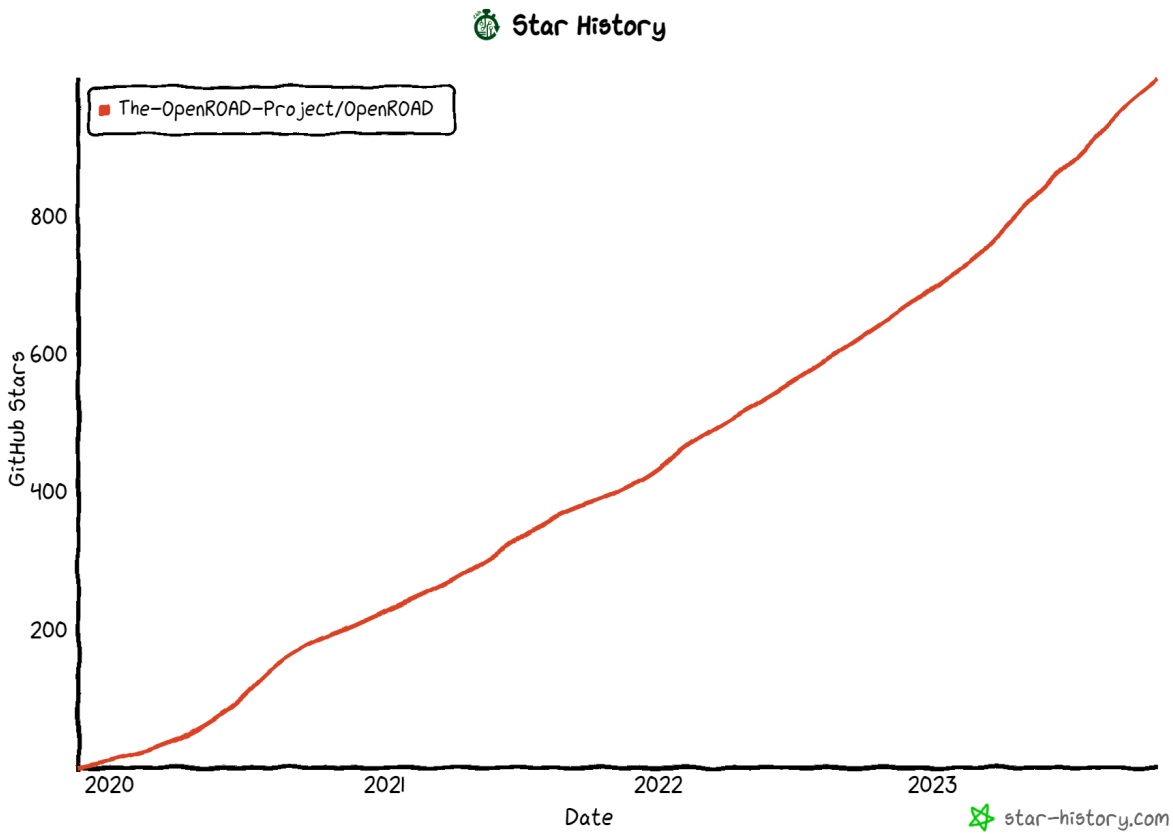
OpenROAD Writeup for final check

Introduction

Welcome to the final blog post for my GSoC'23! Once again, my name is Jack and I am working under the open-source electronic design automation project - OpenROAD. We are a fast growing leading open-source foundational application for semiconductor digital design, as evidenced from our consistent star growth since inception. You may check us out at this link. Allow me to share the four significant milestones of this project.



Improving Ease of Installation

Firstly, OpenROAD is now able to support multiple operating systems. This is essential as one of our primary goals is to democratise chip implementation. And installation is often one of the hardest steps to get right, so that was one of our priorities. Today, we provide options for different types of installation:

- *Prebuilt binaries*: Local installations can often be riddled with incompatibilities or unexpected bugs, as well as taking a long compilation time. We sidestepped this by providing semi-regular updates to OpenROAD binary, reducing the time to installation.
- *Docker*: Echoing previous concerns, we also enabled Docker installation for 9 major operating systems. Docker is extremely flexible and runs on many operating systems (as long as it is supported by Docker).

With these changes, we have observed a 10% reduction of installation related Github issues posted on a weekly basis.

Operating System	Local Installation	Prebuilt Binaries	Docker Installation	Windows Subsystem for Linux
Ubuntu 20.04	Y	Y	Y	-
Ubuntu 22.04	Y	Y	Y	-
CentOS 7	Y	-	Y	-
CentOS 8	Y	-	Y	-
Debian 10	Y	Y	Y	-
Debian 11	Y	Y	Y	-
RHEL	Y	-	Y	-
Windows 10 and above	-	-	Y	Y
macOS	Y*	-	Y	-

Filling missing documentation

Next, we have made considerable improvements to over 20 tool-specific documentations, introducing consistent formatting styles for each page. We introduce default values and datatypes to allow users to use the tools with greater ease.

Options

Switch Name	Description
`-output_maze`	Path to output maze log file (e.g. `output_maze.log`).
`-output_drc`	Path to output DRC report file (e.g. `output_drc.rpt`).
`-output_cmap`	Path to output congestion map file (e.g. `output.cmap`).
`-output_guide_coverage`	Path to output guide coverage file (e.g. `sample_coverage.csv`).
`-drc_report_iter_step`	Report DRC on each iteration which is a multiple of this step. The default value is `0`, and the allowed values are integers `[0, MAX_INT]`.

Rather than having all arguments for a function under a common table, we separated out into developer arguments and developer commands. This is to further make our documentation more beginner-friendly to read, while not alienating our technical userbase. We have also added sections for example scripts and regression test, so as to help onboard newcomers to each tool of the flow.

Useful Developer Commands

If you are a developer, you might find these useful. More details can be found in the source file or the swig file.

Command Name	Description
`find_net`	Get a reference to net name.

Extensible documentation framework

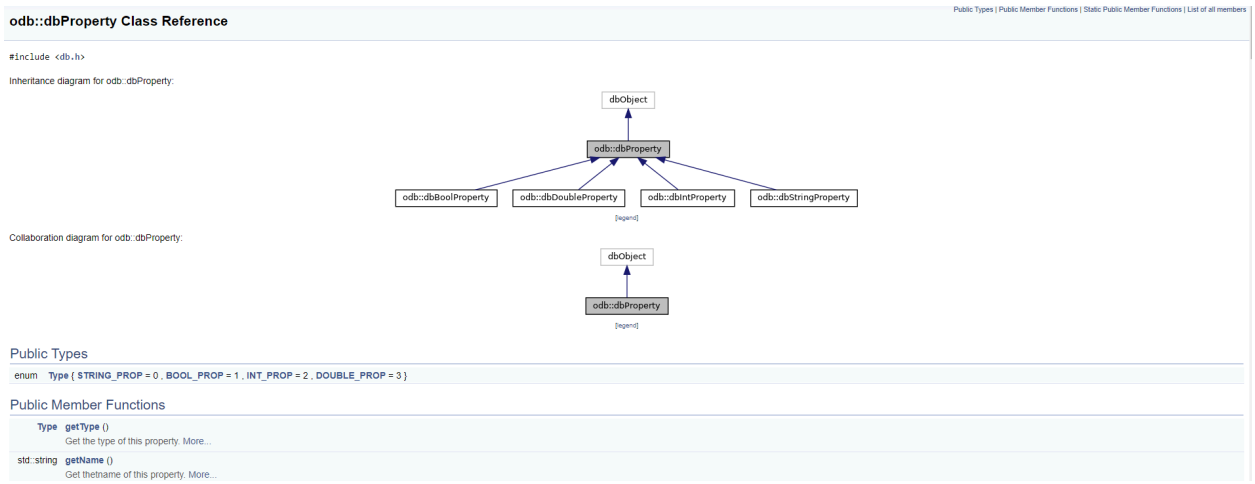
Thirdly, we have introduced extensible documentation frameworks. Now, what do we mean by "extensible"? It means we have created an infrastructure which is easy to use for developers, and allows for greater maintanability. Our goal is to create something that requires minimal changes to add content for documentation.

So how did we do this?

We introduced 4 initiatives, namely: the warning/error messages glossary. We noticed that people were searching for error and warning messages, but our documentation did not have them. So we added a page where all the error/warning messages along with relevant code line number can be generated automatically. On top of that, developers can add useful debug information to help the end user.

GRT	0111	FastRoute.cpp:1303	INFO	-
GRT	0112	FastRoute.cpp:1304	INFO	-
GRT	0113	FastRoute.cpp:381	WARN	-
GRT	0114	FastRoute.cpp:411	WARN	-
GRT	0115	GlobalRouter.cpp:309	WARN	-
GRT	0118	GlobalRouter.cpp:300	ERROR	Do refer to the GUI guide and global routing debugging tips.
GRT	0119	GlobalRouter.cpp:293	ERROR	Do refer to the GUI guide and global routing debugging tips.
GRT	0122	RipUp.cpp:450	ERROR	-

Next, we also introduced automatically generated Doxygen pages, which integrates nicely into our C++/Tcl source code framework. This automatic generation will make it much more convenient for developers to just insert comments into their source code, and allow Doxygen to generate documentation automatically.



Next, we introduced cloud-based packaging. It is important that our framework is able to runnable on cloud, and the ever-popular notebook format. Our Colab based notebook was created with this in mind, and allows for easy transfer to other notebook providers with some modifications. Check out the notebooks here!

ORFS Tutorial [gcd]

- In this tutorial, we will walk through installation steps and get a minimal viable setup running.
- Due to Colab memory limits, we are limited to 12-13Gb of RAM. Thus, you may only perform small designs.
- However, if you have access to cloud providers, feel free to extend this notebook by changing the parameters around.

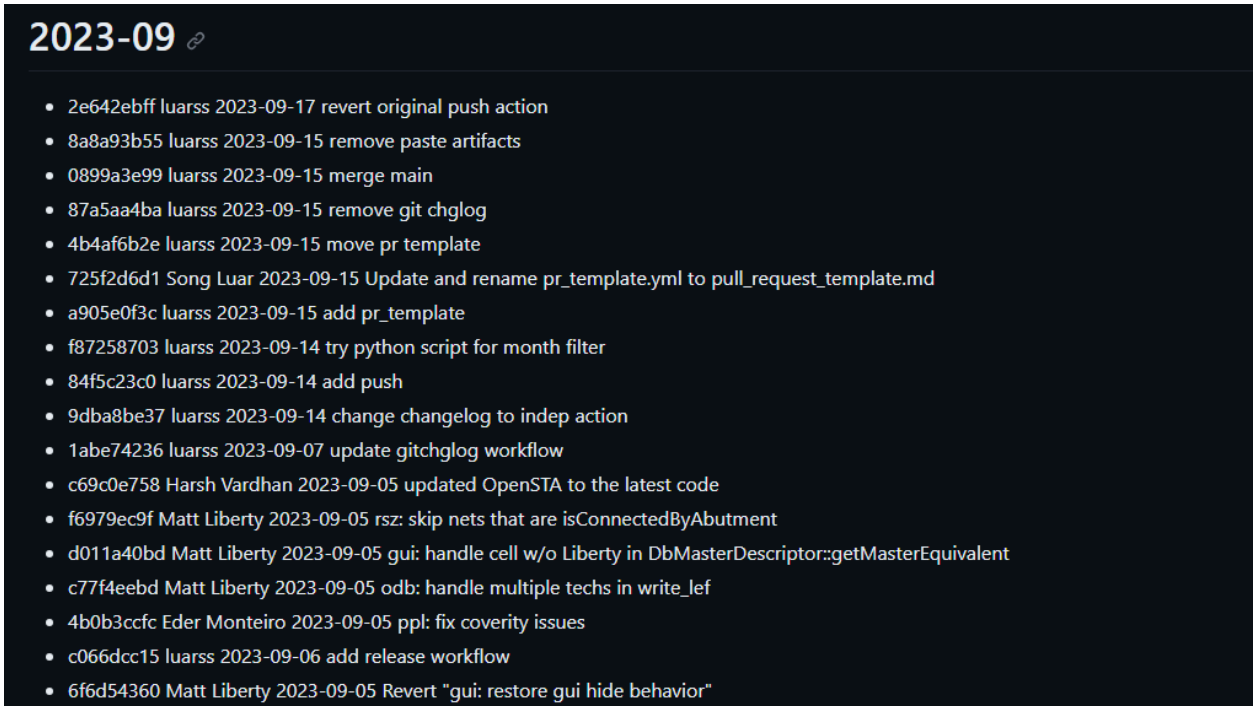
Acknowledgements

- Some of this tutorial is referenced from Chips Alliance's notebooks.

+ Cod

```
import os
import pathlib

!sudo apt-get update
!sudo apt-get install time

!curl -Ls https://micro.mamba.pm/api/micromamba/linux-64/latest | tar -xj bin/micromamba
conda_prefix_path = pathlib.Path('conda-env')
CONDA_PREFIX = str(conda_prefix_path.resolve())
!bin/micromamba create --yes --prefix $CONDA_PREFIX
!echo 'python ==3.7*' >> {CONDA_PREFIX}/conda-meta/pinned
!CI=0 bin/micromamba install --yes --prefix $CONDA_PREFIX \
                     --channel litex-hub \
                     --channel main \
                     openroad \
                     yosys \
                     klayout

!sudo pip install gdstk

PATH = os.environ['PATH']
%env CONDA_PREFIX={CONDA_PREFIX}
%env PATH={CONDA_PREFIX}/bin:{PATH}
```

Lastly, we have the changelog workflow which can be triggered manually. For our open-source project, we have chosen not to do software releases. This means it can be difficult to track the changes between commit numbers. Adding this workflow can help newcomers track the changes easier, by month.



OpenROAD Chatbot

Finally, we are also discussing the potential of creating a chatbot whose purpose is to answer user queries. We were thinking, there are lots of domain knowledge in Slack Channels, Github repos, and so on, so why not create a LLM-based chatbot. Stay tuned for updates!

Personal Reflections

To me, my most valuable takeaway is with regards to code quality. Often times, we as coders tend to opt for the best solution and “hack” something out quickly. Hacking is fine, as a proof of concept - but not for long term code development. Working in open-source projects like this, I have learnt to avoid creating unnecessary files, shortening the code and optimising runtime. In doing our job, we also wish to make life easier, not harder for future developers.

Final words

Thank you very much to my mentors Indira and Vitor for their guidance and insight throughout the project, as well as the OpenROAD dev team for their assistance. Would also like to thank the Google Summer of Code organising committee, and UCSC for creating such a wonderful program. Being able to contribute to actual real open-source projects with real needs, is truly the best of both worlds for aspiring programmers.