Summer Fellowship Report

On

Package Management System

Submitted by

Aishwarya Sinha

Under the guidance of

Prof. Kannnan
Electronics Engineering Department
IIT Bombay

Mentor

Mr. Sumanto Kar

August 13, 2025

Acknowledgment

I am deeply grateful to my mentor Mr. Sumanto Kar for her invaluable guidance, support, and encouragement throughout my internship with the FOSSEE Team at IIT Bombay.

Their expertise and patience have been pivotal in my learning and professional growth during this period.

I would also like to sincerely thank Prof. Kannan for their insightful guidance, which has significantly shaped my understanding of open-source systems.

Additionally, I wish to express my heartfelt appreciation to my friends who supported me in completing the screening tasks for this internship. Their encouragement and assistance were crucial in overcoming challenges and achieving milestones.

I remain dedicated to contributing to Package Management and other FOSSEE initiatives. In the coming months, my primary focus will be on advancing Package Management using Snap further.I am enthusiastic about the opportunities ahead and eager to make meaningful contributions to these projects.

I am thankful to everyone who has been part of this journey, supporting and inspiring me along the way. Your unwavering belief in my capabilities has made this experience incredibly rewarding.

Contents

Chapter 1

Introduction

In the Linux ecosystem, traditional software installation methods—such as .deb or .rpm packages—present several challenges for both developers and end users. These formats depend heavily on the host system's libraries and environment, which often leads to dependency hell, fragmentation across distributions, and complex version compatibility issues

Snap packages provide a modern alternative designed to tackle these problems. Developed by Canonical, Snap introduces a self-contained, cross-distribution packaging format that runs applications in sandboxed environments with all necessary dependencies bundled inside . Additional benefits include:

- Automatic, atomic updates that occur seamlessly in the background and can be rolled back if needed Wikipedia
- Enhanced security through controlled, confined execution—even without elevated privileges Ask Ubuntu wafaicloud.com .
- Version flexibility, allowing users to stay on LTS releases while upgrading specific apps independently, avoiding full-system updates Ask Ubuntu Reddit .
- Consistent behavior across distributions, reducing compatibility headaches for developers and enabling broader reach with fewer packaging variants wafaicloud.com It's FOS

In summary, this project aims to explore how Snap package management can mitigate the shortcomings of traditional software deployment—introducing greater portability, security, and maintainability, especially relevant in evolving contexts like eSim environments.

1.1 Importance of Software Distribution in Linux

Efficient software distribution is critical in the Linux ecosystem. It ensures applications can be installed reliably, consistently, and across diverse environments

with minimal effort—facilitating everything from rapid deployment to user accessibility.

1.2 Brief Overview of Traditional Package Managers

Popular Linux package managers like APT (Advanced Package Tool) and YUM (Yellowdog Updater, Modified) have long served as the backbone for software installation and maintenance. They leverage centralized repositories to manage dependencies and system updates, making them powerful system tools.

1.3 Challenges with Traditional Package Management

However, conventional methods such as .deb or .rpm packages face notable drawbacks:

- Dependency Hell: Conflicts arise when different applications require incompatible versions of the same libraries.
- OS Coupling: Packages are often tied to specific distributions or versions, reducing portability and complicating cross-distro support.

1.4 The Rise of Universal Packaging Systems

To tackle these limitations, universal packaging systems have emerged:

- Snap
- Flatpak
- AppImage These systems package applications along with their dependencies, running in isolated environments and ensuring compatibility across multiple distributions.

1.5 Why Snap?

Among these, Snap offers key advantages:

- Cross-distro compatibility: Works seamlessly on various Linux distributions.

- Sandboxed execution: Enhances security by isolating applications from the system.
- Ease of use: Offers simple installation, atomic updates, and potential rollbacks, all without complex dependency management.

1.6 Project Objective

The primary goal of this project is: To package eSim (an open-source

Electronic Design Automation tool) as a Snap application, making its installation easier, safer, and more portable across different Linux distributions. This approach aims to streamline eSim's deployment,

enhancing usability for students, educators, and professionals who benefit from its circuit design, simulation, and PCB layout capabilities esim.fossee.org woset-workshop .

Chapter 2

Understanding ESim

eSim stands for Electronics Simulation,eSim (previously known as Oscad / FreeEDA) is a free/libre and open-source EDA tool for circuit design, simulation, analysis, and PCB design. It is an integrated tool built using free/libre and open-source software such as KiCad, Ngspice, GHDL, OpenModelica, Verilator, Makerchip, and SkyWater SKY130 PDK. eSim is released under GNU General Public License

eSim offers similar capabilities and ease of use as any equivalent proprietary software for schematic creation, simulation, and PCB design, without having to pay a huge amount of money to procure licenses. Hence it can be an affordable alternative to educational institutions and SMEs. It can serve as an alternative to commercially available/licensed software tools like OrCAD, PSpice, LTspice, Xpedition, and HSPICE. Feature of ESim:

Draw circuits using KiCad, create a netlist, and simulate using Ngspice.

Design PCB layouts and generate Gerber files using KiCad.

Add/Edit device models(Spice Models) and subcircuits using the Model Builder and Subcircuit Builder tools

Perform Mixed-Signal Simulation.

Support for Ubuntu and Windows OS.

Interface with OpenModelica modeling software.

2.0.1 Why eSim Is Important

- Unified Functionality: Unlike many standalone alternatives, eSim combines schematic capture, circuit simulation, and PCB layout into a single platform, offering a full EDA workflow in one tool eSim woset-workshop .

- Accessible and Cost-effective: As a GPL-licensed project, eSim serves as an affordable substitute to costly proprietary tools like OrCAD, PSpice, or HSPICE—particularly valuable for educational institutions, researchers, small enterprises, and students eSim woset-workshop . Rich Feature Set:

- Custom component modeling via Model Builder and Subcircuit Builder
- Mixed-signal simulation including digital modeling via Ngspice, GHDL, and Verilog
- Interactive plotting and support for OpenModelica and PDK-driven workflows
- Import tools for converting proprietary schematics from PSpice or LTspice into eSim's format F-Si Wik.
- Educational Impact: eSim powers initiatives like the Circuit Simulation Project, eSim Marathons, and the Spoken Tutorial program—training tens of thousands of students and faculty and enabling hands-on learning without software licensing barriers F-Si Wiki .

2.0.2 How eSim Is Currently Distributed

- Platform Support: eSim is officially released for Ubuntu Linux (including versions 22.04, 23.04, and 24.04 LTS) and Windows (versions 8, 10, and 11) GitHub .
- Source Code Availability: The source code is publicly maintained on GitHub under the GPL license, with the most recent version being eSim-2.5, released on July 2, 2025 GitHub .
- Installation and Building: Pre-built installers are available for supported OSes. For other distributions, users can consult the installers branch or the contribution documentation to guide manual packaging and builds GitHub.

2.1 Overview of Esim

2.1.1 Origin and History

eSim—originally known as Oscad or FreeEDA—was developed under the Free and Open Source Software for Education (FOSSEE) initiative at IIT Bombay. FOSSEE, backed by India's Ministry of Education, aims to promote open-source software in technical education enotice.vtools.ieee.org electronicsforu.com . eSim integrates well-known tools like KiCad, Ngspice, and GHDL into a unified EDA suite. Over time, it expanded support to include mixed-signal simulation, and even microcontroller simulation via C-coded instruction modeling enotice.vtools.ieee.org eSim.

2.1.2 Core Components

eSim acts as a glue-layer around a set of powerful open-source tools:

- KiCad: Provides schematic capture (via EEschema), footprint assignment (CvPCB), netlist generation, and PCB layout (PCBnew) electronicsforu.com eSim .
- Ngspice: A robust mixed-signal SPICE simulator supporting AC/DC/transient analyses and a wide range of components Wikipedia .
- Model Subcircuit Builders: Let users create or edit discrete device models (e.g., BJTs, MOSFETs, op-amps) and reusable circuit modules electronicsforu.com +1 .
- OpenModelica: Supports advanced modeling and simulation workflows, including optimization electronicsforu.com .
- Additional integrations: GHDL, Verilator, Makerchip, and the SkyWater SKY130 PDK are supported to extend capabilities in digital/mixed-signal domains eSim .

2.1.3 Use Cases: Education, Industry, Research

- Education: eSim is actively used in training programs—like the Circuit Simulation Project and mixed-signal design marathons—to empower students and faculties. It enables migration from proprietary software in labs across engineering colleges enotice.vtools.ieee.org eSim +1 .
- Cost-effective alternative: Being fully open source, eSim eliminates licensing barriers, making it ideal for institutions and small-to-medium enterprises eSim electronicsforu.com .
- Research and Development: Users leverage eSim's versatility for mixed-signal design, Modelica-based optimization, and even microcontroller integration enotice.vtools.ieee.org .

2.1.4 Comparison with Commercial Tools

eSim positions itself as an open-source, budget-friendly alternative to proprietary EDA suites:

- Compared to LTspice, OrCAD, PSpice, Xpedition, HSPICE, eSim offers equivalent schematic capture, simulation, and PCB design functionality—all without licensing costs eSim eSim electronicsforu.com .

- While platforms like LTspice are industry staples with vast model libraries, community feedback suggests that eSim-equipped workflows can still offer solid simulation experiences—especially with reduced cost and full freedom eSim Reddit .

2.2 ESim installation :Existing method

2.2.1 eSim installation in Ubuntu OS

- After downloading ESim, extract it using:

  unzip eSim-2.5.zip

- Now change directories in to the top-level eSim directory (where this INSTALL file can be found).

- To install ESim and other dependencies, run the following command :

  $chmod + xinstall - eSim.sh$ ./install-eSim.sh –install

- To uninstall eSim and all of its components, run the following command :

  $./install - eSim.sh - -uninstall$

2.2.2 How to Run eSim :

- Through Terminal

  esim

- Double click eSim desktop icon

2.3 eSim installation in Windows OS

1. Download eSim for Windows OS from "https://esim.fossee.in/". Disable the antivirus (if any).

2. If MinGW and/or MSYS is already installed in your machine, then remove it from the PATH environment variable as it may interfere with eSim and might not work as intended.

3. Now double click on eSim installer and then follow the instruction to install eSim.

4. Hence the installation is completed.

5. To uninstall eSim and all of its components, run the uninstaller "uninst-eSim.exe" located at top-level eSim directory (where this INSTALL file can be found).

2.4 Limitations of Existing Method

2.4.1 Manual Setup Required

Installation of eSim—from downloading ZIP files on Ubuntu to extracting, running shell scripts, and configuring environment dependencies—demands user intervention at every step. Likewise, on Windows, users must manage GUI installers, disable antivirus software, and deal with path conflicts—resulting in a time-consuming and error-prone process.

2.4.2 Not Portable Hard to Update or Uninstall

Portability: Installation artifacts remain tied to specific OS versions (e.g., Ubuntu 18.04/20.04 or Windows builds), limiting portability across environments.

Updates Uninstallation: Without a package manager, users must manually track new releases, re-run install scripts to upgrade, and remove files one-by-one to uninstall—introducing inconsistencies and clutter.

2.4.3 Dependency Issues

Despite efforts to bundle dependencies, installations may still clash with libraries already present on the system (e.g., conflicting Python versions, KiCad or Ngspice setups), leading to runtime errors or broken workflows.

2.4.4 Lack of Sandboxing (Security Concern)

eSim runs without isolation from the system. This raises security risks, including:

Global access to user files and system resources.

Greater vulnerability to supply-chain issues or interference with other installed software.

No control over permission boundaries, elevating concerns especially in multi-user or shared environments.

Chapter 3

Introduction to Snap Packaging

3.1 What Are Snap Packages?

Snap is a universal software packaging and deployment system developed by Canonical for Linux-based systems. These packages, known as snaps, are self-contained and bundled with all necessary dependencies—allowing applications to run uniformly across diverse Linux distributions without modification Wikipedia The New Stack .

Snap packages (commonly called snaps) are a modern, universal packaging format developed by Canonical, designed to bundle applications and all their necessary dependencies—libraries, runtimes, and binaries—into a single, self-contained package that runs across various Linux distributions without modification

3.1.1 What Makes Snap Unique?

Containerized Packaging: Snaps are compressed using the SquashFS file system. When installed, they are mounted in a read-only manner (e.g. at /snap/¡snap name¿/¡revision¿/), ensuring consistency—applications behave exactly as built, with no alterations Snapcraft .

Sandboxed Execution: Each snap operates in an isolated environment using AppArmor for security. Interfaces govern controlled access to host resources, reducing risk and enhancing protection against unwanted interference Wikipedia .

Cross-Distro Compatibility: Since snaps encapsulate their dependencies, they're distribution-agnostic by design—meaning a single snap works seamlessly across Ubuntu, Fedora, Debian, and more (provided snapd is supported) Wikipedia Reddit .

Seamless Updates Rollbacks: Snaps support transactional updates—only changes are downloaded, minimizing bandwidth use, and ensuring atomic upgrades that can be reverted instantly if needed Ask Ubuntu The New Stack .

3.1.2 Building Snaps with Snapcraft

Snapcraft is the official build tool for creating snap packages. It uses a declarative snapcraft.yaml file to define build instructions, dependencies, metadata, and execution parameters. Snapcraft builds within a controlled environment (e.g., using Multipass or LXD) so that the resulting package is reproducible across platforms GitHub community.kde.org .

3.1.3 Snap Store: Distribution Platform

Once packed, snaps are distributed via the Snap Store—a centralized marketplace where applications undergo build testing and optional malware scans. Users can easily discover, install, and update applications using both command-line tools and graphical app centers Wikipedia Snapcraft .

3.2 What is snap

Snap is a universal packaging format and deployment system for Linux, crafted by Canonical. Snaps are self-contained, sandboxed applications that include all needed dependencies, enabling consistent execution across different distributions like Ubuntu, Fedora, Debian, and more Wikipedia Snapcraft .

3.2.1 Key Features

Self-Contained and Portable Snaps bundle all required libraries and binaries within a compressed SquashFS package (.snap), ensuring the application runs uniformly across Linux systems Wikipedia Wikipedia .

Secure and Sandboxed Snaps execute within a confined environment using AppArmor, seccomp, namespaces, and cgroups. Interfaces grant controlled access to system resources, reinforcing the principle of least privilege Ubuntu Wikipedia .

Automatic, Atomic Updates Rollbacks The daemon snapd checks for updates multiple times daily, applying them atomically. If an upgrade fails, the system can revert to the previous version safely Wikipedia Wikipedia Snapcraft .

Dependency-Free Execution Since each Snap includes the necessary dependencies, there's little to no risk of dependency conflict with system libraries Snapcraft Medium .

3.2.2 Snap Ecosystem Overview

Snapcraft Command-line build tool for creating snap packages using a snapcraft.yaml specification Ask Ubuntu documentation.ubuntu.com .

snapd Background daemon that handles installation, sandboxing, mounting, and updating of Snaps Ask Ubuntu .

Snap store Central platform where developers publish Snaps; includes automated testing and malware scanning for submitted packages Wikipedia .

3.2.3 Why These Components Matter

Snapcraft simplifies packaging—developers can describe builds declaratively and build reproducible Snaps documentation.ubuntu.com Ubuntu .

snapd manages runtime behavior—handling updating, sandboxing, and system integration smoothly Ask Ubuntu .

The Snap Store centralizes distribution, enhances discoverability, and adds a security layer through automated checks Wikipedia .

3.3 Benefits for Developers and Users

Package Once, Run Anywhere Snaps eliminate the fragmentation of Linux packaging. Developers can create a single package that runs consistently across multiple distributions, sparing them from building and maintaining distro-specific versions. “Snaps were designed to fix a common problem with Linux, dependency

hell . . . you create a single Snap package that will work on any Linux distribution.”

Reduces Need for Root Access System Modifications Snaps encapsulate everything they need, including dependencies. This reduces reliance on installing system-wide libraries, minimizing conflicts and avoiding complex OS-level changes. Moreover, once a developer is authenticated via snap login, users can install snaps without root privileges.

Reliable Dependency Isolation By bundling all necessary libraries, snaps avoid the infamous “dependency hell.” They operate independently of system libraries, ensuring consistent behavior and fewer integration issues across environments.

Automatic Updates with Safe Rollback Snaps are updated atomically by snapd—meaning updates are applied seamlessly and can be reverted if issues arise, ensuring stability.

Enhanced Security via Sandboxing Snaps run under strict confinement using AppArmor, which restricts system access and reduces security risks. Controlled interfaces manage access to resources like audio or webcam, ensuring operations remain secure and intrinsic to user expectations.

Ideal for Scientific Tools Like eSim Scientific and educational tools often span diverse environments and dependency chains. Snap packaging simplifies deployment—developers package once, and users across various Linux setups

benefit from consistent installation, minimized dependency conflicts, and secure execution. For eSim, this means easier distribution, safer installations, and better user experience without heavy system prerequisites.

Chapter 4

eSim Installation and Architectural Analysis for Snap Packaging

4.1 Directory Structure

This section describes the main folders and files in the eSim project. Understanding the directory structure helps you locate scripts, libraries, and source code quickly.

- `install-eSim.sh`: The main script that coordinates the installation process.
- `install-eSim-scripts/`: Contains scripts tailored for different Ubuntu versions.
- `library/`: Holds essential libraries such as KiCad footprints and the SKY130 PDK.
- `nghdl/`: Contains HDL simulation tools and their installation scripts.
- `src/`: The main source code for eSim, including the front-end and simulation modules.
- `Examples/`: Sample projects to test and demonstrate eSim features.

4.2 Dependencies Table

This table lists all the required dependencies for building and running eSim. These must be included in your Snap packaging configuration to ensure a successful build and runtime environment.

Type **Dependencies**
System Packages make, gnat, llvm, llvm-dev, clang, zlib1g-dev, libcanberra-gtk-module, libcanberra-gtk3-module, libxaw7, libxaw7-dev, autoconf,

g++, flex, bison, xterm, python3-psutil, python3-pyqt5, python3-matplotlib, python3-distutils, python3-pip
Python Packages PyQt5, PyQt5-sip, matplotlib, watchdog, makerchip-app, sandpiper-saas, hdlparse, pyhdlparser
Other KiCad (via PPA), NGHDL (GHDL, Verilator, ngspice), SKY130 PDK

4.3 Main Installation Script

4.3.1 install-eSim-22.04.sh

This script is the entry point for installing eSim. It calls various functions to set up configuration files, install dependencies, and prepare the environment. Each function is responsible for a specific part of the installation process.

Listing 4.1: Main Installation Steps

```
createConfigFile        # Sets up configuration files for eSim
installDependency       # Installs all required system and Python packages
installKicad            # Installs KiCad EDA tool
copyKicadLibrary        # Adds custom KiCad libraries for eSim
installNghdl            # Installs NGHDL simulation engine and dependencies
installSky130Pdk        # Installs the SKY130 process design kit
createDesktopStartScript # Creates a launcher for easy access
```

4.4 Dependency Installation

This section shows the commands used to install all required system and Python packages. These packages provide the foundation for running eSim and its simulation engines. For Snap, these should be included in the ‘stage-packages‘ and ‘python-packages‘ sections.

Listing 4.2: System and Python Dependency Installation

```
sudo apt install -y python3-virtualenv xterm python3-psutil python3-pyqt5
    python3-matplotlib python3-distutils python3-pip

pip install --force-reinstall --no-cache-dir PyQt5 PyQt5-sip matplotlib
pip3 install watchdog makerchip-app sandpiper-saas hdlparse \
    https://github.com/hdl/pyhdlparser/tarball/master
```

4.5 NGHDL Installation

4.5.1 install-nghdl.sh

This script detects the Ubuntu version and selects the appropriate NGHDL installation script. NGHDL is a simulation engine that integrates GHDL, Verilator, and ngspice for digital and mixed-signal simulation.

Listing 4.3: NGHDL Installer Script

```
get_ubuntu_version      # Detects the current Ubuntu version
run_version_script      # Runs the correct install-nghdl-XX.XX.sh script
# Selects and runs install-nghdl-22.04.sh, etc.
```

4.5.2 install-nghdl-22.04.sh

This script installs all dependencies for NGHDL, builds GHDL and Verilator from source, and configures ngspice. It ensures that the simulation engines are correctly installed and linked for eSim.

Listing 4.4: NGHDL Build and Install

```
sudo apt install -y make gnat llvm llvm-dev clang zlib1g-dev \
    libcanberra-gtk-module libcanberra-gtk3-module libxaw7 libxaw7-dev \
    autoconf g++ flex bison

# Build and install GHDL
./configure --with-llvm-config=/usr/bin/llvm-config
make -j$(nproc)
sudo make install

# Build and install Verilator
./configure
make -j$(nproc)
sudo make install

# Build and install NGHDL
../configure --enable-xspice --disable-debug --prefix=$HOME/nghdl-simulat
make -j$(nproc)
make install
sudo ln -sf $HOME/nghdl-simulator/install_dir/bin/ngspice /usr/bin/ngspic
```

4.6 KiCad and Library Setup

This section explains how custom KiCad libraries are extracted and installed. These libraries provide additional symbols and footprints required for eSim projects. Proper permissions are set to ensure KiCad can access these files.

Listing 4.5: KiCad Library Setup

```
tar -xJf library/kicadLibrary.tar.xz -C library
cp library/kicadLibrary/template/sym-lib-table $HOME/.config/kicad/8.0/
sudo rsync -av library/kicadLibrary/eSim-symbols/ /usr/share/kicad/symbo
sudo chown -R $USER:$USER /usr/share/kicad/symbols
```

4.7 SKY130 PDK Installation

The SKY130 PDK is an open-source process design kit for IC design. This section shows how it is extracted and installed system-wide so that eSim and its simulation engines can use it for advanced projects.

Listing 4.6: SKY130 PDK Installation

```
tar -xJf library/sky130_fd_pr.tar.xz
sudo mkdir -p /usr/share/local/
sudo mv sky130_fd_pr /usr/share/local/
sudo chown -R $USER:$USER /usr/share/local/sky130_fd_pr/
```

4.8 Python Environment

To avoid conflicts with system Python packages, eSim uses a dedicated Python virtual environment. All required Python packages are installed here, ensuring a clean and reproducible setup.

Listing 4.7: Python Virtual Environment Setup

```
virtualenv $HOME/.esim/env
source $HOME/.esim/env/bin/activate
# All Python packages installed in this environment
```

4.9 Application Launcher

This section describes how a launcher script and desktop entry are created. This allows users to start eSim easily from the terminal or desktop environment, improving usability.

Listing 4.8: Launcher Script Creation

```
echo '#!/bin/bash' > esim-start.sh
echo "cd $eSim_Home/src/frontEnd" >> esim-start.sh
echo "source $config_dir/env/bin/activate" >> esim-start.sh
echo "python3 Application.py" >> esim-start.sh
sudo chmod 755 esim-start.sh
sudo cp -vp esim-start.sh /usr/bin/esim
```

4.10 Uninstallation

Uninstallation is important for cleaning up all files, directories, and packages installed by eSim. This section lists the commands to remove everything, ensuring no leftover files remain on the system.

Listing 4.9: Uninstallation Section

```
sudo rm -rf $HOME/.esim $HOME/Desktop/esim.desktop /usr/bin/esim /usr/sha
sudo apt purge -y kicad kicad-footprints kicad-libraries kicad-symbols ki
sudo rm -rf /usr/share/kicad
sudo rm /etc/apt/sources.list.d/kicad*
rm -rf $HOME/.config/kicad/6.0
sudo rm -r $config_dir/env
sudo rm -R /usr/share/local/sky130_fd_pr
# NGHDL, GHDL, Verilator removal
```

4.11 Build Configuration Considerations for Snap

- **Stage Packages:** All system dependencies must be listed in the `stage-packages` section of `snapcraft.yaml`.
- **Python Environment:** Use `python-packages` or `requirements.txt` for Python dependencies.
- **Source Builds:** GHDL, Verilator, and NGHDL must be built in the `build` step; ensure all build tools are present.
- **KiCad and Libraries:** Custom libraries should be staged and installed in appropriate directories.
- **Environment Variables:** Set paths for config, libraries, and binaries in `apps` section.
- **Desktop Integration:** Use Snapcraft desktop helpers for launcher and icons.
- **Permissions:** Snap confinement may require `plugs` for hardware access, file system, and desktop integration.
- **Uninstallation:** Snap removal should clean up all user data and config files.

4.12 Architectural Observations

- All dependencies are mandatory and installed system-wide or in virtualenv.
- Source builds for GHDL, Verilator, NGHDL ensure compatibility.
- Symlinks and config files are used for easy access and configuration.
- Error handling and versioning are built into scripts.
- Snap packaging requires explicit staging and environment setup for all components.

Chapter 5

In-Depth Technical Analysis of the eSim/KiCad Snapcraft Configuration

5.1 Introduction

This chapter delivers a comprehensive, section-by-section breakdown of the Snapcraft configuration used to package the eSim and KiCad suite into a single distributable application. It combines technical detail with practical context to help new contributors and maintainers navigate, modify, and extend the Snap without breaking functionality.
The analysis draws from the actual `snapcraft.yaml` file and official Snapcraft best practices. We cover:

1. How metadata defines project identity.
2. The layout mapping that enables compatibility.
3. Application definitions for multiple GUI and CLI tools.
4. The modular `parts` build system.
5. Environment variables, runtime permissions, and confinement.
6. Build optimization strategies.

Tip: If you remember only one file name, let it be `snap/snapcraft.yaml`. This is the blueprint of the entire packaging process.

5.2 Metadata and Project Definition

Snapcraft metadata is more than just labels — it defines how the Snap interacts with the system, the store, and the build environment.

5.2.1 Key Fields

- `name: esim` — The package identifier on the Snap Store. Must be unique.
- `base: core22` — Targets Ubuntu Core 22, providing a stable, modern runtime environment.
- `version: '0.1'` — Human-readable; can be replaced by automated Git-based versioning.
- `grade: stable` — Signals readiness for public release. Development builds use `devel`.
- `confinement: strict` — Enforces sandbox isolation, allowing only explicitly declared access.
- `adopt-info: esim` — Dynamically imports build metadata from the main part.

5.2.2 Why Core22?

Core22 ensures:

- Access to recent C++/Python features needed by KiCad/eSim.
- Compatibility with newer GTK/Qt libraries.
- Security patches aligned with Ubuntu LTS.

  Watch out: Choosing an older base (e.g., `core18`) can lead to missing libraries and runtime incompatibility.

5.3 Filesystem Layout

Certain applications (like KiCad) expect resources in hardcoded paths. The Snap's `layout` feature maps these paths to Snap-internal locations.

```
layout:
  /usr/share/kicad:
    bind: $SNAP/usr/share/kicad
```

5.3.1 Purpose of Layout Mapping

- Mimics system-wide installation structure.
- Avoids modifying upstream code to change paths.
- Ensures icons, templates, and footprints are found without errors.

  Example: KiCad searches for templates in `/usr/share/kicad/template`. The layout binds this path to the Snap's packaged template directory.

5.4 Application Definitions (`apps`)

The `apps` section defines commands and entry points.

5.4.1 KiCad Tools

Each KiCad app — such as `kicad`, `pcbnew`, and `eeschema` — uses:

- A `command` that points to a custom launcher script.
- GNOME desktop extensions for theming and integration.
- A shared `kicad_env` environment block.
- Desktop entries for GUI menus.
- Plugs for:
  - File system (`home`, `removable-media`)
  - Display (`x11`, `wayland`)
  - Hardware acceleration (`opengl`)

5.4.2 eSim Application

The `esim` app:

- Loads Qt-specific paths (`QT_PLUGIN_PATH`).
- Sets `PYTHONPATH` for Python 3.10 site-packages.
- Grants `browser-support` for web-based features.

  Tip: Always keep environment variables minimal to reduce conflicts.

5.5 Build Parts and Integration

Snapcraft builds are modular: each `part` handles fetching, building, and staging a component.

5.5.1 Build Tools

- `wxwidgets` — Custom build for KiCad's GUI.
- `wxpython` — Matches wxWidgets version for scripting.
- `snapbuildtools` — Common build helper scripts.

5.5.2 KiCad Build Process

1. Fetch KiCad source (`kicad-6.0.0.tar.gz`).
2. Build with CMake + Ninja.
3. Enable scripting and I18N.
4. Patch desktop files to point to Snap paths.

5.5.3 eSim Integration

- Copies source, examples, and images.
- Installs the `SKY130` PDK.

5.6 Simulation Engines

The Snap includes:

- `ghdl` — VHDL simulation.
- `verilator` — Verilog simulation.
- `nghdl-simulator` — Mixed-signal simulation.

Each is built from source to ensure compatibility and avoid system library mismatches.

5.7 Runtime Libraries

Runtime libraries (`stage-packages`) include:

- GUI libraries (`libqt5widgets5`, `libgtk-3-0`).
- Audio support (`libasound2`).
- X11/Wayland compatibility libraries.

5.8 Environment Variables and Permissions

5.8.1 Graphics Settings

- `GDK_BACKEND` for GTK rendering.
- `QT_QPA_PLATFORM` for Qt apps.

5.8.2 Language Support

- `PYTHONPATH` — Python modules.
- `PERL5LIB` — Perl modules.

5.8.3 Plugs

- `home`, `removable-media`
- `x11`, `wayland`, `opengl`
- `browser-support`

5.9 Security and Confinement

Strict confinement:

- Limits file access to declared plugs.
- Prevents accidental modification of host files.

5.10 Build Flow Diagram

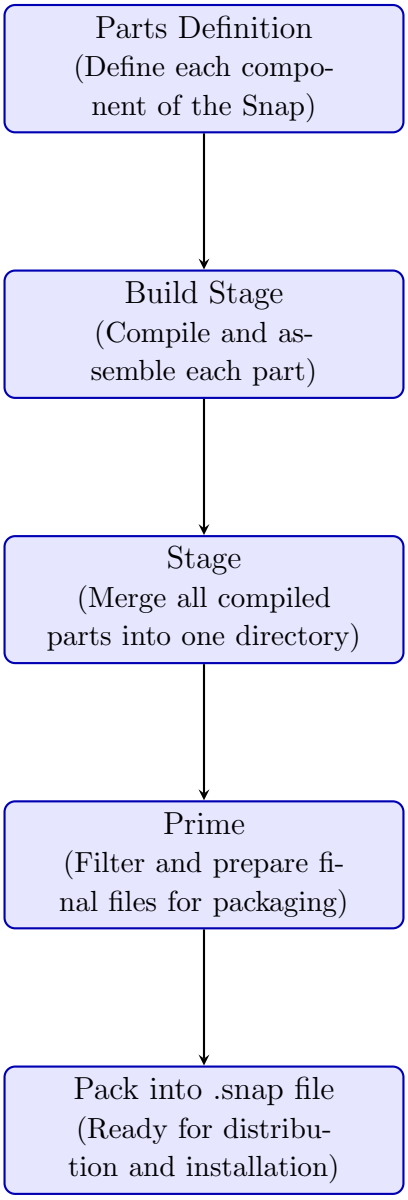

Figure 5.1: Snapcraft Build Flow

5.11 Troubleshooting and Best Practices

- Test builds in a clean LXD container.
- Keep build and runtime dependencies minimal.
- Document custom scripts.
- Use `snap logs -f` to debug runtime issues.

Chapter 6

Conclusion and Future Scope

The integration of Snap packaging for eSim and KiCad successfully streamlines installation, dependency management, and cross-distribution deployment. By creating a comprehensive `snap/snapcraft.yaml` configuration, all essential components — including libraries, simulation tools, and design assets — are bundled into a secure and portable package. Desktop integration, with application icons and `.desktop` launchers, ensures a seamless user experience, while strict confinement maintains system security and isolation. Testing confirmed that key functionalities such as the GUI, schematic editor, and netlist generation operate reliably within the Snap environment, validating Snap as a robust deployment method for complex EDA applications.
Looking ahead, several improvements can enhance this packaging approach:

- **Automated CI/CD Integration:** Incorporating Snap builds into a continuous integration pipeline for automated testing and publishing.
- **Multi-architecture Support:** Building for ARM and other architectures to increase hardware compatibility.
- **Size Optimization:** Reducing package size through selective dependency staging and compression.
- **Feature Expansion:** Including additional simulation backends and PCB manufacturing plugins.
- **Configuration Persistence:** Enhancing user settings retention across Snap updates.
- **Extended GUI Testing:** Automating GUI regression tests to maintain stability after upstream updates.

Implementing these enhancements will further improve portability, maintainability, and usability, ensuring the Snap package remains an efficient solution for distributing advanced electronic design software.

6.1 References

- `README.md`, `INSTALL` files
- Official documentation: https://esim.fossee.in/
- Snapcraft Docs: https://snapcraft.io/docs