



Semester Long Internship Report

On

eSim Tool Manager and Packaging

Submitted by

Suchinton Chakravrty

Under the guidance of

Prof.Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

May 17, 2023

Acknowledgment

I would like to express my sincerest gratitude to the entire FOSSEE team for affording me the esteemed opportunity of being a part of this Fellowship. Working with an open-source organization like FOSSEE has been an enlightening experience, allowing me to delve into the intricacies of simulation software such as eSim and acquire valuable problem-solving skills. I am immensely grateful for this learning experience, which I will treasure in the years to come throughout my career. I extend my heartfelt appreciation to Prof. Kannan M. Moudgalya for his visionary leadership and dedication to advancing this initiative. It is through his guidance that I was granted the privilege of contributing as an intern, however small my role may have been.

Furthermore, I would like to express my warmest gratitude to the entire FOSSEE team, including our mentors Mr. Sumanto Kar, Mr. Rahul Paknikar, and Mrs. Usha Vishwanathan. Their constant support and mentorship throughout my internship have been invaluable. Whenever I encountered challenges, they were readily available to provide guidance and help me navigate through the complexities, leading me to effective solutions. I hold their teachings in the highest regard and will carry them forward in my future endeavours.

I also wish to extend my special thanks to my fellow interns, whose unwavering support and motivation have been instrumental in our collaborative efforts towards the success of this project.

Overall, my time interning at FOSSEE has been a delightful experience, contributing to both personal growth and the development of FOSSEE. I have gained profound insights and knowledge that will undoubtedly shape my future. As an aspiring professional in the semiconductor industry, this internship serves as a significant milestone on my path towards a successful career.

Contents

1	Introduction	4
1.1	eSim	4
1.2	Flatpak	4
2	Task Chosen: Tool Manager and Packaging	6
3	Methodology	7
4	Implementation	9
4.1	Defining the Flatpak Manifest	10
4.1.1	run.sh	10
4.1.2	org.flatpak.FOSSEE_Inst_test.yml	10
4.1.3	python3-py7zr.json	13
4.1.4	git.json	15
4.2	Main Program: FOSSEE_Inst_test.py	16
4.2.1	class AppWindow(...):	16
4.2.2	install_Pip_package():	18
4.2.3	install_Verilator_from_Archive():	18
4.2.4	install_Verilator_SRC():	19
4.2.5	QThread classes:	20
4.2.6	Main Function:	21
4.3	Install Script: Install_verilator.sh	21
4.4	Testing script: test	23
5	Results	24
5.1	GUI & Launching procedure	24
5.2	Installing Python Dependencies for eSim:	25
5.3	Installing Verilator from Archived package(Pre-built):	25
5.4	Installing Verilator from Source:	26

6	Workflow	27
6.1	build_new_flatpak.sh	27
6.2	flapak-pip-generator	28
7	Conclusion and Future Scope	29
7.1	Conclusion	29
7.2	Future Work	29
	Bibliography	30

Chapter 1

Introduction

FOSSEE (Free/Libre and Open Source Software for Education) is an initiative taken by the National Mission on Education through Information and Communication Technology (ICT), Ministry of Human Resource Development (MHRD), Government of India which has successfully developed various open- source tools and promotes the use of these tools in improving the quality of education and helping every individual avail these sources free of cost. The software is being developed in such a way that it can stay relevant with respect to the commercial software.

1.1 eSim

eSim is a free/libre and open source EDA tool for circuit design, simulation, analysis and PCB design developed by FOSSEE, IIT Bombay. It is an integrated tool built using free/libre and open source software such as KiCad, Ngspice, NGHDL and GHDL.[4]

1.2 Flatpak

Flatpak is a software distribution and package management system for Linux. It provides a way to package applications and their dependencies in a self-contained manner, making it easier to distribute and run applications across different Linux distributions.[1] Here are some key features of Flatpak:

- **Sandboxed Environment:** Flatpak packages are sandboxed, meaning they run in isolation from the rest of the system. This enhances security and prevents conflicts between applications or libraries.
- **Dependency Management:** Flatpak allows applications to bundle their dependencies, ensuring that they have the required libraries and components available, regardless of the host system's configuration.
- **Cross-Distribution Compatibility:** Flatpak packages are designed to be portable across different Linux distributions. This means developers can create a single package that works on multiple distributions, reducing the need for distribution-specific packaging.

- **Runtime Environment:** Flatpak provides a runtime environment that contains a set of common libraries and components required by applications. This allows applications to rely on a consistent and up-to-date runtime, reducing the need to bundle large amounts of dependencies within each m package.

Flatpak simplifies the packaging, distribution, and installation of Linux applications by providing a standardized and secure platform. It promotes cross-distribution compatibility and sandboxing, making it easier for developers to reach a wider audience and for users to install and run applications without worrying about system conflicts or complex dependency management.

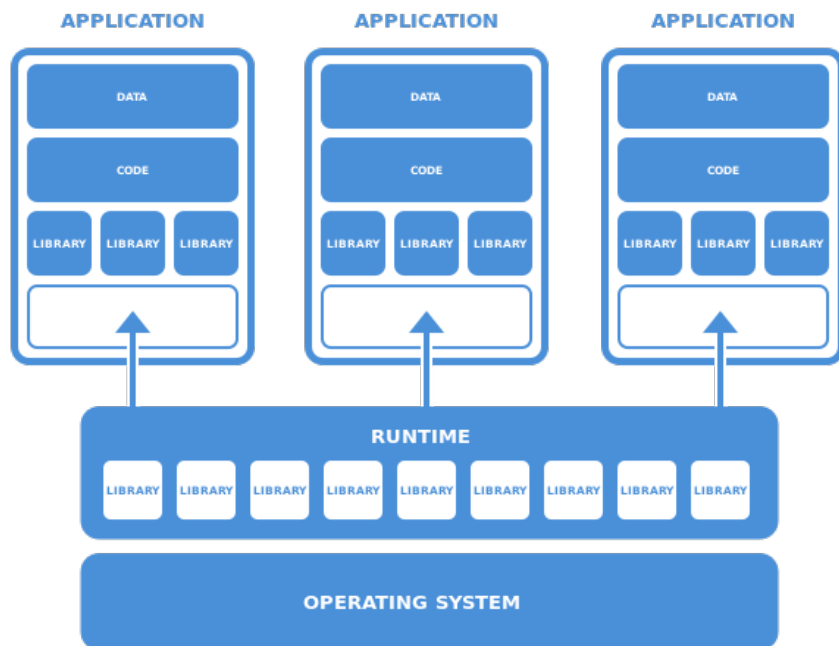


Figure 1.1: Flatpak general scheme

Chapter 2

Task Chosen: Tool Manager and Packaging

Currently, eSim only supports limited versions of some Linux Distributions, namely Ubuntu and Fedora due to the dependency constraints on other platforms as they might be hosted under different names, use different package managers, have different packaging formats, or have outdated packages. This means that the dependencies need to be resolved by self-hosting packages, compiling on the host machine and using alternative packaging management systems.

To address these concerns, my task was to implement a proof-on-concept installer application for eSim which can,

- Install Python dependencies (using PiPy) for eSim in User space while the installer remains sandboxed
- Install pre-built and pre-compiled binaries in the user space/ root file system, including their libraries
- Partially build the packages from source in a sandboxed environment and install them in user space.

In order to meet the requirements, we thought to design an m using the Flatpak packaging format, this way more Linux distributions could be supported without worrying about the native dependencies required to run the installer.

The Installer was created such that it would be -

- User-friendly
- Easily implemented
- Easily scalable
- Platform Independent
- and have its dependencies bundled

Chapter 3

Methodology

To implement the above points, we have taken the following approach:

- `flatpak-spawn --host` is used in the applications to run commands/ system calls with access to resources outside the sandbox. It allows the application to interact with the host system, such as accessing hardware devices or files. Thereby bypassing the limitations of using a sandboxed environment for an installer application.
- Next we need to have certain build tools available during runtime to compile/ build packages, this means that `org.kde.Platform` can not be used as the application's runtime, to overcome this we mount the SDK `org.kde.SDK` as the Runtime, thereby providing us with the necessary build tools during runtime.
Note: This is only a temporary solution and any future work in this direction should consider implementing the runtime dependencies as modules to the `org.kde.Platform` runtime.
- Since the builds take a long time to compile, the GUI must remain responsive, therefore `QThread`s are used to allow for processes to execute parallelly and show real-time installation updates in the GUI.
- To make the Application extensible, execution of individual scripts must be supported such that newer dependencies/ packages can be supported just by adding new installation scripts.

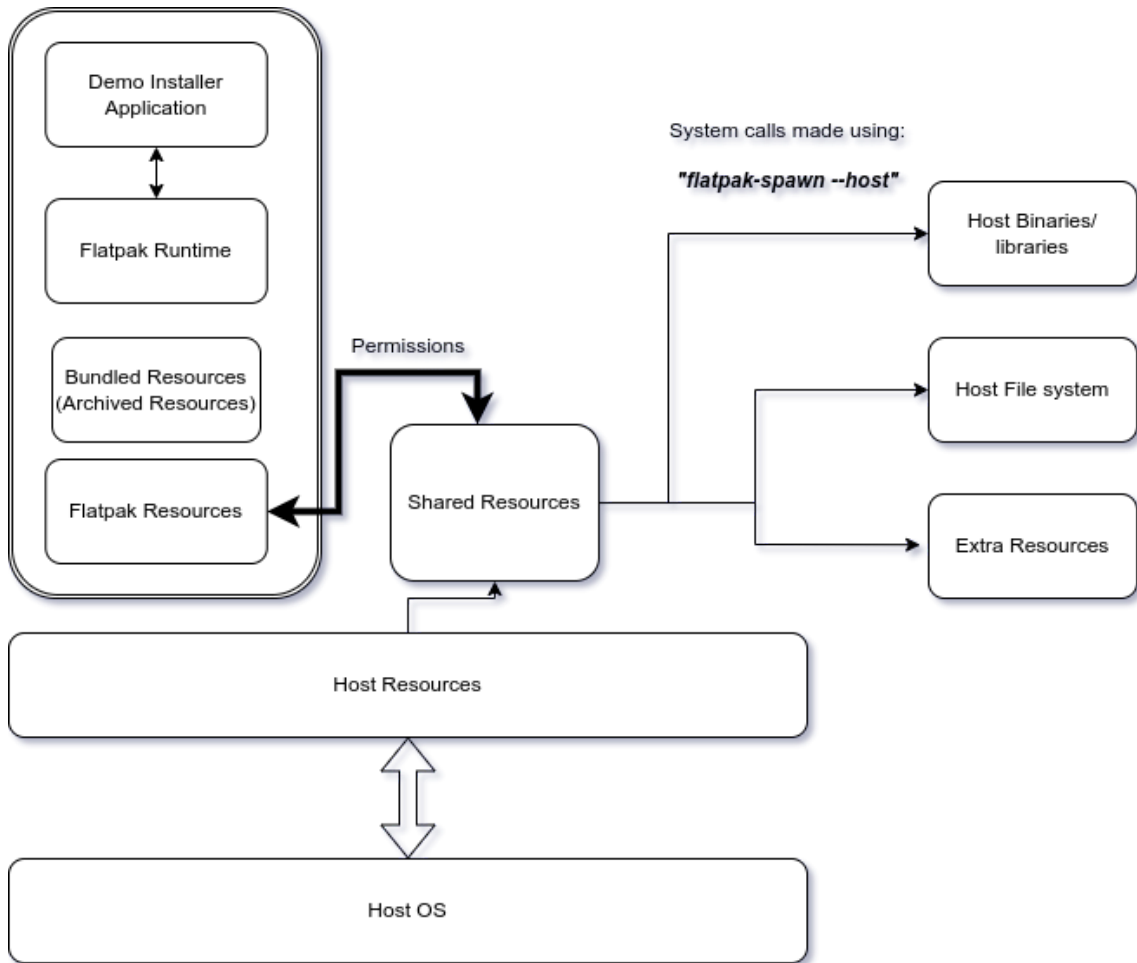


Figure 3.1: Flatpak general scheme

As seen in the Figure above, the flatpak application has bundled resources, to address time constraints, supports using the host resources to install packages using host resources only and also supports partial builds in the Flatpak file system and completion in the host file system.

Chapter 4

Implementation

The application has multiple stages and functions that will be discussed sequentially. The Application's runtime dependencies include; py7zr & pyqt5

Additional dependencies, on the host system, are:

- build tools
- make
- ccache
- python3 & python3-pip (pip3)

For reference, the File structure for the application's repository can be seen below.

```
Flatpak_Test_App/  
├─ flatpak  
│  ├─ assets  
│  │  └─ Fossee_logo.png  
│  │  └─ packages  
│  │     └─ verilator.7z  
│  │     └─ scripts  
│  │        └─ Install_verilator.sh  
│  │        └─ ngspice  
│  │        └─ test  
│  └─ FOSSEE_Inst_test.py  
├─ git.json  
├─ org.flatpak.FOSSEE_Inst_test.yml  
├─ python3-py7zr.json  
├─ run.sh  
├─ Readme.md  
├─ req  
└─ scripts  
   └─ build_new_flatpak.sh  
      └─ flatpak-pip-generator
```

Figure 4.1: Repo File structure 2

4.1 Defining the Flatpak Manifest

A Flatpak manifest is a configuration file that describes the structure and behaviour of a Flatpak application. It provides instructions for building, packaging, and running the application as a Flatpak package.

4.1.1 run.sh

This shell script is executed every time our Flatpak application is launched.

```
#!/bin/sh
python3 -u /app/bin/FOSSEE_Inst_test.py
```

4.1.2 org.flatpak.FOSSEE_Inst_test.yml

The manifest provides the necessary instructions for building and running the application within the Flatpak framework.

- Runtime Configuration: The application utilizes the runtime environment provided by the "org.kde.Sdk" runtime with a specific version of '5.15-22.08'. This runtime provides the necessary dependencies and libraries for the application to run.
- Base Application: The base application for this Flatpak is specified as "com.riverbankcomputing" with a version of '5.15-22.08'. The base application serves as the foundation upon which the application is built.[7]
- Command: The command to execute the application is specified as "run.sh". This script is responsible for launching the application within the Flatpak sandboxed environment.
- Finish Arguments: The finish arguments define the permissions and access rights granted to the application within the Flatpak sandbox. These include:
 - Sharing inter-process communication (IPC)
 - Sharing network access
 - Access to the X11 display server
 - Access to the Wayland display server
 - Access to the host operating system's filesystem
 - Access to the host filesystem
 - Access to the user's home directory
 - Access to the Direct Rendering Infrastructure (DRI)
 - Ability to communicate with the org.freedesktop.Flatpak interface

- Cleanup: The cleanup section specifies the files and directories that should be removed after the application has been built. In this case, the `"/include"` directory and any files with the `".a"` extension will be deleted.
- Files: The files section describes the file permissions and access rights for specific paths within the application's sandbox. In this case, the `"/verilator/*"` path is readable and writable.
- Modules: Modules define the components and build instructions for the application. The Flatpak manifest contains the following modules:

```

app-id: org.flatpak.FOSSEE_Inst_test
runtime: org.kde.Sdk
runtime-version: '5.15-22.08'
sdk: org.kde.Sdk
base: com.riverbankcomputing.PyQt.BaseApp
base-version: 5.15-22.08

command: run.sh

finish-args:
- --share=ipc
- --share=network
- --socket=x11
- --socket=wayland
- --filesystem=host-os
- --filesystem=host
- --filesystem=home
- --device=dri
- --talk-name=org.freedesktop.Flatpak

cleanup:
- '/include'
- '*.a'

files:
- path: /verilator/*
  read: true
  write: true

modules:
- name: FOSSEE_Inst_test
  buildsystem: simple
  build-commands:
    - install -D FOSSEE_Inst_test.py /app/bin/FOSSEE_Inst_test.py
    - install -D run.sh /app/bin/run.sh

```

```
sources:
  - type: file
    path: FOSSEE_Inst_test.py
  - type: file
    path: run.sh

- name: assets
  buildsystem: simple
  build-commands:
    - cp -r assets/ /app/assets/

sources:
  - type: dir
    path: assets/
    dest: assets/

- python3-py7zr.json
```

4.1.3 python3-py7zr.json

This module is used for handling 7z archives in Python. It provides functionality to create, extract, and manipulate 7z files. We use this in our application to extract prebuilt binaries and additional libraries and export them to the host file system. To resolve this dependency we install the package and its own dependencies using pip (provided by the flatpak SDK and not the host system)

```
{
  "name": "python3-py7zr",
  "buildsystem": "simple",
  "build-commands": [
    "pip3 install --verbose --exists-action=i --no-index
↪ --find-links=\"file://${PWD}\" --prefix=${FLATPAK_DEST} \"py7zr\"
↪ --no-build-isolation"
  ],
  "sources": [
    {
      "type": "file",
      "url": "https://files.pythonhosted.org/packages/2a/18/70c32fe9357f3ee
a18598b23aa9ed29b1711c3001835f7cf99a9818985d0/Brotli-1.0.9.zip",
      "sha256":
↪ "4d1b810aa0ed773f81dceda2cc7b403d01057458730e309856356d4ef4188438"
    },
    {
      "type": "file",
      "url": "https://files.pythonhosted.org/packages/14/6f/825068e229aaa59719d2
52d2279a23ba471f95d6df18ca09e9f7f445fda3/inflate64-0.3.1.tar.gz",
      "sha256":
↪ "b52dd8fefcd2ba179e5dfa18d6eca7e2fc822584616271c039d5ef1f9ca90c71c"
    },
    {
      "type": "file",
      "url":
↪ "https://files.pythonhosted.org/packages/22/31/ec5f46fd4c83185b806aa9c736
e228cb780f13990a9cf4da0beb70025fcc/multivolumefile-0.2.3-py3-none-any.whl",
      "sha256":
↪ "237f4353b60af1703087cf7725755a1f6fcaeeea48421e1896940cd1c920d678"
    },
    {
      "type": "file",
      "url":
↪ "https://files.pythonhosted.org/packages/3d/7d/d05864a69e452f003c0d77e728e
155a89a2a26b09e64860ddd70ad64fb26/psutil-5.9.4.tar.gz",
      "sha256":
↪ "3d7f9739eb435d4b1338944abe23f49584bde5395f27487d2ee25ad9a8774a62"
```

```

    },
    {
      "type": "file",
      "url":
↪ "https://files.pythonhosted.org/packages/1b/9c/514791abe077098ca6b4dac2ba
      8780c2f162aa35b5d99e48915a0f4ad2a8/py7zr-0.20.4-py3-none-any.whl",
      "sha256":
↪ "94d0c24217f6582741813ee94490a4ca82bd5f9bf35e4f8610cb588cf7445764"
    },
    {
      "type": "file",
      "url":
↪ "https://files.pythonhosted.org/packages/26/ff/25952179fa892e7d082a34e5
      917d747d05a227ba3c83dd3adac1c3f0be24/pybcj-1.0.1.tar.gz",
      "sha256":
↪ "8b682ed08caabfb7c042d4be083e28ddc692afb1deff5567111f8855071b75c3"
    },
    {
      "type": "file",
      "url": "https://files.pythonhosted.org/packages/3d/07/cfd8f52b906887780131
      7d26dc7225e19421bc659e1395d2cd6933b1a351/pycryptodomex-3.17.tar.gz",
      "sha256":
↪ "0af93aad8d62e810247beedef0261c148790c52f3cd33643791cc6396dd217c1"
    },
    {
      "type": "file",
      "url":
↪ "https://files.pythonhosted.org/packages/49/93/e45d3f8d2725ec448b9178
      b91791c2509d3a9b42d8984a22f576fe2f89be/pyppmd-1.0.0.tar.gz",
      "sha256":
↪ "075c9bd297e3b0a87dd7aeabca7fee668218acbe69ecc1c6511064558de8840f"
    },
    {
      "type": "file",
      "url": "https://files.pythonhosted.org/packages/b0/d8/75d91aa30bac7fa5b88f
      7216768c932b8b7ad48ccff1d2e322655c0571cd/pyzstd-0.15.4.tar.gz",
      "sha256":
↪ "de07ac54f57642f186732075cdce2be3d4a30228c3b17a6d8c6053765dc6eec8"
    },
    {
      "type": "file",
      "url": "https://files.pythonhosted.org/packages/ba/a7/2c12b543f853dae8862
↪ 86b824200eb9d7cd2466e3d14eff1799fbe8223b9/texttable-1.6.7-py2.py3-none-any.whl",
      "sha256":
↪ "b7b68139aa8a6339d2c320ca8b1dc42d13a7831a346b446cb9eb385f0c76310c"

```

```
    }
  ]
}
```

4.1.4 git.json

This module, while not used in the base manifest, serves as an implementation of adding build tools required for runtime dependencies such as git, autoconf, make, etc.

```
{
  "name": "git",
  "make-args": [
    "INSTALL_SYMLINKS=1"
  ],
  "make-install-args": [
    "INSTALL_SYMLINKS=1"
  ],
  "cleanup": [
    "/lib/pkgconfig",
    "/man"
  ],
  "sources": [
    {
      "type": "archive",
      "url":
↪ "https://www.kernel.org/pub/software/scm/git/git-2.40.1.tar.xz",
      "sha256":
↪ "4893b8b98eefc9fdc4b0e7ca249e340004faa7804a433d17429e311e1fef21d2",
      "x-checker-data": {
        "type": "anitya",
        "project-id": 5350,
        "stable-only": true,
        "url-template":
↪ "https://www.kernel.org/pub/software/scm/git/git-$version.tar.xz"
      }
    }
  ]
}
```

4.2 Main Program: FOSSEE_Inst_test.py

The application allows the user to install Pip packages and Verilator. It includes a logo, a select package feature, and three buttons: Install Pip package, Install Verilator from Archive, and Install Verilator from SRC. Some of the functions it performs are listed below.

- When the Install Pip package button is clicked, the selected Pip package is installed using the flatpack-spawn command and pip. If successful, a message box pops up showing that the package was installed successfully. If there is an error, a warning message box is shown.
- When the Install Verilator from Archive button is clicked, Verilator is installed from a 7z archive file using pkexec and cp commands. If successful, a message box pops up showing that Verilator was installed successfully. If there is an error, a warning message box is shown.
- When the Install Verilator from SRC button is clicked, the user can see the progress of Verilator installation from the source code through the InstallThread_Source class that signals progress through the output_signal pyqtSignal.
- The main function shows the GUI.

4.2.1 class AppWindow(...):

The class `AppWindow` initializes several UI elements and assigns them to instance variables. It adds a `QLabel` object named `label` and a `QComboBox` object named `combo`. And then it also adds three push buttons named `install_pkg_button`, `install_verilator_from_Archive_button`, and `install_verilator_SRC_button`. For each button, a corresponding method is connected to its clicked signal.

Lastly, it adds a `QPlainTextEdit` object named `Install_Output` and creates an instance of a custom thread class `InstallThread_SRC` and connects its `output_signal` to the `update_output` method. Finally, it creates a `QVBoxLayout` and adds all elements to it before setting the layout for the `AppWindow`.

```
import sys
import subprocess
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QMessageBox
import PyQt5.QtCore
from PyQt5.QtCore import *
from PyQt5.QtGui import QPixmap
from PyQt5.QtWidgets import QLabel
import py7zr
```

```

import os

class AppWindow(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.Logo = QLabel(self)
        self.pixmap = QPixmap('/app/assets/Fossee_logo.png')
        self.Logo.setPixmap(self.pixmap)
        self.Logo.setAlignment(PyQt5.QtCore.Qt.AlignRight)

        self.label = QtWidgets.QLabel('Select package to install:')
        self.combo = QtWidgets.QComboBox()
        self.combo.addItem(['makerchip-app', 'sandpiper-saas', 'matplotlib'])

        self.install_pkg_button = QtWidgets.QPushButton('Install Pip package')
        self.install_pkg_button.clicked.connect(self.install_Pip_package)

        self.install_verilator_from_Archive_button =
        ↪ QtWidgets.QPushButton('Install Verilator from Archive')
        self.install_verilator_from_Archive_button.clicked.
        ↪ connect(self.intall_Verilator_from_Archive)

        self.install_verilator_SRC_button = QtWidgets.QPushButton('Install
        ↪ Verilator from SRC')
        self.install_verilator_SRC_button.clicked.
        connect(self.install_Verilator_SRC)

        self.Install_Output = QtWidgets.QPlainTextEdit()

        self.install_thread = InstallThread_SRC()
        self.install_thread.output_signal.connect(self.update_output)

        layout = QtWidgets.QVBoxLayout()
        layout.addWidget(self.Logo)
        layout.addWidget(self.label)
        layout.addWidget(self.combo)
        layout.addWidget(self.install_pkg_button)
        layout.addWidget(self.install_verilator_from_Archive_button)
        layout.addWidget(self.install_verilator_SRC_button)
        layout.addWidget(self.Install_Output)
        self.setLayout(layout)

```

4.2.2 install_Pip_package():

This function retrieves the currently selected item in a combo box. It then attempts to call subprocess to execute the command `flatpak-spawn --host sys.executable -m pip install <package>` where `<package>` is the selected item from the combo box. If this command executes successfully, a message box and plain text output inform the user that `<package>` was installed successfully. If an exception is thrown during execution, a message box and plain text output inform the user that an error has occurred. The exception is also printed to the plain text output.

```
def install_Pip_package(self):
    package = self.combo.currentText()
    try:
        subprocess.check_call(['flatpak-spawn', '--host', sys.executable,
                               ↪ '-m', 'pip', 'install', package])
        QtWidgets.QMessageBox.information(self, 'Success', f'{package}
        ↪ installed successfully')
        self.Install_Output.appendPlainText(f'Success: {package} installed
        ↪ successfully')
    except Exception as e:
        QMessageBox.warning(self, "ERROR", str(e))
        self.Install_Output.appendPlainText("[ERROR!]: ")
        self.Install_Output.appendPlainText(str(e))
```

4.2.3 install_Verilator_from_Archive():

This function installs Verilator from the archive file "verilator.7z", stored in the "/app/assets/packages/" directory (Inside sandboxed filesystem). The output directory is set as the current working directory using Python's `os.getcwd()` function, and the source file is set to "output_dir/verilator/bin/verilator".

The destination directory is set to "/usr/bin/" which exists in the User's filesystem. The `py7zr.SevenZipFile` module extracts the archive file using the `extractall()` function.

If the archive extraction is successful, the `subprocess.check_call(...)` function copies the `source_file` to the `destination_dir`. If that fails, a warning is issued, and the error message is added to the `Install_Output` plain text field in the GUI.

```
def intall_Verilator_from_Archive(self):
    file_name = "/app/assets/packages/verilator.7z"
    output_dir = os.getcwd()
```

```

source_file = f"{output_dir}/verilator/bin/verilator"
destination_dir = "/usr/bin/"

with py7zr.SevenZipFile(file_name, mode='r') as z:
    z.extractall(path=output_dir)

try:
    subprocess.check_call(['flatpak-spawn', '--host', "pkexec", "cp",
        ↪ source_file, destination_dir])
    QtWidgets.QMessageBox.information(self, 'Success', 'Verilator
        ↪ installed successfully from Archive')
    self.Install_Output.appendPlainText('[Success]: verilator installed
        ↪ successfully from Archive')
except subprocess.CalledProcessError as e:
    QMessageBox.warning(self, "ERROR", str(e))
    self.Install_Output.appendPlainText("[ERROR!]: ")
    self.Install_Output.appendPlainText(str(e))

## Used to show updates for install_Verilator_SRC
def update_output(self, text):
    self.Install_Output.appendPlainText(text)

```

4.2.4 install_Verilator_SRC():

This function starts a QThread subprocess (multi-threaded execution) to run the installation script for verilator. The "Install_verilator.sh" bash script executes with the necessary arguments required to install the compiled binary in the user file system. It will be reviewed in the later sections of the report.

```

def install_Verilator_SRC(self):
    try:
        self.Install_Output.clear()

        self.Install_Output.clear()
        self.install_thread.start()

    except Exception as e:
        self.Install_Output.appendPlainText("[ERROR!]: ")
        self.Install_Output.appendPlainText(str(e))
        QtWidgets.QMessageBox.information(self, "Error", str(e))
        print("[ERROR!]: ", e)

```

4.2.5 QThread classes:

This code defines two QThread classes `install_verilator` and `InstallThread_SRC` required for the installation of Verilator.

`install_verilator` extends QThread and defines three methods - `__init__()`, `start()` and `run()`. `__init__()` initializes the command required for Verilator installation, `start()` starts the thread execution and `run()` performs the actual installation by running the specified command.

`InstallThread_SRC` also extends QThread and defines only the `run()` method. It initializes the command required for Verilator installation and executes it using `subprocess.Popen()`. It reads the command output until there is no more output left to read. If the installation succeeds, [Info] is emitted via `output_signal` and printed to the console. Else, [ERROR] is emitted via `output_signal` and printed to the console.

```
class install_verilator(QThread):
    P_progress = pyqtSignal(str)
    def __init__(self, command, parent=None):
        QThread.__init__(self, parent)
        self.command = command
    def start(self):
        QThread.start(self)
    def run(self):
        try:
            p = subprocess.run(str(self.command), shell=True, capture_output=True)
            status = p.stdout.decode()
            if p:
                print(status)
                self.P_progress.emit(status)
        except Exception as e:
            self.Install_Output.appendPlainText("[ERROR!]: ")
            self.Install_Output.appendPlainText(str(e))
            QMessageBox.warning(None, "Invalid Install Commands!", "Error in
            ↪ install_verilator")

class InstallThread_SRC(QThread):
    output_signal = pyqtSignal(str)

    def __init__(self):
        super().__init__()

    def run(self):
        command = "/app/assets/scripts/Install_verilator.sh"
```

```

process = subprocess.Popen(["bash", command], stdout=subprocess.PIPE,
→ stderr=subprocess.STDOUT)

while True:
    output = process.stdout.readline()
    if output == b'' and process.poll() is not None:
        break
    if output:
        self.output_signal.emit(str(output.strip())[2:-1])
        print(output.strip()[2:-1])

rc = process.poll()
if rc == 0:
    self.output_signal.emit("[Info] Installation succeeded!")
    print("Installation succeeded!")
else:
    self.output_signal.emit("[ERROR] Installation failed!")
    print("Installation failed!")

```

4.2.6 Main Function:

The main function checks whether the current file is being run as the main program. If it is, then it creates a QApplication instance, initializes an instance of AppWindow, shows the AppWindow and starts the application event loop by calling "app.exec_()" which will wait for events to occur.

```

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    ex = AppWindow()
    ex.show()
    sys.exit(app.exec_())

```

4.3 Install Script: Install_verilator.sh

The install script clones the Verilator repository from Github, switches to the specified version, partially builds it using autoconf in the flatpak file system and installs it using make to the user file system. It asks for the root password to execute some commands with elevated privileges.[8] The `install_verilator.sh` script does the following:

- Clones the Verilator repository from <https://github.com/verilator/verilator>
- Changes directory to `verilator/`
- Checks out version 4.106

- Builds Verilator using autoconf and configure
- Asks for the root password to install make, make install and finally displays the version of Verilator installed.
- Cleans up by removing the cloned directory using flatpak-spawn and `rm -rf./verilator`

```
#!/bin/bash

## Clone Verilator repository

echo '# Please wait... cloning https://github.com/verilator/verilator'
git clone http://git.veripool.org/git/verilator
echo '# repo cloned...'
echo "=====\\n"

echo '# changing dir to verilator/'
cd ./verilator

echo '# current working directory is: '
pwd
echo "=====\\n"

# Switch to version 4.106
git checkout v4.106

# Build Verilator
autoconf

echo '# please wait... This may take some time'
echo "=====\\n"
./configure --prefix=/usr --libdir=/usr/lib64

echo '# current working directory is (supposed to be verilator): '
pwd
echo "======"
echo "Please enter root password:"
flatpak-spawn --host sudo -S make

echo 'Installing make...'
echo "======"
echo "Please enter root password:"
flatpak-spawn --host sudo -S make install
```

```
flatpak-spawn --host verilator --version
```

```
echo "=====\\n"
```

```
echo "Cleaning up..."
```

```
flatpak-spawn --host rm -rf ./verilator
```

4.4 Testing script: test

This script is used to verify if the necessary build tools are available in the runtime of the Flatpak application.

```
#!/bin/sh
```

```
echo "This is a test script"
```

```
echo "====="
```

```
git --version
```

```
echo "====="
```

```
make --version
```

```
echo "====="
```

```
autoconf --version
```

```
echo "====="
```

Chapter 5

Results

5.1 GUI & Launching procedure

The application can be launched via the terminal using the command:

```
flatpak run org.flatpak.FOSSEE_Inst_test
```

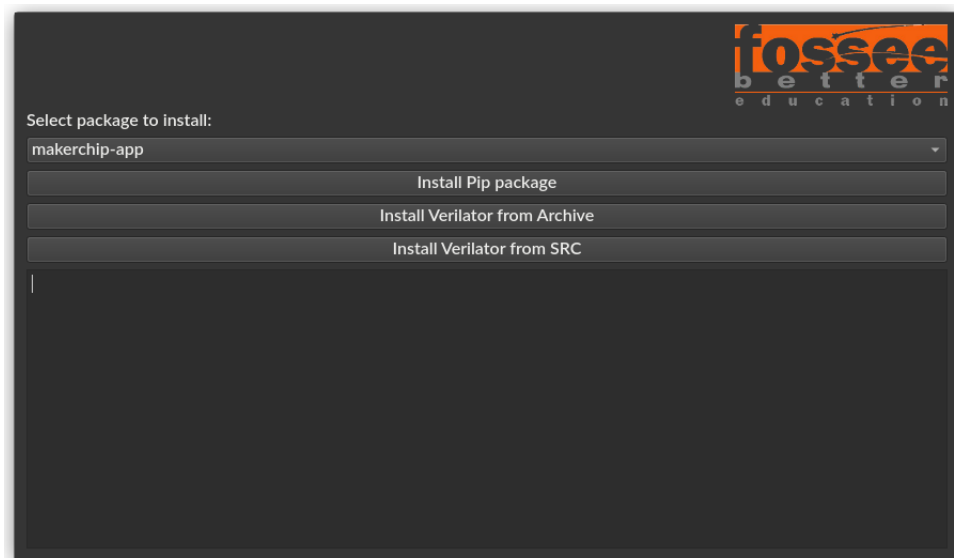


Figure 5.1: GUI of the Application

5.2 Installing Python Dependencies for eSim:

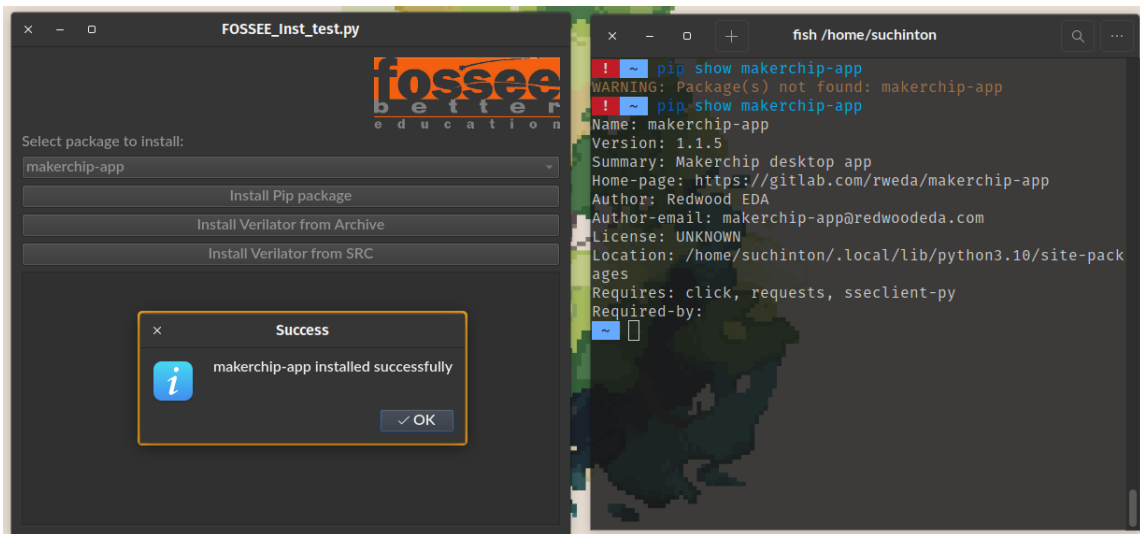


Figure 5.2: Installing Python Packages using pip(3)

5.3 Installing Verilator from Archived package(Pre-built):

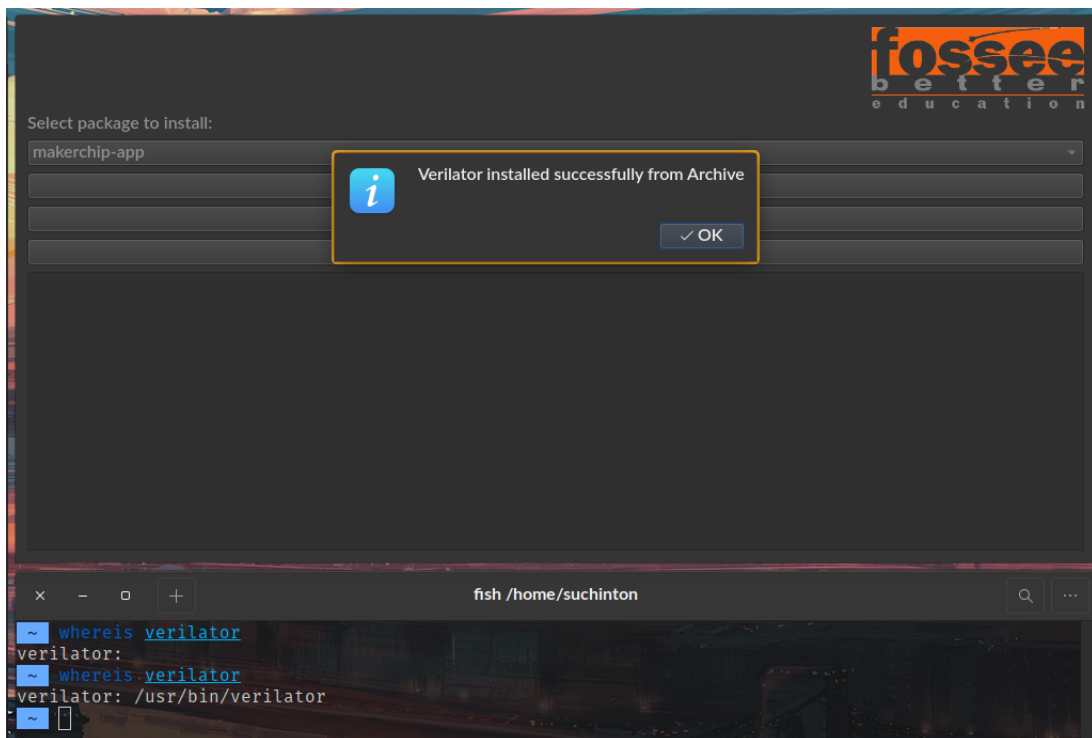
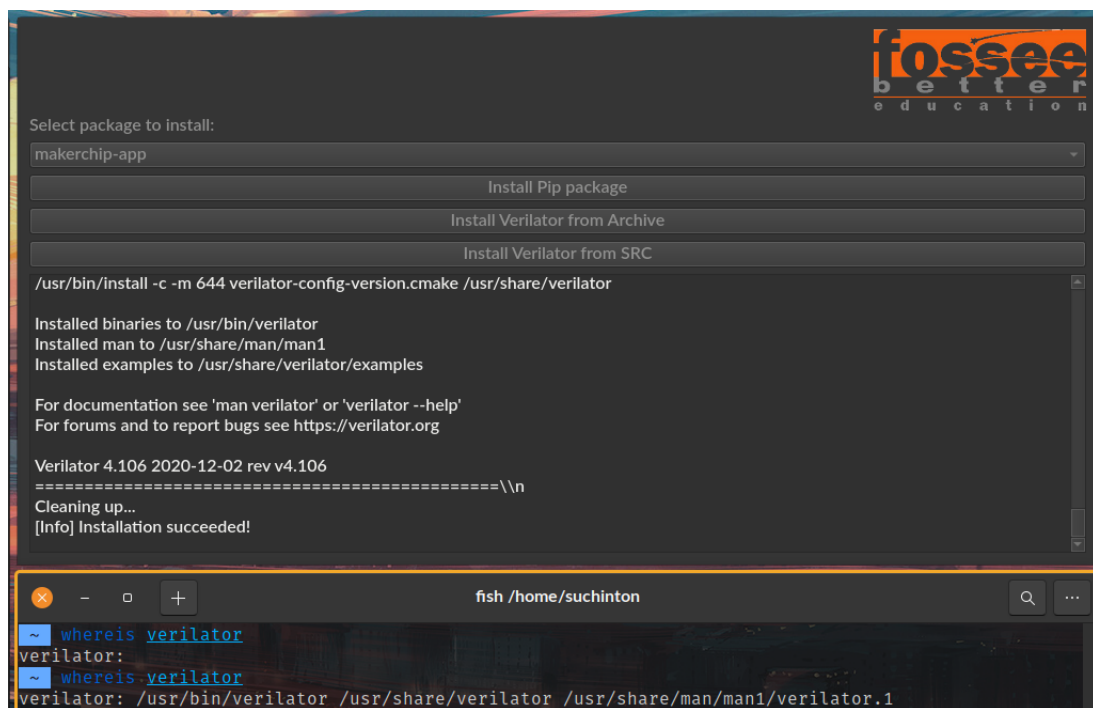


Figure 5.3: Verilator install; Archived package

5.4 Installing Verilator from Source:



```
Select package to install:
makerchip-app
Install Pip package
Install Verilator from Archive
Install Verilator from SRC
/usr/bin/install -c -m 644 verilator-config-version.cmake /usr/share/verilator
Installed binaries to /usr/bin/verilator
Installed man to /usr/share/man/man1
Installed examples to /usr/share/verilator/examples
For documentation see 'man verilator' or 'verilator --help'
For forums and to report bugs see https://verilator.org
Verilator 4.106 2020-12-02 rev v4.106
=====\\n
Cleaning up...
[Info] Installation succeeded!

fish /home/suchinton
~ whereis verilator
verilator:
~ whereis verilator
verilator: /usr/bin/verilator /usr/share/verilator /usr/share/man/man1/verilator.1
```

Figure 5.4: Installing Verilator from Source

For more information, please refer to the following GitHub Repository:

<https://github.com/suchinton/eSim/tree/installers>

Chapter 6

Workflow

To start building and installing the package, we need to first install flatpak, the KDE runtime and the PyQt5 Base App required for running and packing our Qt5 App.

```
# NOTE: Equivalent commands can be used to install these packages, refer to
→ https://flatpak.org/setup/
# Install flatpak/ flatpak-builder on Ubuntu (or Ubuntu-based distros)
$ sudo apt install flatpak flatpak-builder -y

# for installing the KDE SDK and Runtime
$ flatpak install org.kde.Sdk//5.15-22.08

# for installing the Base App
$ flatpak install com.riverbankcomputing.PyQt.BaseApp//5.15-22.08
```

6.1 build_new_flatpak.sh

Relevant commands can be found by running the `build_new_flatpak.sh` script found in the `scripts/` directory.

```
#!/bin/sh

echo "Reference Commands:"
echo "===== "
echo "To build, run command:"
echo "  $ flatpak-builder --force-clean build-dir
→ org.flatpak.FOSSEE_Inst_test.yml"
echo "===== "
echo "To install the app, run command:"
```

```

echo " $ flatpak-builder --user --install --force-clean build-dir
↳ org.flatpak.FOSSEE_Inst_test.ym"
echo "=====
echo "To run application, run command:"
echo ' $ flatpak run org.flatpak.FOSSEE_Inst_test to run app'
echo "=====
echo "To resolve new pip dependencies, run command:"
echo ' $ python3 flatpak-pip-generator
↳ --requirements-file='$HOME/Repos/Test/req' --output pypi-dependencies --yaml'
echo "=====

```

Note: Build tools are required on the host machine as well to complete the installation of packages in the correct directories.

6.2 flatpak-pip-generator

This Python script, readily made available by the Flatpak Community[1], helps generate Flatpak manifests for Python packages. It script helps automate the process of generating Flatpak manifests for Python packages by fetching the required source packages, extracting metadata, and generating the necessary dependency information. Here is a brief overview of the script's purpose and functionality:

- It imports necessary modules and sets up command-line argument parsing using the argparse module.
- The script reads the requirements file or the list of packages specified in the arguments.
- It determines the output file name based on the options and packages specified. The script downloads the required source packages using pip download command and saves them to a temporary directory.
- The script generates the dependencies for each package, including recursively downloading the packages to list their dependencies.
- It generates the Flatpak manifest data in JSON or YAML format, including information about the packages, their sources, and dependencies.

Chapter 7

Conclusion and Future Scope

7.1 Conclusion

We were able to successfully implement a fully responsive, proof-of-concept application, which can resolve the various dependencies of eSim and make it a Distro-Agnostic tool. We were also able to successfully utilize the required resources from the user space while keeping the demo application and its runtime dependencies sandboxed.

7.2 Future Work

- Runtime dependencies like build-essentials, git, bison, etc can be added as separate modules in the Flatpak Manifests to reduce the overall package size and reduce its redundant libraries and binaries which are not never utilized.
- Only the GUI components could be packaged in the Flatpak package, making additional dependencies optional (or) bundled in the runtime during the build process of the application.
- Resources that need to compile on the user system can take up a lot of time to build, therefore packages required by eSim could be self-hosted and pulled during runtime, for improved installation times and overall safety of the application.
- Lastly the Application could be hosted on Flathub, which is a centralized repository for hosting and distributing Linux applications. Hosting the application on such a platform provides easy access, installation, and updates for users on various Linux distributions.

Bibliography

- [1] Flatpak Official Documentation. 2022.
URL: <https://docs.flatpak.org/en/latest/>

- [2] Github Official Website
URL: <https://github.com/FOSSEE/eSim/>

- [3] Wikipedia Official Website. 2023.
URL: <https://en.wikipedia.org/wiki/Flatpak>

- [4] eSim Official website. 2023.
URL: <https://esim.fossee.in/>

- [5] Python Official Website. 2023.
URL: <https://www.python.org/>

- [6] Linux Official Website. 2023.
URL: <https://www.linux.org/>

- [7] Riverbank Computing Official Website. 2023.
URL: <https://riverbankcomputing.com/software/pyqt/>

- [8] Veripool, Verilator Official website. 2023.
URL: <https://www.veripool.org/verilator/>