Semester Long Internship Spring 2026

On

Automated eSim Tool Manager

Submitted by

Moreshwar Balasaheb Naikwadi
Final Year Student,
B. Tech, Computer Science And Engineering,
Specializing in Cyber Security And Forensics,
MIT Art Design And Technology University.

Under the guidance of

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

June 6, 2026

Acknowledgment

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

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

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

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

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

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

Contents

Chapter 1

Introduction

1.1 Introduction to eSim

eSim is an open-source Electronic Design Automation (EDA) software developed under the FOSSEE initiative at Indian Institute of Technology Bombay. It is designed for circuit simulation, schematic design, PCB design, and electronic system analysis. eSim integrates multiple open-source tools into a unified environment, allowing users to perform analog and digital circuit simulations efficiently.

The software combines schematic capture, PCB designing, simulation, and waveform analysis into a single platform. eSim mainly integrates tools such as Ngspice for analog circuit simulation, KiCad for PCB designing, GHDL and Verilator for digital simulations, and other supporting libraries and packages. The purpose of eSim is to provide students, researchers, hobbyists, and professionals with a completely open-source alternative to expensive commercial EDA software.

Circuit simulation is one of the most important features of eSim. It allows users to test and analyze electronic circuits virtually before physically implementing them. Through simulation, engineers and students can identify errors, optimize designs, analyze performance, and reduce hardware development costs. eSim supports both analog and digital simulations, making it suitable for a wide range of electronics and embedded system applications.

PCB (Printed Circuit Board) design is another essential functionality provided through the integration of KiCad. Users can design professional PCB layouts, create footprints, route connections, and generate manufacturing files directly from the eSim environment. This integration creates a seamless workflow between simulation and hardware implementation.

As an open-source EDA platform, eSim encourages collaborative development, community contributions, educational accessibility, and customization. Unlike proprietary software, eSim can be freely modified and extended according to user requirements. The open-source ecosystem also helps educational institutions provide affordable access to advanced EDA tools.

1.2 Overview of eSim

eSim is an open-source Electronic Design Automation (EDA) software developed under the FOSSEE initiative at Indian Institute of Technology Bombay. It is designed to provide an integrated environment for circuit simulation, schematic design, PCB design, and electronic system analysis. eSim combines multiple open-source tools into a unified platform, enabling students, researchers, educators, and professionals to perform analog and digital simulations efficiently without relying on expensive proprietary software.

The eSim ecosystem integrates several important software tools such as Ngspice, KiCad, GHDL, Verilator, LLVM, and Python-based utilities. Ngspice is used for analog circuit simulation, KiCad is used for PCB designing and schematic capture, while GHDL and Verilator support digital hardware simulations. Together, these tools create a complete simulation and hardware design environment.

One of the major advantages of eSim is its open-source nature. Since the software is freely available, educational institutions and students can access professional-grade EDA tools without financial limitations. The platform encourages collaborative development, community contributions, and continuous improvement through open-source participation.

The eSim Tool Manager plays a critical role within the eSim ecosystem. It automates the installation, updating, configuration, and management of various tools required by eSim. Without the Tool Manager, users would have to manually download, install, configure, and maintain multiple software packages individually, which is often complex and error-prone.

The project work carried out during this internship/fellowship focused on enhancing both the Standalone System and Integrated System of the eSim Tool Manager. The project involved automation improvements, bug fixing, GUI enhancements, dependency management, update management, version verification, multi threading implementation, progress tracking systems, logging systems, and overall stabilization of the project across Windows and Ubuntu platforms.

1.3 Objectives of the Project

The primary objective of the project was to improve and enhance the eSim Tool Manager system to provide a more reliable, automated, stable, and user-friendly experience for users working with eSim and its dependencies.

The project mainly focused on both Standalone and Integrated Tool Manager systems across Windows and Ubuntu operating systems. The major objective was to simplify the management of external tools required by eSim, including installation, updating, dependency management, configuration handling, and package verification. title The core objectives of the project are listed below:

Automation of Tool Installation

One of the primary objectives was to automate the installation process of software tools such as Ngspice, KiCad, GHDL, Verilator, LLVM, Chocolatey, and required Python packages. The system aimed to eliminate the need for manual installations and complex configuration steps.

Dependency Management

The project aimed to improve dependency handling by automatically checking, downloading, verifying, and configuring dependencies required by various software packages.

GUI Enhancement

Another important objective was to improve the graphical user interface of the Tool Manager to provide better usability, clearer feedback mechanisms, organized layouts, progress indicators, and improved interaction for users.

Progress Tracking and Status Indicators

The project focused on implementing real-time progress bars and download status indicators for installations and updates of different software packages. This was done to provide transparency and improve user experience.

Logging and Debugging Support

The implementation of detailed logs and activity viewers was another major objective. The logs system helps users and developers analyze execution details, identify errors, and troubleshoot issues efficiently.

Auto Version Checking

The Tool Manager was enhanced to automatically check installed versions and compare them with the latest available versions. This feature simplified update management and improved software maintenance.

Cross-Platform Support

The project aimed to maintain proper functionality across both Windows and Ubuntu operating systems by handling platform-specific commands, paths, dependencies, and installation methods.

Multithreading and UI Responsiveness

One of the major technical objectives was to solve GUI freezing issues during lengthy installations and downloads. Multithreading was implemented to execute heavy operations in the background while keeping the GUI responsive.

Project Stabilization

The project also focused on stabilizing the standalone and integrated systems by identifying and fixing bugs related to path handling, package detection, dependency

verification, execution flow, and update systems.

EXE Generation and Distribution

The standalone Windows system was packaged into executable (.exe) files for easier distribution and usability.

GitHub Contribution and Open-Source Development

Another objective was to contribute improvements to the official eSim Tool Manager repository through pull requests, code enhancements, debugging, and collaborative development workflows.

1.4 Methodology

The methodology followed during the development and enhancement of the eSim Tool Manager project involved research, analysis, implementation, debugging, testing, stabilization, and documentation phases. The project was executed systematically to ensure proper understanding of the existing architecture while introducing new improvements without affecting existing functionalities.

Initial Research and Exploration

The project began with understanding the eSim ecosystem, FOSSEE initiative, and the architecture of the Tool Manager system. Research was conducted on how eSim integrates external tools such as Ngspice, KiCad, GHDL, Verilator, and LLVM. Existing documentation, GitHub repositories, previous fellowship reports, and related workflows were studied thoroughly.

The file structures, branches, plugin systems, and repository organization of the eSim Tool Manager were explored in detail. Meetings were conducted with mentors and team members to understand the scope of the project and expected deliverables.

Understanding Existing Architectures

The architecture and workflows of both the Standalone System and Integrated System were analyzed deeply. The standalone system operates independently for package installation and management, while the integrated system works within the eSim ecosystem and communicates directly with eSim components.

The installation workflows, updater mechanisms, dependency checking systems, package managers, GUI structure, JSON-based version tracking, and configuration systems were studied carefully.

Environment Setup

Development environments were configured on both Windows and Ubuntu systems. Necessary dependencies, Python packages, package managers, and supporting tools were installed and tested. Ubuntu virtual environments and standalone execution systems were also explored to understand Linux-specific workflows.

Bug Identification and Analysis

The next phase involved identifying limitations and bugs in the existing systems. Issues related to path handling, dependency management, GUI responsiveness, package detection, installation workflows, and software execution were analyzed and documented.

The integrated system and standalone system were both tested extensively to identify stability issues and user experience problems.

Feature Development and Enhancement

After understanding the existing codebase and identifying limitations, new features were implemented to improve the functionality and usability of the Tool Manager system.

Major features developed during the project include:

Progress bars for software installation and updates Download status bars for Ngspice, KiCad, GHDL, LLVM, and Chocolatey GUI redesign and enhancements View Logs feature Dependency Viewer feature Automatic version checking system Improved package management section Multithreading implementation Enhanced execution system Better dependency verification Improved package update work-

flows Auto version validation Logs architecture and debugging support Multithreading Implementation

One of the most critical improvements was implementing multithreading to prevent GUI freezing during lengthy operations. Background worker threads were introduced to handle installations, downloads, and updates while maintaining GUI responsiveness.

Testing and Validation

The system was tested extensively across Windows and Ubuntu platforms. Functional testing, GUI testing, dependency verification, installation testing, update testing, and error handling validation were performed to ensure stability and reliability.

Different software tools such as KiCad, Ngspice, GHDL, Verilator, LLVM, Chocolatey, and Python packages were tested repeatedly under multiple scenarios.

Git and GitHub workflows were also explored during workshops and collaborative development sessions.

Documentation and Reporting

Detailed project reports, logs, screenshots, architecture diagrams, workflow diagrams, and implementation documentation were prepared throughout the project lifecycle. Final reports and executable builds of the standalone system were generated before submission.

Chapter 2

Literature Survey

The literature survey phase involved studying existing reports, research materials, documentation, and workflows related to the eSim Tool Manager project. Previous semester-long internship and fellowship reports were analyzed to understand the architecture, implementation methods, workflows, limitations, and future scope of the existing systems.

Research was conducted on open-source EDA tools, package management systems, dependency management mechanisms, GUI frameworks, subprocess execution systems, and multithreading techniques. Technologies such as Python, Chocolatey, Ubuntu package management systems, JSON version handling, logging systems, and concurrent execution models were studied extensively.

The literature survey also involved understanding how modern software installers and update managers function in professional environments. Various package managers and updater systems used in Windows and Linux ecosystems were analyzed to identify best practices and possible improvements for the eSim Tool Manager.

The previous reports highlighted several limitations in the older system, including lack of proper progress tracking, GUI freezing issues, insufficient logging systems, dependency conflicts, inconsistent version management, and platform-specific issues. These studies helped in identifying areas where improvements and optimizations were required.

The GitHub repository structure, code organization, plugin systems, updater workflows, dependency verification systems, and package installation flows were studied thoroughly during the literature survey phase. Understanding the existing implementation helped in planning enhancements while maintaining compatibility with the existing architecture.

Chapter 3

Problem Statement

3.1 Problem Statement

Before the development and enhancement of the eSim Tool Manager, users faced multiple technical and usability challenges while installing and managing external tools required by eSim.

The installation process required users to manually download, install, configure, and maintain several software packages individually. Different tools required different dependencies, environment variables, versions, and configurations. This made the setup process complicated, especially for beginners and students.

Dependency conflicts were one of the major issues. Missing packages, incompatible versions, and incorrect configurations often caused installation failures and runtime errors.

Manual path configuration was another major problem. Users had to manually configure environment variables and software paths after installation. Incorrect configurations frequently caused eSim to fail in detecting installed tools properly.

Version mismatch problems also created instability because some versions of tools were incompatible with specific versions of eSim.

Cross-platform inconsistency between Windows and Ubuntu systems increased development and maintenance complexity. Different commands, installation workflows, and package management systems had to be handled separately.

The absence of centralized management created confusion for users because there was no unified interface for managing downloads, installations, updates, logs, and dependencies.

The system also lacked proper progress tracking and real-time feedback during lengthy operations, resulting in poor user experience.

3.2 Approach

To solve the identified problems, a systematic approach was followed to improve the Tool Manager system.

The first step involved analyzing the existing architecture and identifying limitations in both standalone and integrated systems. After understanding the workflows, improvements were planned for installation systems, update systems, dependency management, GUI design, logging mechanisms, and execution handling.

The approach mainly involved:

Automating installation and update workflows Implementing progress bars and status indicators Introducing detailed logging systems Adding dependency viewer functionality Implementing automatic version checking Enhancing GUI responsiveness Implementing multithreading for background execution Fixing path-related issues Improving package detection systems Stabilizing execution workflows Enhancing user interaction and feedback systems

The approach also focused on maintaining compatibility across Windows and Ubuntu platforms while ensuring proper integration with eSim.

Chapter 4

Implementation

4.1 Case 1 The Windows Standalone System

The Standalone System Development phase focused on improving and stabilizing the Windows standalone version of the eSim Tool Manager. The standalone system is designed to operate independently without requiring deep integration into the complete eSim ecosystem. The primary purpose of the standalone system is to automate the installation, updating, management, and verification of external software tools required by eSim such as Ngspice, KiCad, LLVM, GHDL, Verilator, Chocolatey, and various Python packages.

The work on the standalone system began with detailed exploration and understanding of the existing architecture. The older standalone system had several limitations related to dependency handling, GUI responsiveness, package verification, path management, update workflows, and user interaction. Understanding the existing codebase was one of the biggest challenges because the system was already partially developed and modifications had to be implemented carefully without breaking existing functionalities.

The first stage of development involved exploring the repository structure, execution flow, package manager modules, updater systems, dependency verification systems, configuration files, JSON-based version handling systems, and GUI workflows. Previous internship reports and existing documentation were studied thoroughly to understand the architecture and functionality of the system.

After understanding the existing structure, multiple bugs and issues were identified in the standalone system. Path-related problems were among the major issues. Installed software paths were not being detected properly in some scenarios, causing failures in package verification and software execution. Various path resolution mechanisms were analyzed and improved to ensure correct handling of software installation directories and environment variables.

Dependency management was another critical area of improvement. Different software tools required different dependencies and libraries. In many cases, missing dependencies caused installation failures and execution errors. The dependency verification mechanism was enhanced to automatically identify missing packages and provide better error handling and user feedback.

One of the most significant improvements made to the standalone system was

the implementation of real-time progress bars and status indicators. Earlier, users had no clear visibility regarding installation progress during long-running downloads and installations. To solve this issue, dynamic progress bars were implemented for various tools including Ngspice, KiCad, LLVM, GHDL, and Chocolatey. These progress indicators provided users with visual feedback regarding current download and installation progress.

Download status bars were also implemented for all major packages. These download indicators continuously monitored installation status and displayed real-time updates to users. This significantly improved user interaction and reduced confusion during lengthy operations.

The graphical user interface (GUI) of the standalone system was redesigned and enhanced extensively. Earlier versions of the GUI lacked proper organization, visual feedback mechanisms, and usability improvements. New GUI sections were introduced for package downloads, dependency management, version checking, logs viewing, and update management. The updated interface became more structured, informative, and user-friendly.

Another major feature implemented during the project was the "View Logs" system. Logging functionality was introduced to store detailed execution information related to downloads, installations, updates, warnings, errors, and package management activities. A dedicated logs viewer GUI was added so users could easily access logs for debugging and troubleshooting purposes. This feature also helped developers analyze hidden runtime errors and improve system stability.

In addition to logs viewing, a "View Dependencies" feature was added to the standalone system. This functionality allowed users to view package dependencies and software requirements directly through the GUI. The dependency viewer improved transparency and helped users better understand how different tools interact with the system.

Automatic version checking functionality was another major enhancement implemented during development. The system was improved to automatically detect installed versions of software packages and compare them with the latest available versions. This eliminated the need for manual version verification and simplified update management for users.

One of the most technically challenging tasks during development was solving GUI freezing issues. Initially, the GUI became unresponsive during lengthy operations such as downloads, updates, and installations because all tasks were executed on the main thread. To solve this issue, multithreading was implemented using Python threading mechanisms.

Background worker threads were introduced to handle heavy operations independently while keeping the GUI responsive. This significantly improved system responsiveness and overall user experience. Implementing multithreading required careful synchronization and execution management to avoid race conditions and unexpected behavior.

The execution system of the standalone manager was also stabilized and optimized. Subprocess handling, command execution workflows, error detection systems, and installation monitoring mechanisms were improved extensively. This helped reduce runtime crashes and increased system reliability.

The project also involved improving package update workflows. Dedicated update bars and status indicators were added for various software packages such as KiCad, Ngspice, LLVM, GHDL, and Python packages. These update indicators provided users with better visibility during software updates.

Another important contribution was fixing bugs related to software package detection and dependency verification. Multiple issues related to package existence checking, version validation, incorrect dependency handling, and execution paths were resolved during the stabilization phase.

The package management section of the standalone GUI was redesigned to improve organization and usability. The download packages section was updated with better layouts, progress tracking systems, and clearer status messages.

The project also involved generating executable (.exe) builds for the Windows standalone system. Packaging the standalone system into executable files improved usability and allowed easier distribution of the Tool Manager to users.

Extensive testing and validation were performed throughout the development process. Different installation scenarios, dependency conflicts, package updates, download failures, version mismatches, and GUI interactions were tested repeatedly to ensure system stability.

The standalone system ultimately became significantly more stable, interactive, automated, and user-friendly after these improvements. The implementation of progress bars, logs systems, dependency viewers, multithreading, automatic version checking, GUI enhancements, and execution optimizations transformed the standalone Tool Manager into a more reliable and professional software management system.

Project Overview

The Standalone System is a Windows-based package and dependency management solution developed for the eSim ecosystem. The primary objective of this project is to provide users with a graphical interface through which all required tools, packages, dependencies, and simulation environments can be installed, updated, monitored, and maintained automatically.

The system eliminates the need for users to manually install multiple software packages and configure environment variables. Instead, the Standalone System automates the entire setup process through a centralized GUI-based Tool Manager.

The system manages:

Chocolatey KiCad LLVM GHDL NGSpice Python Packages System Dependencies Version Tracking Installation Tracking Status Monitoring Logging Management

Problem Statement

Before the development of the Standalone System, users had to:

Manually Install Tools

Users needed to:

Download installers

Configure environment variables Install dependencies Verify installation paths Check

software versions Common Issues Faced Incorrect software versions Missing dependencies PATH configuration errors Installation failures Compatibility issues Lack of installation records Difficult troubleshooting

These issues increased setup time and reduced user productivity.

Project Objectives

The objectives of the Standalone System were:

Automation

Automate installation and management of all required tools.

User-Friendly Interface

Provide GUI-based controls for installation and updates.

Version Management

Track installed versions and latest available versions.

Status Monitoring

Show:

Installed/Not Installed Current Version Latest Version Update Availability Logging

Maintain detailed logs for troubleshooting.

JSON-Based Tracking

Store all installation details inside:

install details.json

Project Architecture

The system is divided into multiple modules.

Module 1: Chocolatey Manager

File:

chocolatey.py

Purpose:

Install Chocolatey Update Chocolatey Detect installation status Retrieve installed version Update JSON records Module 2: Dependency Manager

File:

dependencies.py

Purpose:

Install system dependencies Read dependencies from JSON Install packages using Chocolatey Update installation records Module 3: KiCad Package Manager

File:

kicad package manager.py

Purpose:

Install KiCad Update KiCad Fetch latest versions Display installed version Compare versions Store installation details Module 4: GHDL Package Manager

File:

ghdl package manager.py

Purpose:

Fetch latest release from GitHub Download binaries Extract packages Configure PATH Update installation records Module 5: LLVM Package Manager

Purpose:

Install LLVM Detect installed version Check update status Compare installed and latest versions Display package status Module 6: JSON Installation Database

File:

install details.json

Purpose:

Central storage for:

Package names Installed versions Installation dates Installation paths Dependency lists Python packages

My Role and Responsibilities

During this project, I was responsible for:

System Analysis Understanding installation flow Identifying missing functionalities Studying package management systems Development Modifying Python scripts Implementing installation logic Creating update mechanisms GUI Enhancements Improving package manager interfaces Adding status displays Adding version information Testing Installation testing Update testing Failure testing Version verification Debugging Resolving installation issues Fixing version detection errors Correcting JSON updates

Major Work Performed

A. Installation Status Management Previous Problem

The system only installed packages.

Users could not determine:

Whether a package was installed Which version was installed Installation location Solution

Implemented installation status tracking using:

"package name": "", "version": "", "installed": "", "installed date": "", "install directory": "" Result

Users can now see:

Installation status Installation date Installed version Installation path B. Version Tracking System Problem

No mechanism existed to determine:

Current version Latest version Update availability Solution

Added version detection mechanisms.

Examples:

Chocolatey choco -v KiCad

Chocolatey package version lookup.

GHDL

GitHub API version retrieval.

Result

The GUI now displays:

Installed Version Latest Version Update Available C. JSON Database Enhancement Problem

Package information was not centralized.

Solution

Enhanced:

install details.json

to store complete installation history.

Benefits Persistent storage Easier tracking Faster status retrieval D. Dependency Installation System Problem

Dependencies had to be installed manually.

Solution

Created automated dependency installation.

Dependencies are loaded from:

"dependencies": ["make", "gnat", "clang"]

and installed automatically.

Benefits Reduced setup time Consistent environments Fewer installation errors E. GHDL Automation Problem

Users manually downloaded releases.

Solution

Implemented:

GitHub API integration Release detection Asset selection Download automation Extraction automation Benefits

Fully automated GHDL installation.

F. KiCad Package Automation Problem

Manual installation process.

Solution

Created:

Version selection dropdown Installation button Update button Version comparison system Benefits

Simplified package management.

G. Logging System Improvements Problem

Troubleshooting was difficult.

Solution

Implemented logging for:

Installations Updates Errors Warnings

Generated:

tool manager.log Benefits Easy debugging Error tracking Activity monitoring H. GUI Improvements Work Done

Added:

Status Labels Installed Version Labels Latest Version Labels Update Indicators Progress Monitoring Benefits

Improved user experience.

Challenges Faced

Challenge 1: Version Detection Failure Problem

Installed versions were not displayed correctly.

Cause

Version retrieval logic was inconsistent.

Solution

Implemented dedicated version extraction functions.

Result

Accurate version reporting.

Challenge 2: Installation Status Not Updating Problem
Packages showed incorrect status.
Cause
JSON records were not updated after installation.
Solution
Created automatic JSON update functions.
Result
Accurate installation records.
Challenge 3: Package Path Detection Problem
Installed software locations varied.
Solution
Created dynamic directory detection logic.
Result
Correct path tracking.
Challenge 4: Chocolatey Integration Problem
Chocolatey installation sometimes failed.
Solution
Added:
Error handling Status verification Directory checks Result
Reliable package management.
Challenge 5: GHDL GitHub Release Changes Problem
Release assets change frequently.
Solution
Implemented dynamic asset selection logic.
Result
Compatible installation process.
Challenge 6: GUI Synchronization Problem
GUI labels were not refreshing after installation.
Solution
Created status refresh functions.
Result
Real-time updates.
Challenge 7: Version Comparison Issues Problem
String-based comparison caused incorrect update messages.
Example:
9.10 ¡ 9.2 Solution
Implemented proper version comparison logic.
Result
Correct update notifications.
Recent Enhancements Implemented
During the latest development cycle, significant improvements were made:

Version Status Display
Added:
Installed Version Latest Version Outdated/Latest Status LLVM Enhancements Implemented:

Installed Version Detection Latest Version Detection Update Availability Check Log Viewer Feature

Added a button to:

View Logs

which opens:

tool manager.log

for user troubleshooting.

Dependency Manager Enhancements

Planned and implemented support for:

Dependency status checking Log viewing Better diagnostics 10. Testing Activities Performed Functional Testing

Verified:

Installation Updates Status tracking GUI Testing

Verified:

Buttons Labels Refresh mechanisms JSON Testing

Verified:

Data writing Data reading Data consistency Error Testing

Verified:

Failed downloads Invalid versions Missing packages Integration Testing

Verified:

Chocolatey + KiCad Chocolatey + Dependencies GHDL + JSON LLVM + Status System

Outcomes Achieved

The Standalone System now provides:

One click installation

Automatic updates

Version tracking

Status monitoring

Dependency management

Package management

JSON-based records

Logging system

Error handling

GUI-based control

Reduced manual effort

Faster environment setup

Conclusion

The Standalone System significantly improves the deployment and maintenance of the eSim environment on Windows platforms. Through the implementation of automated package installation, dependency management, version tracking, installation monitoring, logging mechanisms, and GUI enhancements, the system has evolved into a reliable and user-friendly software management solution.

My contribution focused on improving automation, version management, installation tracking, GUI functionality, logging, dependency handling, troubleshooting support, and overall system reliability. Multiple technical challenges related to version detection, installation status updates, package integration, and GUI synchronization were successfully identified and resolved, resulting in a more stable, maintainable, and efficient standalone package management system for eSim users.

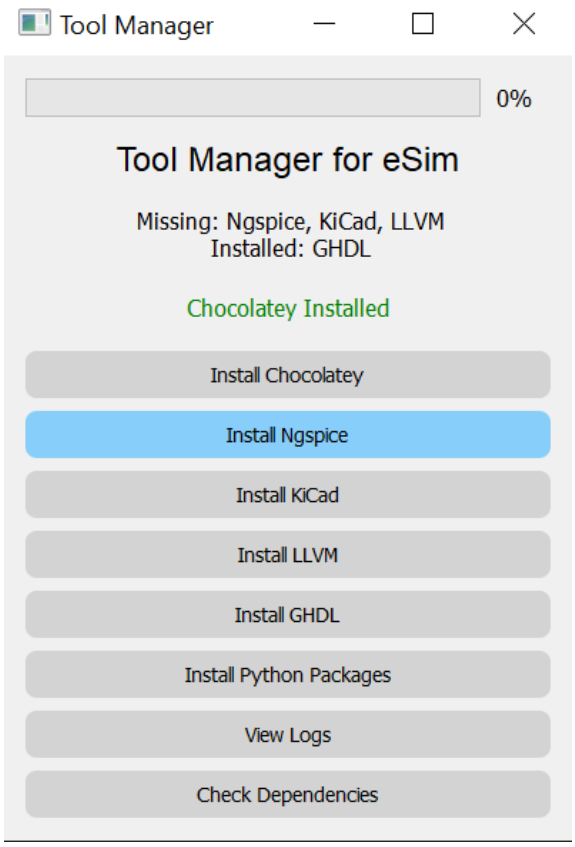

Figure 4.1

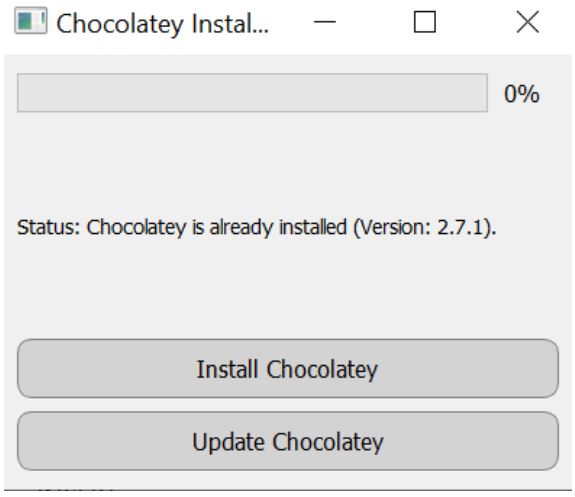

Figure 4.2

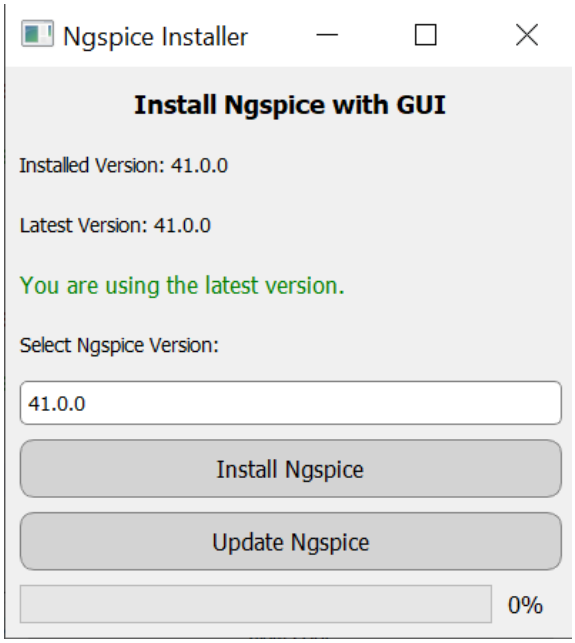

Figure 4.3

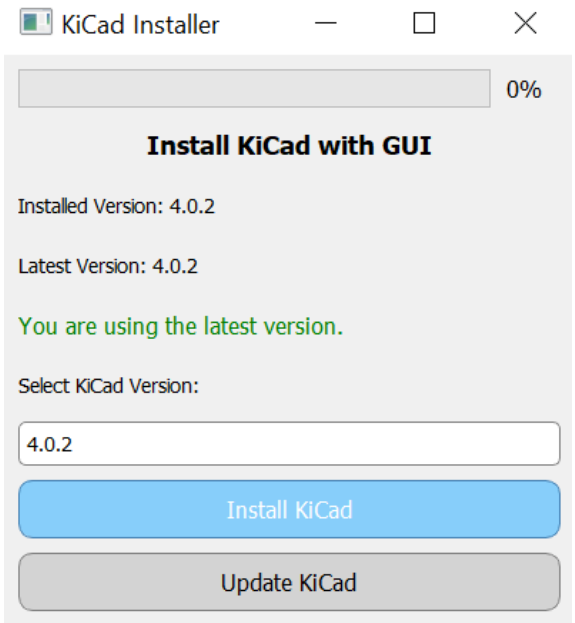

Figure 4.4

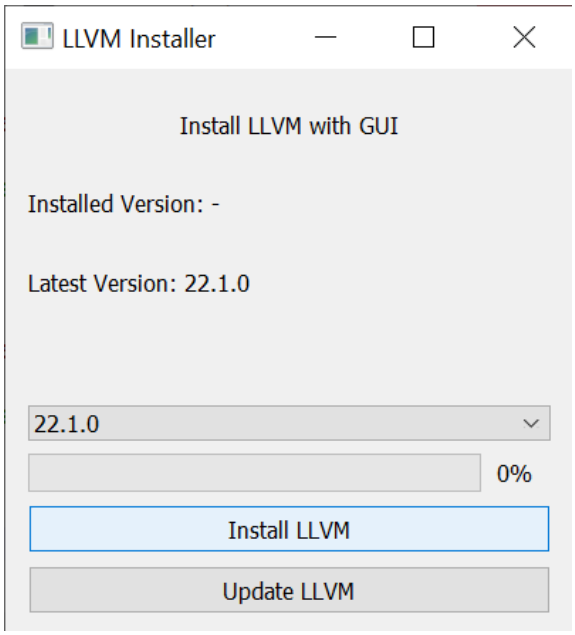

Figure 4.5

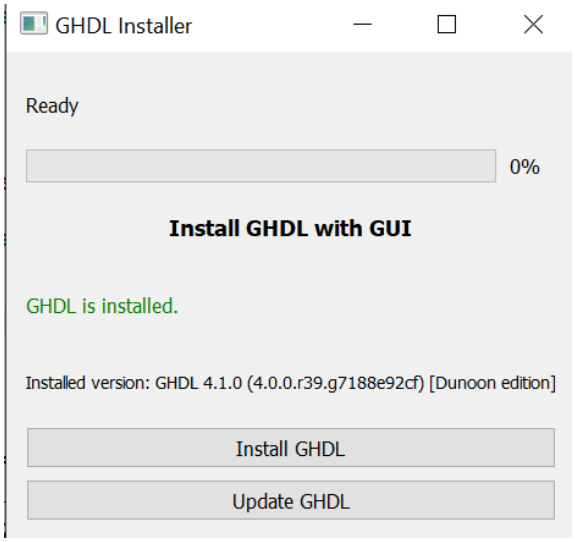

Figure 4.6

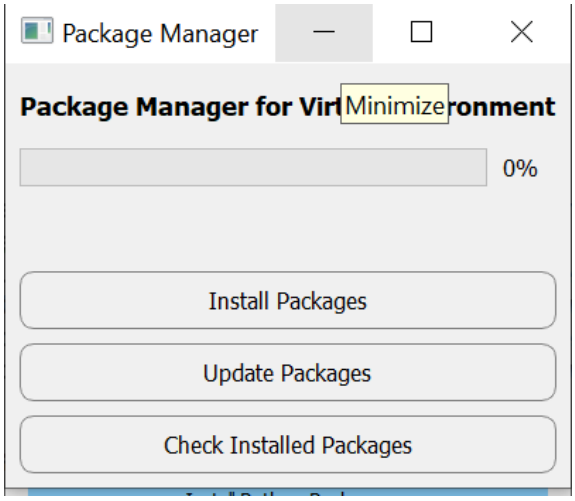

Figure 4.7

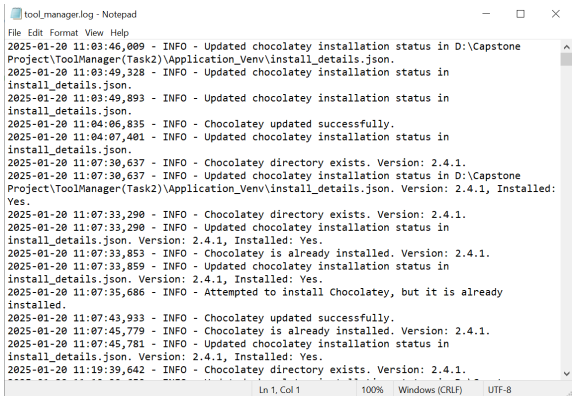

Figure 4.8

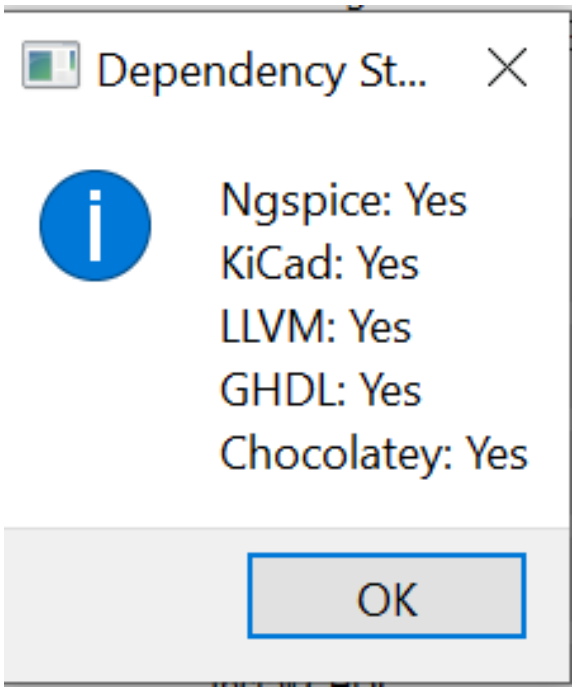

Figure 4.9

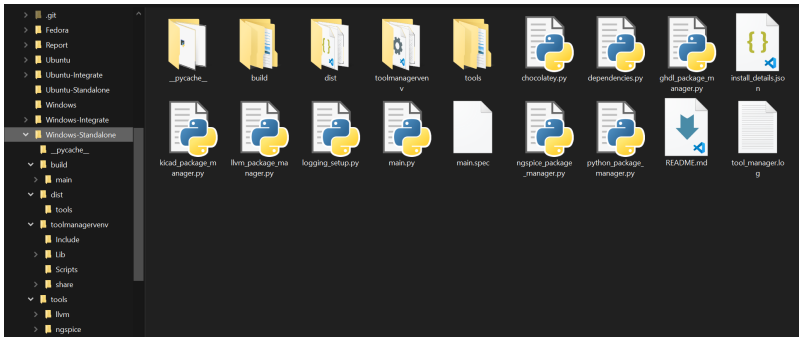

Figure 4.10

4.2 Case 2 The Windows Integrated System

The Integrated System Development phase focused on improving and stabilizing the integrated updater subsystem of the eSim Tool Manager for both Ubuntu and Windows platforms. Unlike the standalone system, the integrated system works directly within the eSim ecosystem and communicates closely with eSim components, package managers, configuration systems, and simulation workflows.

The integrated Tool Manager is responsible for managing installation, updating, dependency handling, package verification, configuration management, and version tracking of external tools such as KiCad, Ngspice, GHDL, Verilator, LLVM, and supporting packages within the eSim environment.

The work on the integrated system began with deep analysis of the existing updater architecture, package management systems, JSON-based version handling mechanisms, configuration systems, GUI structures, dependency verification workflows, and software installation pipelines. The repository structure and updater workflows were explored carefully to understand how the integrated system interacts with eSim and external software tools.

One of the major goals of the integrated system development was to improve software stability and resolve existing bugs affecting package management and updater functionality. Multiple issues related to package verification, dependency handling, updater execution, software configuration, and path management were identified and analyzed.

KiCad updater workflows were studied in detail to understand installation handling, version verification, configuration updates, library preservation, and compatibility management. Several bugs related to KiCad package detection, configuration handling, and updater execution were fixed during development.

Similarly, issues related to Ngspice integration and updater workflows were analyzed and resolved. Problems related to dependency verification, package installation, software detection, and update validation were fixed to improve reliability.

The Verilator updater subsystem also required multiple improvements. Various bugs affecting package updates, dependency management, and configuration workflows were corrected to stabilize the Verilator integration process.

The GHDL updater system was enhanced to improve version checking, dependency handling, update validation, and execution reliability. Configuration handling mechanisms were optimized to improve compatibility with the eSim environment.

Dependency management became one of the most important aspects of the integrated system development. Different packages required different supporting libraries, configuration files, and environment settings. The dependency verification system was improved to automatically detect missing dependencies and provide better feedback to users.

The integrated system also involved improvements in path handling and software detection mechanisms. Multiple path-related issues affecting package detection and execution were resolved to ensure proper communication between eSim and external tools.

Version management was another important area of improvement. The integrated system used JSON-based version tracking mechanisms to monitor installed

software versions and compare them with available updates. The version checking system was improved to provide more accurate package verification and update notifications.

Several GUI enhancements were implemented to improve usability and user interaction. The integrated updater interfaces were redesigned to provide better layouts, clearer status messages, organized update sections, and improved feedback mechanisms.

Download package sections and package management interfaces were also updated to provide smoother workflows and improved navigation. Better status messages and notifications were introduced to improve user understanding during installation and update operations.

The integrated system also included improved logging and debugging support. Logs were used to track package installations, update operations, dependency verification, configuration updates, and runtime errors. This improved transparency and helped developers analyze issues more effectively.

Testing and validation played a major role during the integrated system development process. Various installation scenarios, update workflows, dependency conflicts, version mismatches, missing package situations, and error conditions were tested repeatedly across both Ubuntu and Windows systems.

The updater systems for KiCad, Ngspice, Verilator, and GHDL were tested extensively to ensure proper functionality and compatibility with eSim. Package installation workflows, version verification systems, dependency checks, configuration updates, and error handling mechanisms were validated thoroughly.

Cross-platform compatibility was another major challenge during development. Windows and Ubuntu systems required different commands, package managers, installation methods, and environment configurations. Separate handling mechanisms were implemented to ensure consistent functionality across operating systems.

The integrated system ultimately became more stable, reliable, automated, and maintainable after these improvements. Enhanced dependency management, improved updater systems, better GUI interfaces, accurate version checking, improved path handling, and optimized package workflows significantly improved the overall functionality of the eSim Tool Manager within the eSim ecosystem.

The integrated system development also provided valuable practical experience in large-scale software maintenance, cross-platform architecture handling, package management systems, dependency resolution, debugging complex workflows, and collaborative open-source development.

Project Overview

The Integrated System is a centralized Windows-based Tool Management and Package Integration Platform developed for the eSim ecosystem. The primary objective of this project is to combine all required installation, update, verification, monitoring, dependency management, and troubleshooting functionalities into a single integrated application.

The system provides a unified graphical interface through which users can install, update, verify, manage, and monitor all eSim-related tools without manually downloading software, configuring paths, or managing dependencies.

The Integrated System extends the capabilities of the Standalone System by introducing centralized management, real-time status monitoring, advanced version verification, automated dependency resolution, re-download functionality, installation verification, logging, and integrated package coordination.

The system manages:

NGSpice GHDL KiCad Verilator Icarus Verilog Python Packages System Dependencies Download Management Version Tracking Installation Verification Status Monitoring Logging System Re-download Management Integrated Tool Coordination

Problem Statement

Before the development of the Integrated System, users encountered several difficulties while setting up and maintaining the eSim environment.

Manual Installation Process

Users had to:

Download installers individually Configure environment variables manually Verify installations manually Install dependencies separately Check compatibility manually Troubleshoot installation issues independently Common Issues Faced Missing dependencies Incorrect software versions Broken installations PATH configuration errors Download failures Version mismatches Difficulty identifying installed tools Lack of centralized management Reinstallation complexity Time-consuming setup process

These issues increased installation time, reduced productivity, and created challenges for new users.

Project Objectives

The objectives of the Integrated System were:

Centralized Tool Management

Provide a single platform for managing all eSim tools.

Automated Installation

Automate downloading, extraction, installation, and configuration.

Installation Verification

Verify that tools are installed correctly and operational.

Version Management

Track installed and latest available versions.

Dependency Resolution

Automatically identify and install required dependencies.

Real-Time Status Monitoring

Display:

Installed Status Installation Path Current Version Latest Version Update Availability Logging and Diagnostics

Maintain detailed logs for debugging and troubleshooting.

User-Friendly GUI

Provide an intuitive interface for tool management.

Project Architecture

The Integrated System is divided into multiple modules.

Module 1: Tool Detection Manager Purpose

Detect installed software automatically.

Functions Check executable availability Detect installation paths Verify tool accessibility Update tool status Supported Tools NGSpice GHDL Verilator Icarus Verilog KiCad Module 2: Version Management System Purpose

Retrieve and compare software versions.

Functions Installed version detection Latest version retrieval Version comparison Update notifications Module 3: Download Manager Purpose

Manage software downloads.

Functions Download packages Monitor download progress Verify downloads Redownload failed packages Module 4: Installation Manager Purpose

Manage software installation.

Functions Package extraction Installation execution Installation verification Postinstallation configuration Module 5: Dependency Manager Purpose

Handle dependency installation.

Functions Dependency detection Dependency installation Missing package identification Dependency verification Module 6: Logging System Purpose

Record system activities.

Functions Installation logs Update logs Error logs Diagnostic logs Module 7: GUI Management System Purpose

Provide user interaction.

Functions Tool status display Progress monitoring Version display Update indicators Error notifications

My Role and Responsibilities

During this project, I was responsible for:

System Analysis Understanding eSim installation workflows Studying package dependencies Identifying missing functionalities Analyzing existing limitations Software Development Modifying Python scripts Creating installation workflows Implementing verification mechanisms Integrating multiple tools GUI Development Improving user interface design Adding status indicators Implementing progress tracking Creating update notifications Testing Installation testing Update testing Dependency testing Failure testing Version verification testing Debugging Fixing installation errors Resolving version detection issues Correcting tool status problems Improving stability

Major Work Performed

A. Installation Verification System Previous Problem

The system could not confirm whether a tool was installed correctly.

Solution

Implemented executable-based verification.

Examples:

ngspice –version ghdl –version iverilog -V Result

Users can now verify successful installation instantly.

B. Tool Status Monitoring Problem

Users could not determine tool status.

Solution

Implemented real-time status monitoring.

Result

The GUI now displays:

Installed Not Installed Version Available Update Required C. Version Tracking Enhancement Problem

No reliable version tracking mechanism existed.

Solution

Implemented version extraction and comparison systems.

Result

The GUI displays:

Installed Version Latest Version Update Availability D. Download Progress Monitoring Problem

Users had no visibility into download progress.

Solution

Added dynamic progress bars.

Result

Users receive real-time download feedback.

E. Re-download Functionality Problem

Interrupted downloads required manual intervention.

Solution

Implemented Re-download button.

Benefits Recovery from failed downloads Reduced installation failures Improved reliability F. Logging System Enhancement Problem

Troubleshooting was difficult.

Solution

Implemented centralized logging.

Generated files:

logs.txt installation logs error logs Benefits Easy debugging Activity tracking Error identification G. Dependency Management Automation Problem

Dependencies required manual installation.

Solution

Created automatic dependency verification and installation.

Benefits Faster setup Reduced errors Consistent environment H. Integrated Tool Coordination Problem

Each tool previously operated independently.

Solution

Created centralized integration logic.

Benefits Unified workflow Better maintenance Simplified management

Challenges Faced

Challenge 1: Tool Detection Failure Problem

Installed tools were not always detected correctly.

Cause
Different installation directories and executable locations.
Solution
Implemented dynamic detection mechanisms.
Result
Accurate installation detection.
Challenge 2: Version Detection Inconsistencies Problem
Different tools return version information differently.
Solution
Created dedicated version parsing functions.
Result
Accurate version reporting.
Challenge 3: Download Interruptions Problem
Large packages occasionally failed during download.
Solution
Implemented download validation and re-download support.
Result
Reliable package installation.
Challenge 4: Installation Verification Problem
Tools appeared installed but failed to execute.
Solution
Added executable verification checks.
Result
Accurate installation validation.
Challenge 5: Windows Path Management Problem
Installation paths varied between systems.
Solution
Implemented dynamic path discovery.
Result
Correct path tracking.
Challenge 6: GUI Synchronization Problem
Status labels did not refresh immediately.
Solution
Created automatic refresh functions.
Result
Real-time status updates.
Challenge 7: Dependency Resolution Problem
Missing dependencies caused installation failures.
Solution
Added dependency checking before installation.
Result
More reliable installations.

Recent Enhancements Implemented

During the latest development cycle, several improvements were implemented.

Advanced Version Detection

Added:

Installed Version Detection Latest Version Detection Update Availability Status NGSpice Version Verification

Implemented:

Installed version retrieval Version comparison Installation validation Download Recovery System

Implemented:

Re-download functionality Download validation Error recovery Enhanced Logging

Added:

Detailed installation logs Error diagnostics Tool verification logs Improved Status Dashboard

Added:

Tool Status Indicators Installation Verification Status Update Notifications Better Error Handling

Implemented:

Exception handling User notifications Installation recovery mechanisms 10. Testing Activities Performed Functional Testing

Verified:

Tool installation Updates Version tracking Status monitoring GUI Testing

Verified:

Buttons Labels Progress bars Status indicators Installation Testing

Verified:

Fresh installations Reinstallations Failed installations Version Testing

Verified:

Installed versions Latest versions Update notifications Download Testing

Verified:

Download completion Interrupted downloads Re-download functionality Integration Testing

Verified:

NGSpice + Detection System GHDL + Download Manager Dependency Manager + Installation Manager Logging System + GUI Version Manager + Tool Detection

Outcomes Achieved

The Integrated System now provides:

Centralized Tool Management

One-Click Installation

Automated Updates

Installation Verification

Version Tracking

Status Monitoring

Download Progress Tracking

Re-download Support

Dependency Management

Logging System

Error Handling

Real-Time GUI Updates
Improved Reliability
Faster Environment Setup
Simplified User Experience

Conclusion

The Integrated System significantly improves the deployment, maintenance, and management of the eSim ecosystem on Windows platforms. By integrating installation management, version tracking, dependency resolution, download management, installation verification, logging, and status monitoring into a single application, the system provides a complete and reliable solution for eSim users.

My contribution focused on enhancing installation automation, version verification, dependency management, GUI functionality, download recovery mechanisms, tool integration, logging infrastructure, and system reliability. Multiple technical challenges related to tool detection, version management, installation verification, dependency handling, path management, and GUI synchronization were successfully resolved. As a result, the Integrated System has evolved into a scalable, maintainable, and efficient software management platform that greatly reduces manual effort and improves the overall user experience.

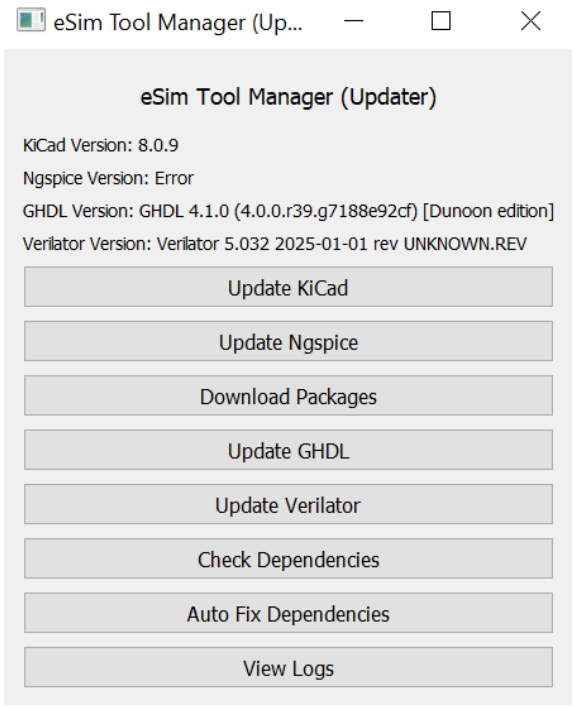

Figure 4.11

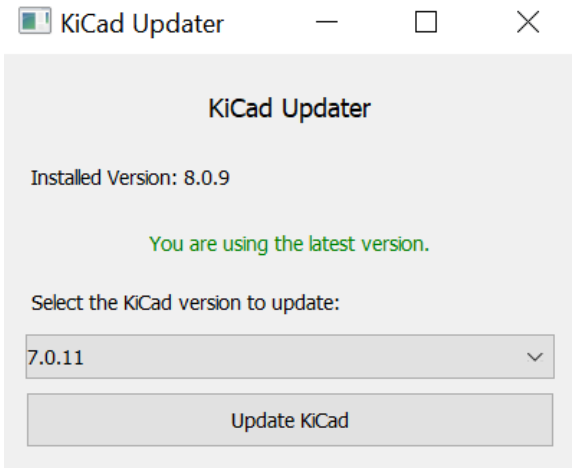

Figure 4.12

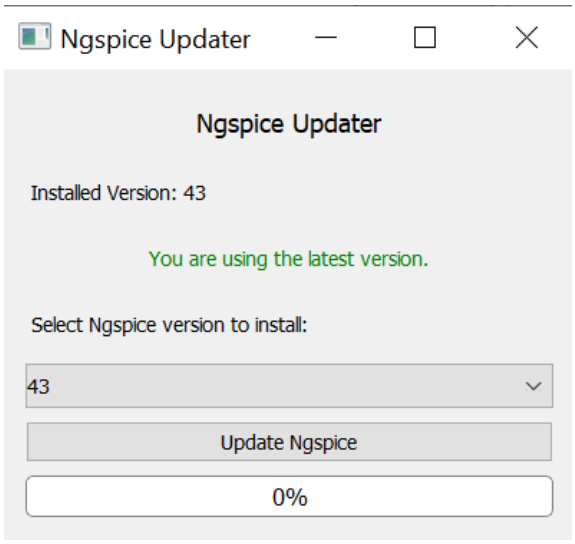

Figure 4.13

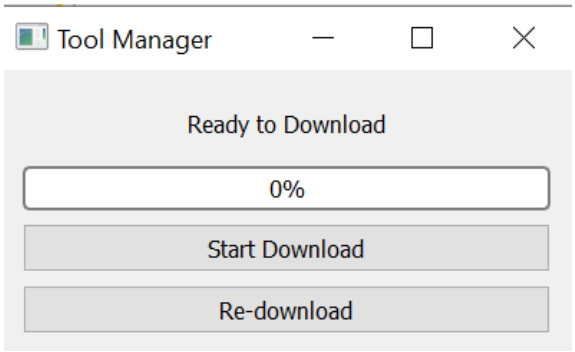

Figure 4.14

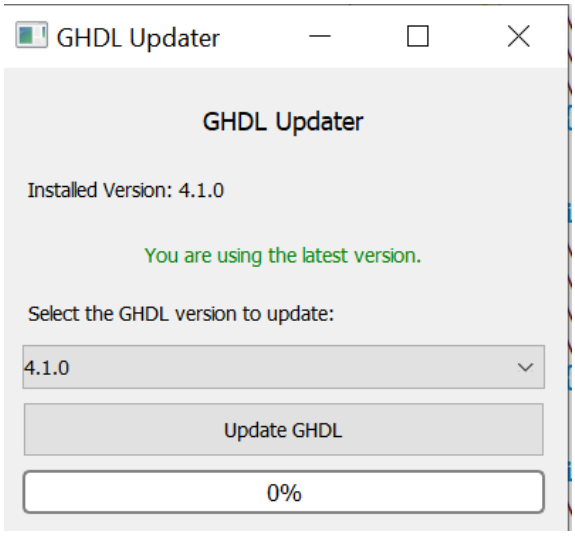

Figure 4.15

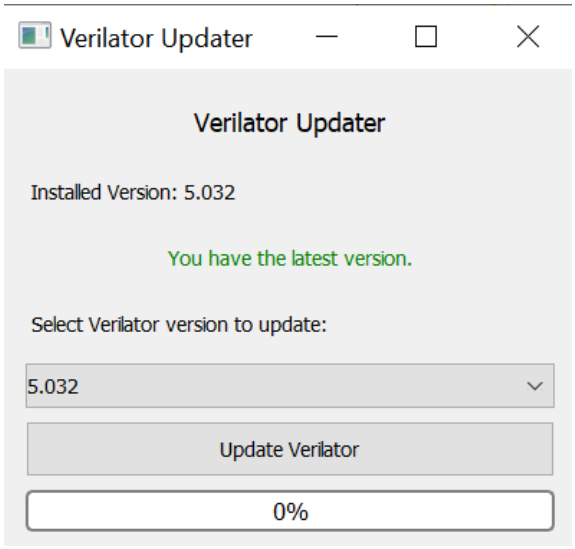

Figure 4.16

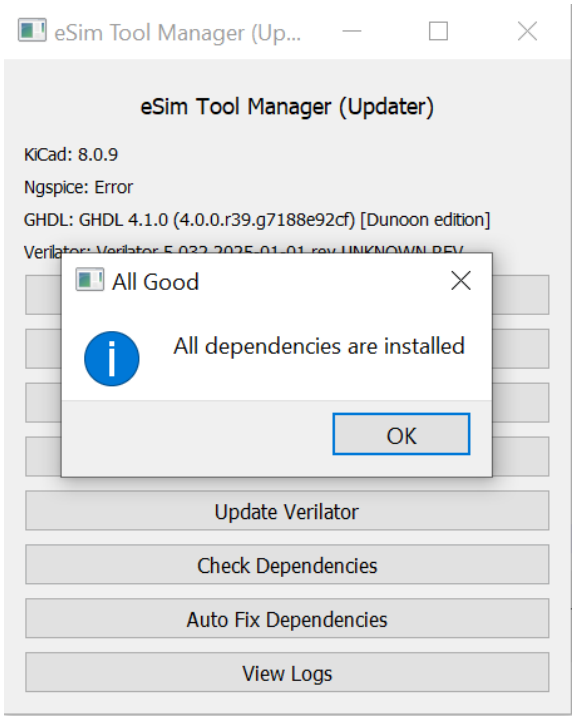

Figure 4.17

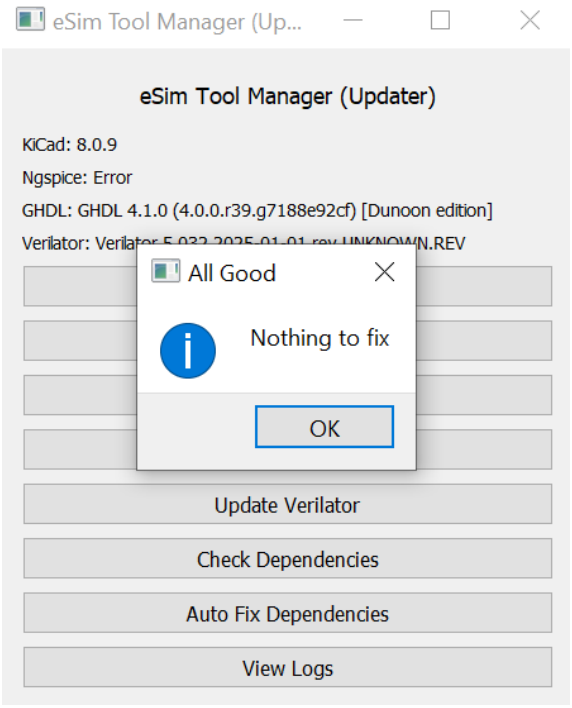

Figure 4.18

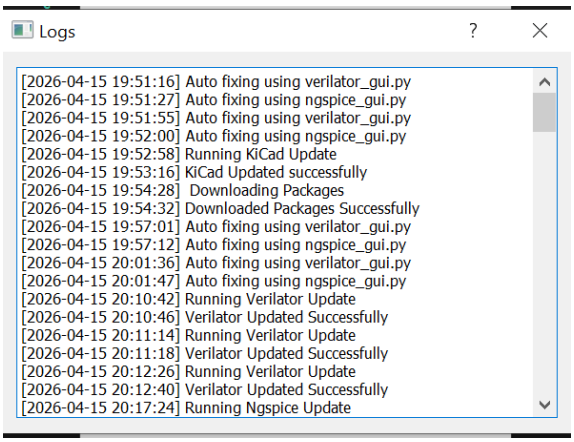

Figure 4.19

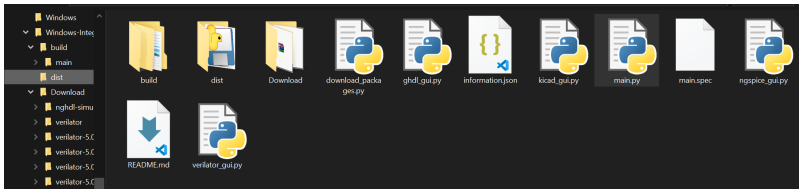

Figure 4.20

Chapter 5

Conclusion and Future Scope

The eSim Tool Manager project successfully improved the automation, usability, reliability, and maintainability of the eSim ecosystem. The project enhanced both standalone and integrated systems through feature development, debugging, stabilization, dependency management, GUI improvements, and multithreading implementation.

The system now provides better installation workflows, automatic updates, progress tracking, dependency verification, logging systems, and improved user interaction. These improvements significantly reduced manual effort and technical complexity for users.

The project also provided valuable real-world experience in software engineering, cross-platform development, debugging, GUI development, automation systems, package management, multithreading, and collaborative open-source contribution workflows.

Future improvements can include: MacOS support Cloud-based synchronization Automatic background updates Plugin marketplace integration Advanced analytics dashboard AI-based dependency management Enhanced installer engine Better cross-platform abstraction Real-time monitoring systems

Chapter 6

Bibliography

[1] FOSSEE eSim Project. eSim Resources. Available at: https://esim.fossee.in/resources

[2] Python Software Foundation. The Python Programming Language. Available at: https://www.python.org/

[3] Vogt, H. (n.d.). Ngspice, the open source Spice circuit simulator- Intro.. Available at: https://ngspice.sourceforge.io/

[4] KiCad EDA. (n.d.) KICAd EDA.. Available at: https://www.kicad.org/

[5] Chocolatey Chocolatey- The package manager for Windows.(n.d.). Chocolatey Soft ware. Available at: https://chocolatey.org/

[6] LLVM The LLVM Compiler Infrastructure project.(n.d.). Available at: https://llvm.org/