Frequently-Asked Questions (FAQ)

General

(faq-whats-openlane=)

What is OpenLane?

OpenLane is a piece of software for ASIC implementation, initially released by Efabless Corporation in 2020 for use with the Google OpenMPW program.

Version 1.0 of OpenLane is a simple but stable and battle-tested flow, primarily intended for implementing designs for Efabless's MPW programs.

Version 2.0 of OpenLane reimagines it as not just a single flow, but as an infrastructure by which flows can be implemented. An OpenLane 1-compatible flow named "Classic" exists, but is in beta pending further silicon validation.

Should I use OpenLane 1 or OpenLane 2?

As of the time of writing:

If you're targeting an Efabless MPW project such as chipIgnite, we currently exclusively recommend OpenLane 1 given its extensive use (and thus silicon-

validation) for such a purpose. While OpenLane 2 does include a flow similar to OpenLane 1.0 (named “Classic”,) that flow is awaiting further silicon validation and is still in beta and should only be used at the user’s own risk. In the future, OpenLane 2.0’s “Classic” flow will replace OpenLane 1 for this use-case.

For now however, if you’re an educator looking for an elegant notebook-based interface to introduce students to ASIC implementation or if you’re implementing a complex chip for which the OpenLane 1 flow is insufficiently flexible (e.g., you require custom implementation steps or ECOs), we recommend OpenLane 2.

How is OpenLane different from OpenROAD?

OpenROAD is one of many utilities used by OpenLane, which integrates it and many other tools in order to achieve a full RTL-to-GDSII flow.

OpenROAD is primarily developed by The OpenROAD Project, which involves many corporations and academic institutions (primarily the University of California, San Diego, Parallax Software, and Precision Innovations). OpenLane, on the other hand, is primarily developed by Efabless Corporation.

The two projects are affiliated but are otherwise distinct and are managed by different teams.

Can I use OpenLane with my (company’s) proprietary PDK?

In general, yes, but you would have to create OpenLane configuration files for said PDK. See Using non-Volare PDKs for more info.

Is OpenLane silicon-proven?

OpenLane 1.0 has been used for countless verified tapeouts, including more or less every open-source design on the Google MPW shuttles.

OpenLane 2.0 is relatively less battle-tested but was used for TinyTapeout 3.5.

Why should I use OpenLane over other open-source RTL-to-GDS-II flows?

Point of Comparison	OpenROAD Flow Scripts	SiliconCompiler	OpenLane <2.0	OpenLane ≥2
Architecture	Monolithic	Plugin-based	Monolithic	Plugin-based
Configuration	Tcl Files	Python Files	JSON/Tcl Files	JSON/Tcl/Python Files
Programming Language	GNU Make	Python	Tcl	Type-checked Python
Maintainer	The OpenROAD Project	ZeroASIC	Efabless	Efabless
Dependencies	Separate (Build Scripts)	Separate (Build Scripts)	Bundled	Bundled
Cloud Service	No	Yes	No	No (Planned)
Proprietary Tool Support	No	Yes	No	Yes (with Plugins)
Pre-built Binaries	`.deb` (x86-64)	N/A	Docker (x86-64, ARM64)	* Natively through Nix: Linux and macOS (x86-64, ARM64) * Docker (x86-64, ARM64)
Open-Source PDK Support	`sky130`, `gf180mcu`, `nangate45`, `asap7`	`sky130`, `gf180mcu`, `asap7`	`sky130`, `gf180mcu`	`sky130`, `gf180mcu`
Community Examples	Limited	Limited	[illegible] project	[illegible] OL1 Examples

Point of Comparison	OpenROAD Flow Scripts	SiliconCompiler	OpenLane <2.0	OpenLane ≥2
			wafer shuttles	

Setup

Why does running OpenLane on Windows require the Windows Subsystem for Linux (WSL)?

In short, a lot of the open-source EDA tools OpenLane relies on presume a Linux-based environment, so they would be non-trivial to port to Windows as we'd have to make sure every tool both compiles *and* behaves as expected on Windows.

Why do you use Nix?

Nix allows us to create a near-perfectly reproducible environment on macOS and all Linux distributions with just a single set of scripts, and the rich community ecosystem surrounding it also enables us to distribute these environments in their entirety to end-users.

Similar to Docker, this mostly eliminates variables related to the user's environment, although unlike Docker, it maintains integration with the user's filesystem, doesn't add a virtualization penalty on macOS, and does not require the entire image to be re-downloaded every time an update occurs.