

Packaging

How to package

First and foremost clone the **verilator** github repo and compile the program's stable version. Follow the link for instructions.

Once the repo is cloned **cd** into the repo and set up a virtual environment

```
cd linux_package_management_system # The repo
python3 -m venv env
source env/bin/activate
pip install -r requirements.txt
```

Once the virtual environment is set move into the **src** folder use **cx_freeze** to make our program a stand alone executable.

```
cd src
cxfreeze -c main.py --target-dir=dist
```

Now from the directory where verilator was built, copy the **bin** and **include** directories into the **dist** directory which was created using **cx_freeze**.

After copy **bin** and **include** into the **dist** directory compress it using **tar**

```
tar cJvf program.tar.xz dist
```

Now move back to the root directory of the repo and use the template shell script stored in there to create the required installer

```
cd .. #cd to the repo
cp installer_template_script.sh installer.sh
```

This **installer.sh** (or **installer_template_script.sh**) has an bash script written in it.

It has a marker called **__ARCHIVE__** to which we will soon mark our compressed tar file using **cat** whose line number from which it is stored would be stored in a variable named **ARCHIVE** using some **awk** and using it the actual file would read using **tail** and piping its output to **tar** to extract the file into a directory which we mention inside the **sh** file (Here it is **\${HOME}/verilator_gui**)

Now we concatenate the compressed **tar.xz** file to **installer.sh** and make it executable. **cat src/program.tar.xz >> installer.sh** **chmod +x installer.sh** This is the packaging process of the installer.

installer_template_script.sh

```
#!/bin/bash
```

```
echo ""
echo "Verilog GUI Installer"
echo ""
```

```

# Create destination folder
DESTINATION="${HOME}/verilator_gui"
mkdir -p ${DESTINATION}

# Find __ARCHIVE__ maker, read archive content and decompress it
ARCHIVE=$(awk '/^__ARCHIVE__/ {print NR + 1; exit 0; }' "${0}")
tail -n+${ARCHIVE} "${0}" | tar xpv -C ${DESTINATION}

# Put your logic here (if you need)

chmod +x ${HOME}/verilator_gui/dist/bin/*
export VERILATOR_ROOT=${HOME}/verilator_gui/dist
export PATH=$VERILATOR_ROOT/bin:$PATH

echo ""
echo "Installation complete."
echo ""

# Exit from the script with success (0)
exit 0

__ARCHIVE__

```

Reason for the packaging approach

After doing some investigation, I discovered that **cx_freeze** can create a standalone Python executable. I tested its performance by installing it on an Ubuntu 22.04 and running it on an MX Linux, and I was really pleased with its output. In addition, the produced executable resembled a straight executable and was independent of any distribution. After going through its documentation for some time, I gained enough faith in the **cx_freeze**'s cross-platform execution. Then, I desired a unique kind of packaging that was independent of a particular distro. Actually, I was looking for a packaging technique that would allow me to give my final product a **.run** extension. In , I wasn't sure whether I needed to make separate **.deb** or **.rpm** packages in addition to the **.run**. But fortunately, I stumbled into this website, which I adored, by accident. The link for the website is:

<https://tinyurl.com/2p9x4djp>

I was able to create a **.sh** file using the website's script that contained both the installer script and the whole application in a compressed format.

Once I noticed this was possible I copied the **bin** and **include** directory from the **verilator** build and pasted it in the **dist** directory which was created by **cx_freeze** on making the python program executable(The executable is located

in a directory that we named `dist`). Then the file was compressed into a `tar.xz` file.

Now I had to edit the installer shell script found in the previously mentioned website. So, I did the following with the file.

- Changed the installation directory to `${HOME}/verilator_gui`
- changed permission for all files inside `${HOME}/verilator_gui/dist/bin` as executable
- Set the path variable for verilator for a proper installation

Now the installer script was only missing the actual software which was in the `tar.gz` file which is concatenated to the file via

```
cat program.tar.gz >> installer.sh
```

A note on dependency

Here I have compiled `verilator` in my computer and is using an OS-distributed `Verilator` in the program.

While compiling I've used `GLIBC_2.31` which was released in 2020. This enables me to run the program with later versions of `GLIBC` as well.

Installation

Download the `installer.sh` file.

You can do this either by cloning the repo or directly downloading it from Google Drive.

https://drive.google.com/file/d/1HVtLxvJNPOGXQtFQGIPhE6OVPPJTsd62/view?usp=share_link

Note

An alternate download link is provided because the `installer.sh` is somewhat of a big file and GitHub doesn't allow files above 100 MiB

So, the file was uploaded using `git-lfs` and hence has this bandwidth restriction of 1 GiB. So, once the 1 GiB monthly bandwidth is used, downloading the file would not be possible. This Google Drive link was provided to ensure that the `installer.sh` file would be accessible even amidst a wear out of the monthly free bandwidth that `git-lfs` provides.

Once downloaded open a terminal in the directory in which the installer file has been downloaded.

Then:

```
chmod +x installer.sh
./installer.sh
```

This will now install the program in a folder named `verilator_gui` inside your `home` directory.

The program is accessible from the `dist` directory inside the `verilator_gui` directory

```
cd ~/verilator_gui/dist
```

One could now simply open up the program by

```
./main
```

Dependencies

The software Verilator does comes packaged inside this program and is installed when running the `installer.sh`.

The stable version inside the Verilator GitHub repo was compiled along with a `GLIBC 2.31` so that it would support a wide variety of distros especially the Debian based ones that still hasn't moved on to the later versions such as `GLIBC 2.35` or `GLIBC 2.34`. Additional dependency warnings if any will be displayed in the dialogue box that opens up once the `Build` button is pressed inside the program.

Usage

Once properly installed open the program from the directory at which the program was installed.

```
cd ~/verilator_gui/dist
./main
```

Now, the GUI build using `PyQt5` will open up.

In the program there will be 3 text fields associated with each, a button as well. Along with the first field is a button with `Input` written inside it.

Clicking that button will open a dialogue box that helps to pick a verilog file.

Along with the second field is a button with `Input` written inside it.

Clicking that button will open a dialogue box that helps to pick a wrapper file.

Along with the first field is a button with `Output` written inside it.

Clicking that button will open a dialogue box that helps to pick a directory to where the compiled simulation is to be saved.

Inside the directory mentioned in the `Output` field there will be a directory named `obj_dir` inside which the simulation will be present.

It can be executed as follows:

```
cd <output dir>
obj_dir/Vour
```

Along with this there is a radio button that helps to pick whether to use a `C++` execution or a `SystemC` execution.

The spinbox helps to set the number of cores that will go into the make command for compilation.