues, we developed a custom **Python-based Launcher (run_esim_docker.py)** that abstracts the complexity of Docker.

Key Objectives:

1. **Simplified Setup and Execution:**
    a. **One-Click Execution:** Deliver the entire application logic within a standalone executable (.exe) to eliminate the need for users to have Python installed.
    b. **Eliminate Installation Steps:** Remove the requirement for users to manually install dependencies.
2. **Efficient Resource Management:**
    a. **Reduced Resource Usage:** Utilize a Docker container to ensure the environment is lightweight (~3.5GB), contrasting with the higher overhead of a traditional Virtual Machine (VM).

3. **Cross-Platform Accessibility:**
    a. **Containerization:** Bundle eSim into a single, comprehensive Docker image.
    b. **Cross-Platform Launcher:** Develop a dedicated standalone executable to manage and control the container across different operating systems.
4. **Seamless GUI Experience:**
    a. **Cross-Platform GUI:** Provide robust Graphical User Interface (GUI) access on Windows, Linux, and macOS by leveraging X11 and VNC technologies.
    b. **Display Automation:** Automatically support both native (X11) and browser-based (VNC) display modes for maximum flexibility.

Chapter 3: Background and Literature Review

This chapter explores the core technologies used in the project, specifically containerization and display forwarding protocols.

3.1 Docker and Containerization

Docker is a platform that uses containerization technology to package applications and their dependencies into isolated units called containers. Unlike virtual machines, containers share the host operating system's kernel, making them lightweight and fast [3].

Key advantages of containerization for this project:

- **Dependency isolation** - All eSim dependencies are bundled together
- **Consistency** - The same container runs identically on any system
- **Portability** - Users do not need to install complex dependencies manually
- **Version contro**l - Different versions can be maintained as separate images

3.2 Display Technologies: X11 and VNC

Since eSim is a GUI application, displaying its interface from within a Docker container requires special handling.

- **X11 Forwarding:** X11 is the standard display protocol on Linux systems. X11 forwarding allows GUI applications running remotely (or in containers) to display their windows on the local machine. On Linux, this works natively. On Windows, an X server like VcXsrv is required. On macOS, XQuartz provides X11 support.
- **VNC (Virtual Network Computing):** VNC is a graphical desktop sharing system that transmits keyboard and mouse input and receives screen updates over a network. For this project, TigerVNC runs inside the container, and noVNC provides browser-based access without requiring users to install a VNC client [4,5].

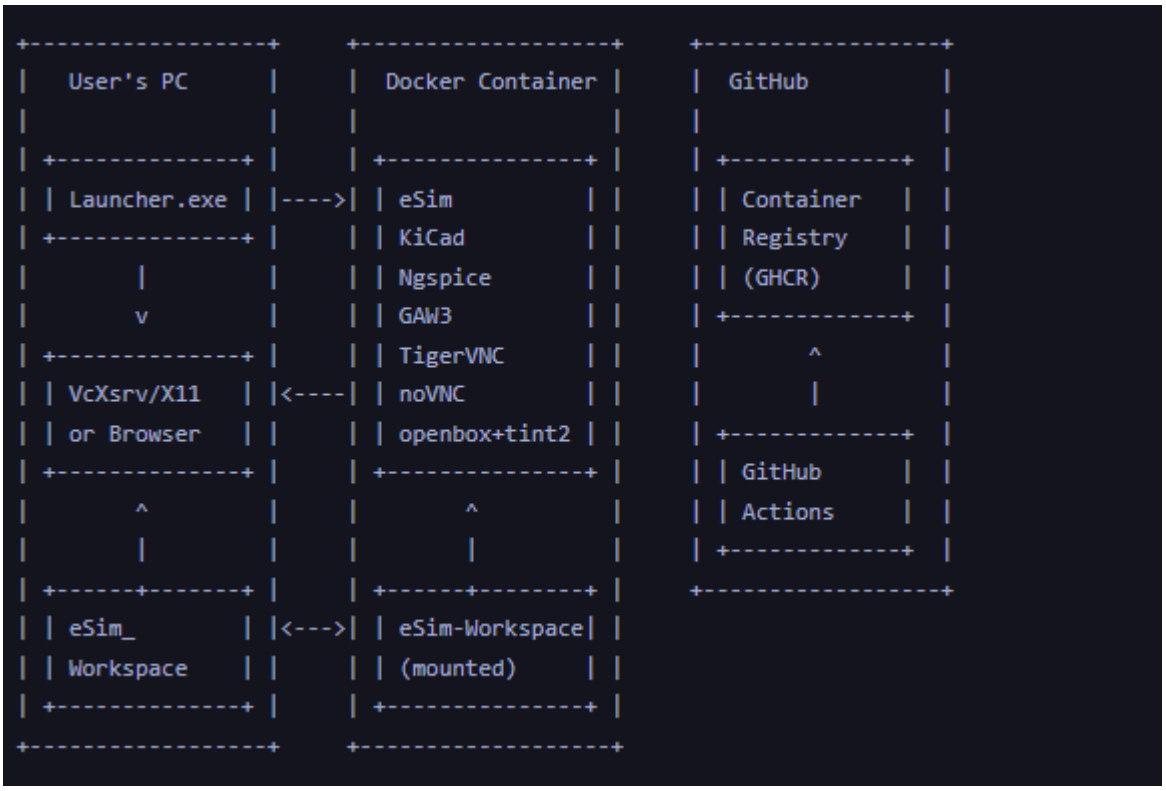
Chapter 4: System Design and Architecture

This chapter describes the design of the Docker image and the logic of the Python-based orchestrator.

4.1 Architecture Overview

The system consists of three distinct layers:

1. **Docker Image** - Contains eSim, KiCad, Ngspice, GAW3, and all required libraries. Built using a multi-stage Dockerfile and hosted on GitHub Container Registry.
2. **Python Launcher** - A command-line application that handles Docker operations, display configuration, and user interaction. Distributed as platform-specific executables.
3. **CI/CD Pipeline** - GitHub Actions workflow that automatically builds the Docker image and creates release executables when a new version is tagged.



[Architecture Diagram showing the flow between the Launcher, Docker, and the User's Display.]

4.2 Workflow Design

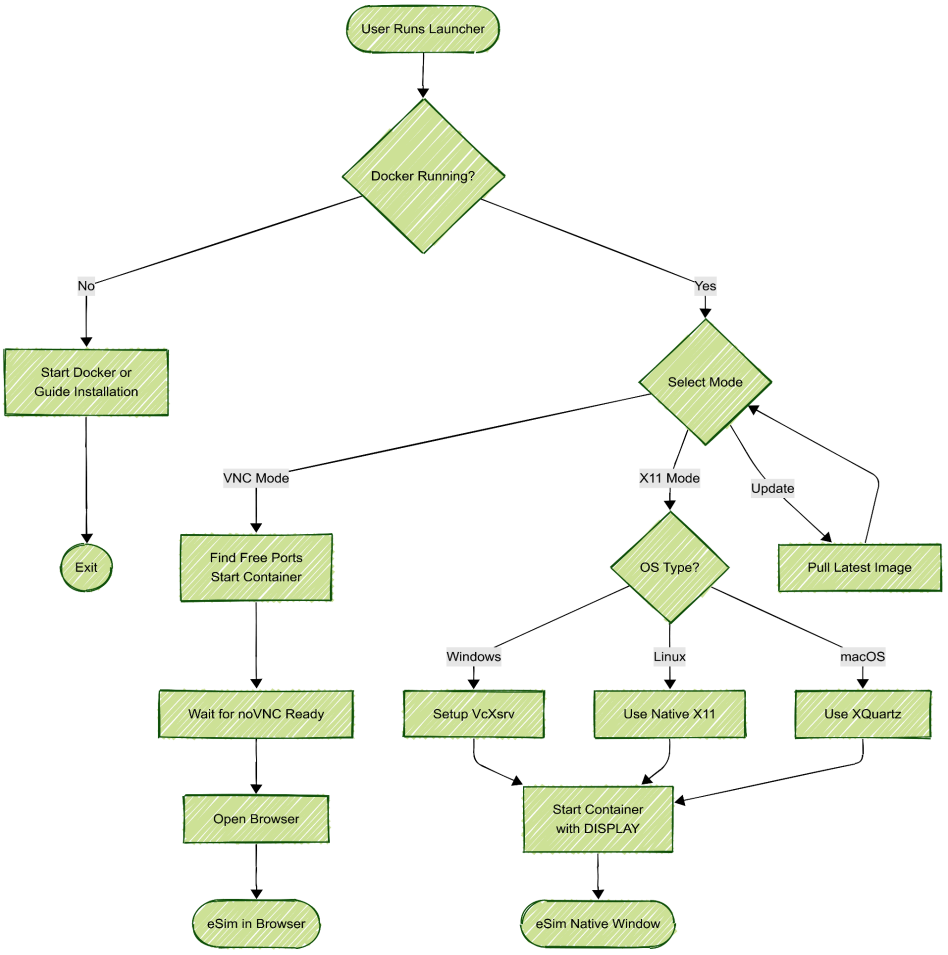
When the launcher is executed, it follows a rigorous pre-flight checklist:

VNC Mode Workflow:

- User selects VNC mode from the launcher menu
- Launcher checks for available ports (6080 for noVNC, 5901 for VNC)
- Docker container starts with port mappings and workspace mount
- Inside container: TigerVNC starts, followed by websockify and openbox
- Launcher polls the noVNC HTTP endpoint until it responds
- Browser opens automatically to the VNC URL
- User interacts with eSim through the browser

X11 Mode Workflow:

- User selects X11 mode from the launcher menu
- On Windows: Launcher ensures VcXsrv is installed and running
- DISPLAY environment variable is configured for the host
- Docker container starts with X11 socket forwarding
- eSim window appears directly on the host desktop



[Flowchart showing the launcher's decision-making process for VNC vs X11 modes.]

Chapter 5: Implementation

This chapter details the technical execution of the project, including Dockerfile optimization and CI/CD pipelines.

5.1 Multi-Stage Dockerfile Design

To minimize the image size, a **Multi-Stage Build** was implemented.

Stage 1: Builder

- The builder stage handles all compilation tasks:
- Installing build dependencies (gcc, make, cmake, etc.)
- Cloning the eSim repository from GitHub
- Setting up a Python virtual environment with required packages
- Compiling GAW3 (analog waveform viewer) from source

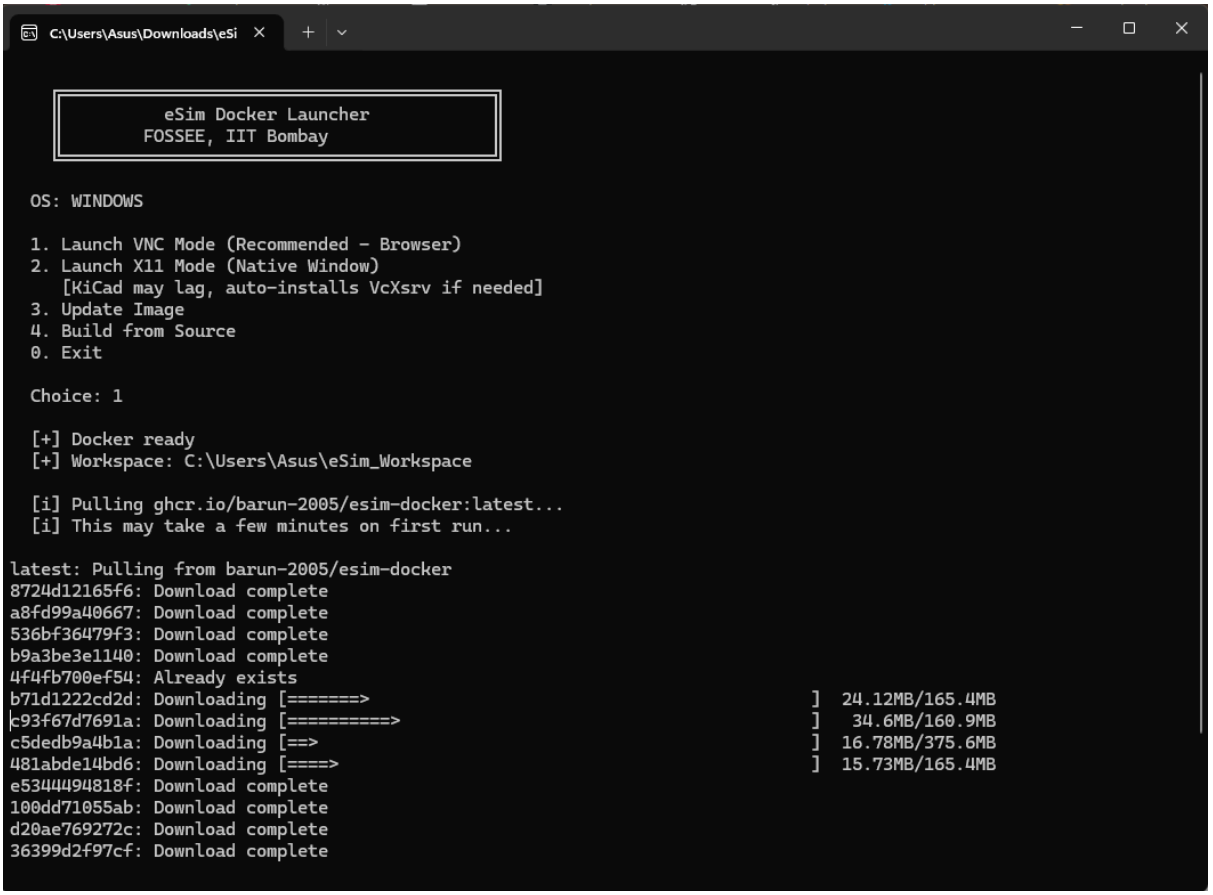
```
FROM ubuntu:22.04 AS builder
# Install build tools
RUN apt-get update && apt-get install -y \
   build-essential gcc g++ make cmake git \
   python3 python3-pip python3-venv \
   autoconf automake libtool
# Clone eSim
RUN git clone --depth 1 https://github.com/FOSSEE/eSim.git
/build/esim
# Setup Python virtual environment
RUN python3 -m venv /build/venv \
   && /build/venv/bin/pip install matplotlib numpy scipy PyQt5
# Build GAW3 from source
RUN git clone --depth 1
https://github.com/StefanSchippers/xschem-gaw.git /build/gaw3 \
   && cd /build/gaw3 && ./configure && make && make install
```

Stage 2: Runtime

The runtime stage starts fresh and copies only the necessary files:

- Installing runtime dependencies (KiCad, Ngspice, display libraries)
- Copying compiled GAW3 and eSim from the builder stage
- Setting up user configuration files
- Creating the startup script

```
FROM ubuntu:22.04
# Install runtime packages only
RUN apt-get update && apt-get install -y \
    kicad ngspice tigervnc-standalone-server novnc \
    openbox tint2 python3 python3-wxgtk4.0
# Copy from builder stage
COPY --from=builder /build/gaw3-install/usr/local /usr/local/
COPY --from=builder /build/esim /usr/local/esim
COPY --from=builder /build/venv /opt/venv
```



[SCREENSHOT : Terminal output showing the Docker build logs.]

5.2 Python Launcher Logic

The launcher script (run_esim_docker.py) is structured into several functional sections:

1. Helper Functions

Utility functions for formatted output and common operations:

```
def info(msg):
    print(f"  [i] {msg}")
def ok(msg):
    print(f"  [+] {msg}")
def warn(msg):
    print(f"  [!] {msg}")
```

2. OS Detection

Functions to detect the operating system and adjust behaviour accordingly:

```
def get_os():
    system = platform.system().lower()
    if system == "windows":
        return "windows"
    elif system == "darwin":
        return "macos"
    else:
        return "linux"
```

3. Docker Management

Functions for checking Docker status, pulling images, and running containers:

```
def docker_ok():
    try:
        subprocess.run(["docker", "info"],
capture_output=True)
        return True
    except:
        return False
def pull_image(image):
    subprocess.run(["docker", "pull", image])
```

4. Port Management

Dynamic port allocation to avoid conflicts:

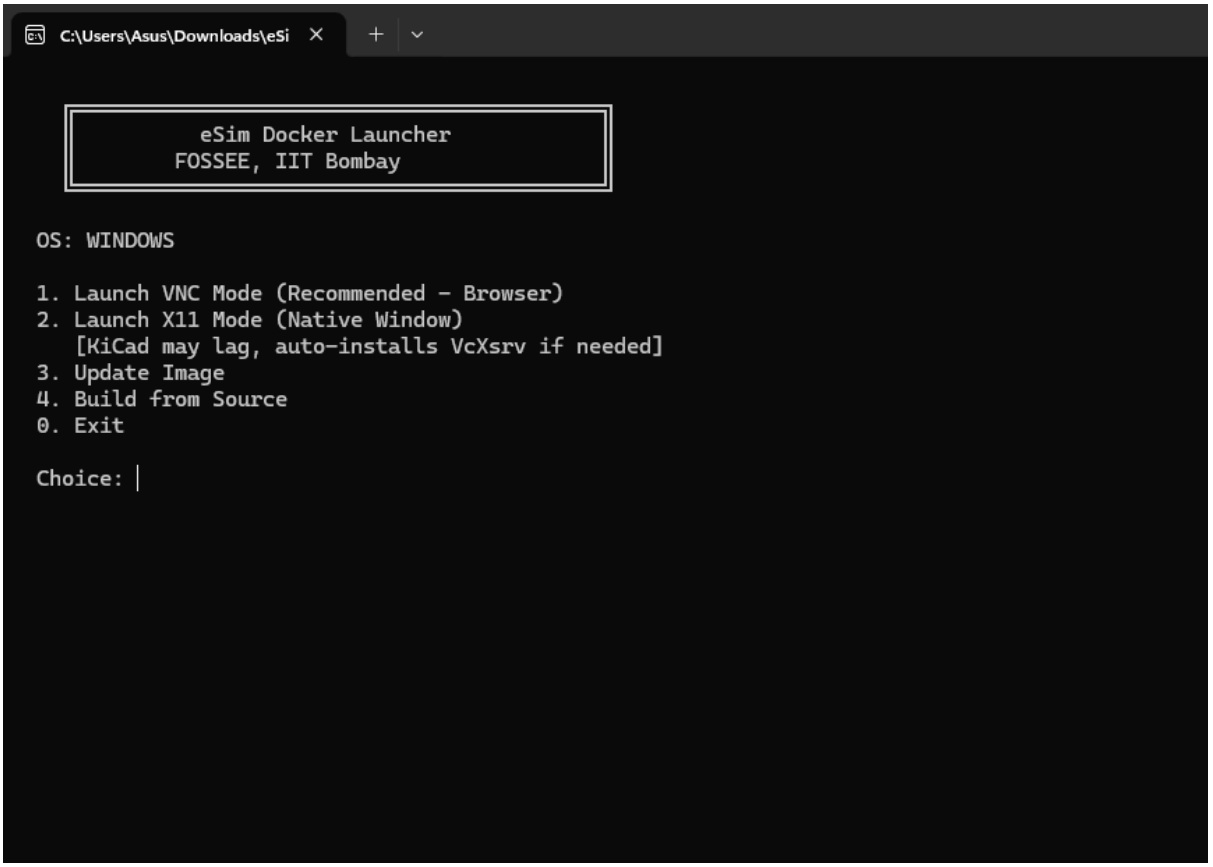
```
def find_free_port(start=6080, tries=20):
    for port in range(start, start + tries):
        try:
            with socket.socket() as s:
                s.bind(('', port))
                return port
        except OSError:
            continue
```

```
    raise RuntimeError("No free port found")
```

5. VNC and X11 Launch Functions

Separate functions for each display mode:

```
def launch_vnc(image, workspace):
    vnc_port = find_free_port(6080)
    cmd = ["docker", "run", "-p", f"{vnc_port}:6080", ...]
    # Start container and open browser
def launch_x11(image, workspace, os_type):
    if os_type == "windows":
        start_vcxsrv()
    # Configure DISPLAY and start container
```



[SCREENSHOT : Launcher menu showing options]

5.3 GitHub Actions CI/CD

The GitHub Actions workflow automates the entire build and release process:

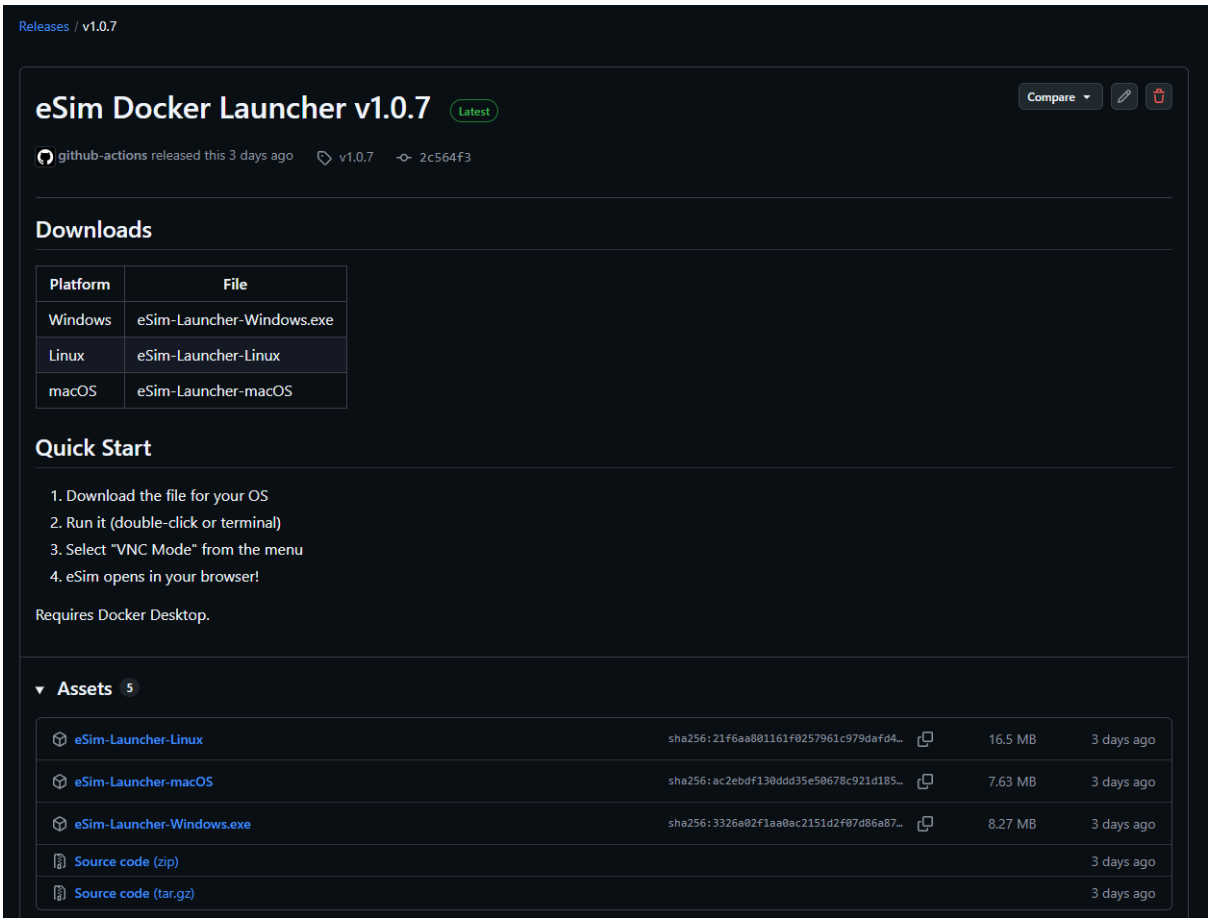
Workflow Triggers:

- Triggered on push to main branch (Docker build only)
- Triggered on version tag push (full release with executables)

Jobs:

1. **Docker Build Job** - Builds the Docker image and pushes to GHCR
2. **Executable Build Jobs** - Runs on Windows, Linux, and macOS runners to create platform-specific binaries using PyInstaller [6]
3. **Release Job** - Creates a GitHub Release and uploads all artifacts [7]

```yaml
name: Build and Release
on:
 push:
   branches: [main]
   tags: ['v*']
jobs:
 docker:
   runs-on: ubuntu-latest
   steps:
     - uses: actions/checkout@v4
     - name: Build and push Docker image
       run: |
         docker build -t ghcr.io/${{ github.repository }}:latest .
         docker push ghcr.io/${{ github.repository }}:latest
 build-windows:
   runs-on: windows-latest
   steps:
     - uses: actions/checkout@v4
     - name: Build executable
       run: pyinstaller --onefile run_esim_docker.py
```



[SCREENSHOT : GitHub Release page]

Chapter 6: Testing and Results

This chapter presents the verification of the tool across platforms and simulation of a sample circuit.

6.1 Testing Methodology

Testing was conducted on multiple platforms to ensure cross-platform compatibility:

Platform	Test Environment
Windows 10	Native installation
Windows 11	Native installation
Ubuntu 22.04	Native installation
macOS	Not directly tested (CI build verified)

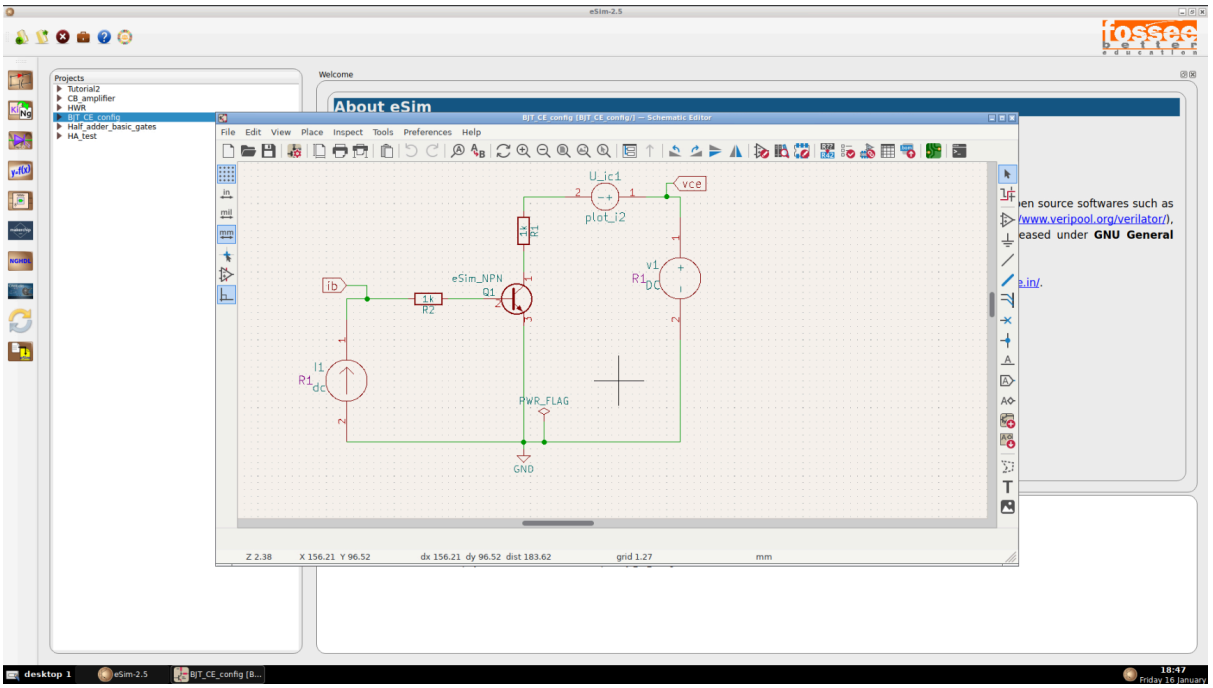
Test cases included:

1. Fresh installation (Docker not installed)
2. Docker installed but not running
3. VNC mode launch and basic eSim operations
4. X11 mode launch and basic eSim operations
5. Circuit simulation and waveform viewing
6. Workspace persistence across container restarts

6.2 Platform Testing Results

Windows Testing:

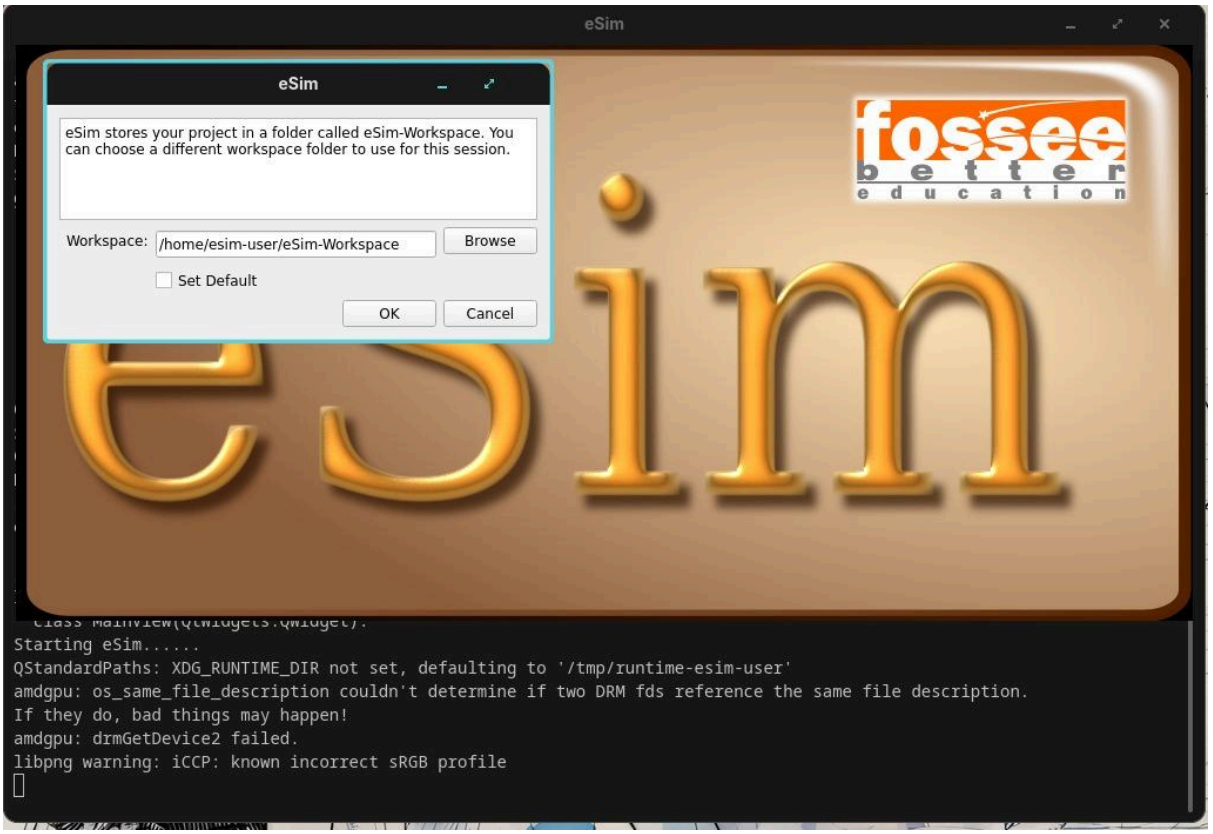
- VNC mode: Works correctly, browser opens automatically
- X11 mode: Works with VcXsrv, some slight lag in KiCad

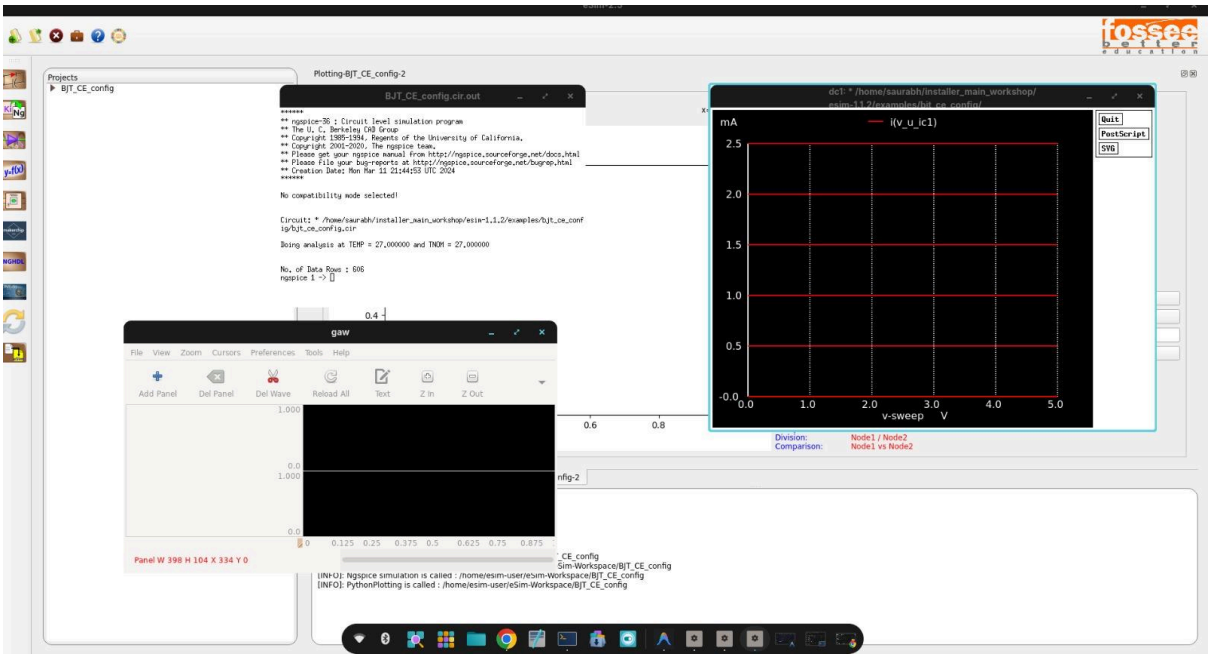


[SCREENSHOT : eSim running on Windows via VNC mode]

Linux Testing:

- X11 mode: Works natively with excellent performance
- VNC mode: Works correctly as fallback option



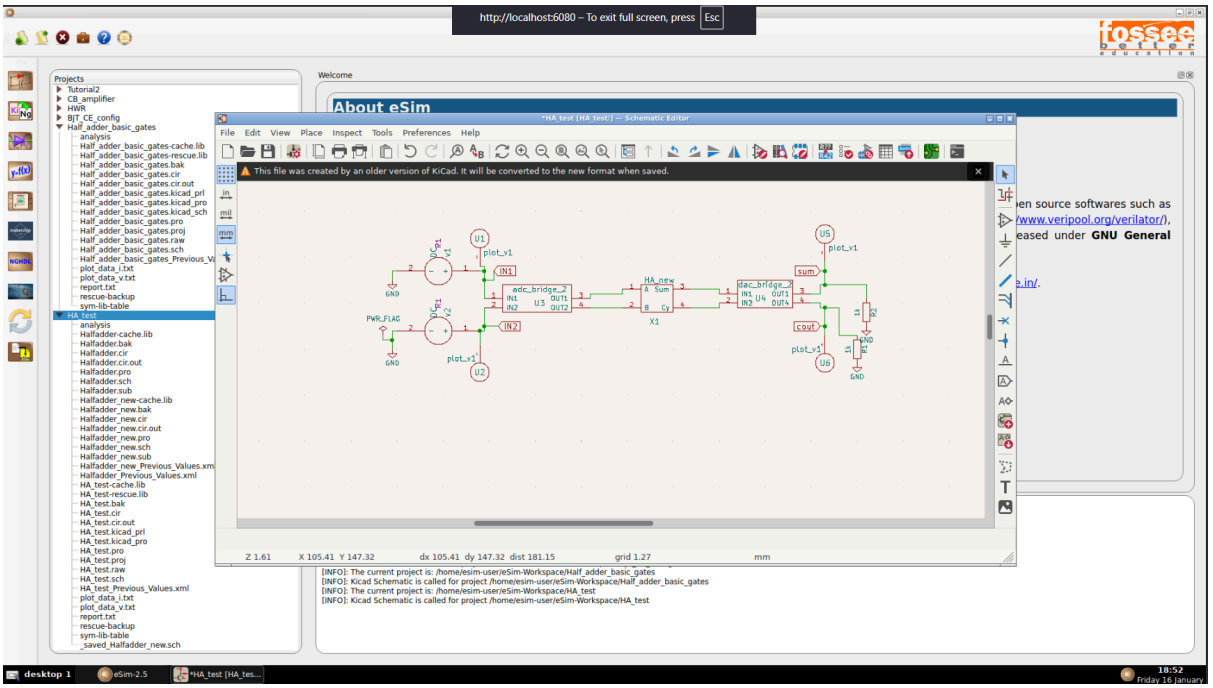


[SCREENSHOT : eSim running on Linux via X11 mode]

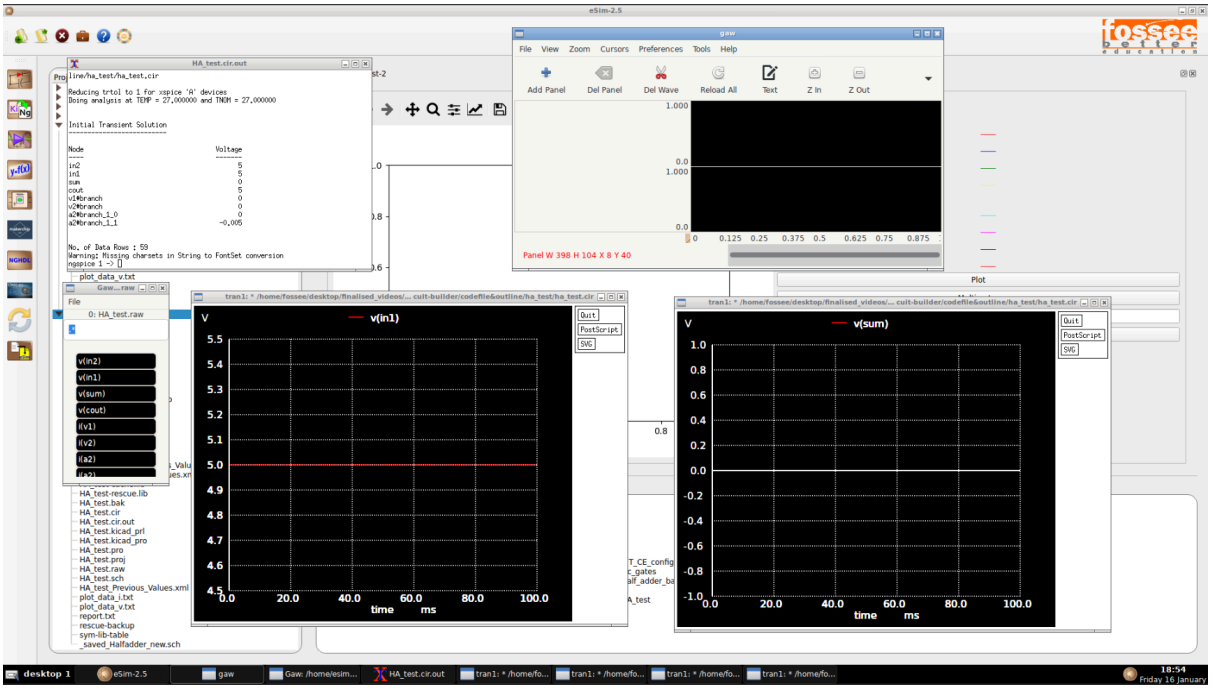
Simulation Test:

A simple half-wave rectifier circuit was created and simulated to verify complete functionality:

1. Created schematic in KiCad (through eSim)
2. Generated netlist
3. Ran Ngspice simulation
4. Viewed waveform in GAW3



[SCREENSHOT : Half-Adder schematic]



[SCREENSHOT PLACEHOLDER: Simulation output waveform in GAW3]

Chapter 7: Challenges and Solutions

This chapter documents the technical hurdles encountered and the engineering decisions made to overcome them.

Challenge 1: Port 6080 Already in Use

Problem: When launching VNC mode multiple times or with leftover containers from previous sessions, the noVNC port 6080 would sometimes be occupied, causing the container to fail.

Solution: Implemented dynamic port allocation. The launcher scans for available ports starting from 6080 and automatically uses the next available port if the default is busy.

Challenge 2: Large Docker Image Size

Problem: The initial Docker image exceeded 5 GB due to inclusion of build dependencies and intermediate compilation files.

Solution: Adopted multi-stage Docker build. The first stage compiles GAW3 and sets up dependencies. The second stage starts fresh and copies only the final binaries, resulting in a significantly smaller image.

Challenge 3: Workspace Dialog Hidden Behind Splash Screen

Problem: In VNC mode, the eSim workspace selection dialog appeared behind the splash screen, requiring users to manually move windows.

Solution: Switched from XFCE to openbox window manager for VNC mode. Openbox handles window focus more predictably. Added tint2 panel to provide a taskbar for window management.

Challenge 4: Browser Opening Too Early

Problem: The browser would open before the noVNC server was fully ready, showing connection errors.

Solution: Replaced simple TCP port check with HTTP health check. The launcher now waits until noVNC returns a valid HTTP response before opening the browser.

Challenge 5: X11 on Windows

Problem: Windows lacks native X11 support, preventing direct display of Linux GUI applications.

Solution: Integrated VcXsrv X server. The launcher auto-detects, installs (via winget if needed), and configures VcXsrv with appropriate settings for seamless X11 forwarding.

Chapter 8: Conclusion and Future Scope

The following enhancements could be implemented in future versions:

1. **AppImage for Linux** - Convert the Linux binary to AppImage format for better distribution. AppImages are self-contained and work across different Linux distributions without requiring users to set executable permissions manually.
2. **GUI Launcher** - Replace the current CLI launcher with a graphical interface using PyQt5 or tkinter. This would make the tool more accessible to users who are not comfortable with command-line interfaces.
3. **Automatic Updates** - Implement a mechanism to check for and download new versions of both the launcher and Docker image automatically.
4. **Resolution Selection** - Allow users to select their preferred VNC resolution from the launcher menu instead of using the default 1920x1080.
5. **Persistent Settings** - Save user preferences (display mode, resolution, workspace location) to a configuration file so they persist across sessions.
6. **macOS Native Support** - Improve and test X11 support on macOS with XQuartz integration.

Conclusion

This internship project successfully achieved its objective of creating a cross-platform deployment solution for eSim. The Docker-based approach eliminates the complexity of native installation and ensures consistent behaviour across Windows, Linux, and macOS.

- Key accomplishments include:
- A fully functional Docker image containing eSim and all dependencies
- A user-friendly launcher with support for VNC and X11 display modes
- Automated CI/CD pipeline for continuous builds and releases
- Solutions for various technical challenges including port conflicts, image size, and display issues

The solution is now publicly available on GitHub and can be used by anyone who wants to run eSim without going through the traditional installation process. This work contributes to making electronic circuit simulation more accessible to students and hobbyists.

Bibliography

1. eSim Official Website - https://esim.fossee.in/
2. FOSSEE Project - https://fossee.in/
3. Docker Documentation - https://docs.docker.com/
4. TigerVNC Documentation - https://tigervnc.org/
5. noVNC Project - https://novnc.com/
6. PyInstaller Documentation - https://pyinstaller.org/
7. GitHub Actions Documentation - https://docs.github.com/en/actions
8. KiCad Documentation - https://www.kicad.org/
9. Ngspice Manual - https://ngspice.sourceforge.io/

Appendix

Appendix A: GitHub Repository

The complete source code is available at:
https://github.com/Barun-2005/eSim-docker

Appendix B: Quick Start Guide

For Windows Users:

1. Install Docker Desktop from https://www.docker.com/products/docker-desktop
2. Download eSim-Launcher-Windows.exe from the Releases page
3. Double-click the launcher and select VNC mode
4. eSim opens in your browser

For Linux Users:

1. Install Docker using your distribution's package manager
2. Download eSim-Launcher-Linux from the Releases page
3. Run: chmod +x eSim-Launcher-Linux && ./eSim-Launcher-Linux
4. Select X11 mode for best performance