Bridging Academic Open-Source EDA to Real-World Usability

Invited Paper

Austin Rovinski
rovinski@umich.edu
University of Michigan
Ann Arbor, Michigan

Tutu Ajayi
ajayi@umich.edu
University of Michigan
Ann Arbor, Michigan

Minsoo Kim
mik226@ucsd.edu
University of California, San Diego
La Jolla, California

Guanru Wang
wguanru@umich.edu
University of Michigan
Ann Arbor, Michigan

Mehdi Saligane
mehdi@umich.edu
University of Michigan
Ann Arbor, Michigan

ABSTRACT

Several academic EDA tools have been released; however, few are used in real tapeouts, even by other academics. Robust open-source tools require feedback and direction from users. To this end, OpenROAD employs end-users as internal design advisors who bring with them the experience of multiple tapeouts and EDA tool flow development. This paper discusses the OpenROAD design advisors' ongoing work to bring OpenROAD from a collection of tools to an end-to-end autonomous design flow. We discuss our work to fill in the gaps for a full RTL-to-GDS design flow, assemble a full-flow test suite reflective of real tapeouts, debug flow-level issues between tools, and bridge the gap between OpenROAD developers and others in the open-source community. Lastly, we discuss OpenROAD's long-term goal to become fully autonomous, and what that means from a user's perspective.

CCS CONCEPTS

• **Hardware → Software tools for EDA**.

KEYWORDS

OpenROAD, open-source, RTL-to-GDS

ACM Reference Format:
Austin Rovinski, Tutu Ajayi, Minsoo Kim, Guanru Wang, and Mehdi Saligane. 2020. Bridging Academic Open-Source EDA to Real-World Usability: Invited Paper. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '20), November 2–5, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3400302.3415734

1 INTRODUCTION

Access to high-quality electronic design automation (EDA) tools, required engineering expertise, and lengthy project schedules have long been a barrier to hardware startups and hobbyists. Restrictive licensing has also been a massive impediment to the academic community due to virtually all licenses prohibiting sharing of scripts or results. With this motivation in mind, The OpenROAD Project aims to create a fully autonomous, open-source tool chain for full "RTL-to-GDS" digital layout generation. With such a tool, numerous issues can be addressed, including engineering resources, licensing, collaboration, and reproducibility [7, 8, 12].

A critical aspect of creating usable open-source software is to receive feedback and direction from users. As such, OpenROAD employs a group of experienced digital SoC designers as *internal design advisors*. Our job is to act as the first users of the OpenROAD tools and form a quick feedback loop with developers in order to iterate software quickly.

The original intent of the internal design advisor was to use real-world designs for tool testing and feedback, as well as provide human intelligence to guide tool development (similar to application or product engineers). Over the course of the project, however, it became apparent that key responsibilities for realization of OpenROAD goals [7, 8] did not fit cleanly into the project's organization structure. As such, the design advisor role has evolved to encompass several additional tasks which can be categorized under two responsibilities:

Flow Development. Even with well-defined OpenROAD tool interfaces, orchestrating a full RTL-to-GDS flow is a non-trivial task. Since creating the previous iterations of OpenROAD-flow [7, 8], we have substantially overhauled the flow to transition from a tool chain of stand-alone binaries to an integrated app. In addition, we are responsible for interjecting flow-level solutions which increase autonomy and reduce burden on developers. Such solutions act as initial scaffolding to improve autonomy from a user's standpoint and allow developers to focus on critical tool features.

Test Infrastructure. The integration of OpenROAD tools into a single app highlighted the project's need for continuous integration (CI) infrastructure (as noted by Kahng [12]). The OpenROAD Project operates in a delicate situation of testing with proprietary commercial data but developing with public infrastructure (e.g. GitHub). In addition, CI infrastructure requires maintaining good unit and integration tests for code coverage and metric tracking. We have set up a Jenkins CI infrastructure to balance these demands and create a secure but productive CI flow for the tool developers.

In this paper, we discuss the background of The OpenROAD Project (Section 2), and the main responsibilities of the internal

ICCAD '20, November 2–5, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415734

design advisors (Sections 3 & 4) in order to bring OpenROAD from a collection of tools to a full RTL-to-GDS flow. Next, we discuss OpenROAD's long-term goals of autonomy and quality, and the roadmap we want from a user's perspective (Sections 5 & 6). We conclude with our key lessons learned and goals for OpenROAD (Section 7).

2 BACKGROUND

The OpenROAD Project was launched in June 2018 within the DARPA IDEA program to create an open-source, fully autonomous RTL-to-GDS flow. The RTL-to-GDS process implements a register-transfer-level (RTL) description of a circuit into the Graphic Design System (GDS) format, representing a mask layout which fabrication facilities use to manufacture chips. This implementation process is broken down into several sequential steps, generally referred to as a design implementation flow. Flows differ among designers for a variety of reasons, but the most basic flows include synthesis, floorplanning, placement, clock tree synthesis (CTS), optimization, and routing. Verification is also important to verify that the design is manufacturable and free from critical bugs.

"OpenROAD" or "the OpenROAD app" is a collection of physical design tools which can take a Verilog netlist and perform placement and routing (PnR) to output a physical design in the Design Exchange Format (DEF). The OpenROAD app only covers the PnR portion of the flow (floorplanning to routing[1]). The other important steps – synthesis, DEF to GDS conversion, and verification – are performed using third-party open-source tools (Yosys [16] and KLayout [13]). To fulfill OpenROAD's RTL-to-GDS commitment, "OpenROAD-flow" acts as a wrapper around these tools and forms a full design implementation flow. These tools are all available from github.com/The-OpenROAD-Project.

This past year, The OpenROAD Project has spent significant effort on support for a commercial 14nm platform. With this milestone coming to a close, OpenROAD is entering the second phase of the project and ready to focus on additional directions.

3 FLOW DEVELOPMENT

3.1 Original Intent

The initial task of the internal design advisors was to test the OpenROAD tools with previously taped-out designs and provide rapid feedback throughout the development process. It quickly became apparent that a flow would be required to achieve this task; however, flow development was not clearly defined at the launch of The OpenROAD Project.

OpenROAD started as a collection of separate tool binaries accomplishing various steps of a design flow. The tools varied in maturity and some tools/steps were not initially available. In addition, the interfaces and expectations between tools were not concretely defined. To address this, we needed a flow able to pass designs through all available steps provided by the development team. This included the ability to skip or work around flow steps as necessary. Therefore, the earliest iterations of our flow leveraged commercial tools to generate "clean" artifacts (DEFs, floorplans, guides, etc.) needed to exercise each tool/step. The initial iteration of the flow chained steps together using GNU Make, and this flow has continued to evolve with the maturing tools.

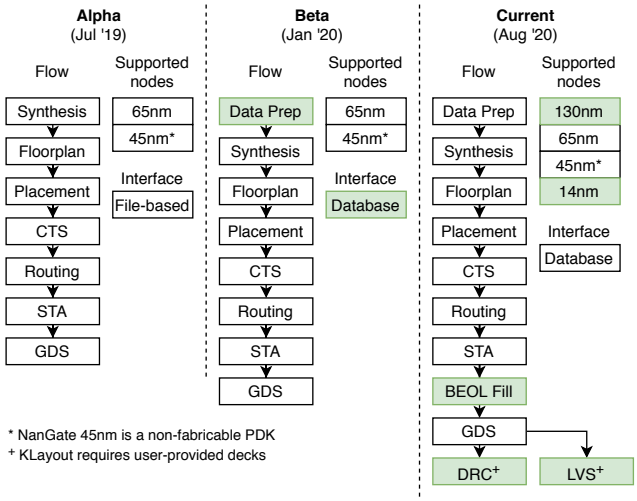

Figure 1: Evolution of OpenROAD-flow

3.2 Responsibilities

Maintenance of the OpenROAD-flow repository is now the largest and foremost responsibility of the design advisors. While the OpenROAD app offers a collection of PnR tools, there remains a huge gap between the OpenROAD app and a full RTL-to-GDS flow. OpenROAD-flow provides wrappers around Yosys, the OpenROAD app, and KLayout to fulfill this role. Yosys provides synthesis from Verilog RTL to netlist, OpenROAD provides PnR from netlist to DEF, and KLayout provides DEF to GDS conversion as well as design rule checking (DRC) and layout-versus-schematic (LVS) checking.

Figure 1 shows the evolution of OpenROAD-flow from its initial implementation [7, 8] to the current iteration. The largest improvement to OpenROAD-flow was the shift from a file-based interface to a unified database interface. The alpha release of OpenROAD used separate binaries for each tool and relied on a patchwork of configuration files and command-line arguments to run each tool. Once the tools were integrated into a unified app[1], all interfaces were also unified into a single binary with a Tcl interface. OpenROAD-flow has expanded its support to include 14nm FinFET and the newly open-sourced SkyWater 130nm platform. In addition, open-source DRC and LVS are available through KLayout; however, very few process development kits (PDKs) have KLayout rule decks available. Community-sourced decks are available for NanGate45 and are expected for SkyWater130, but the outlook for other commercial PDKs remains dim.

In addition to adding more flow stages as shown in Figure 1, another key responsibility is simplifying the user's flow interactions. To this end, the important data preparation step aims to minimize the number of user adjustments needed to set up a commercial PDK, such that it is usable by OpenROAD-flow. Optimization, while not explicitly shown, is embedded in the placement and CTS stages.

[1]TritonRoute is a separate binary at time of writing, but integration is expected soon.

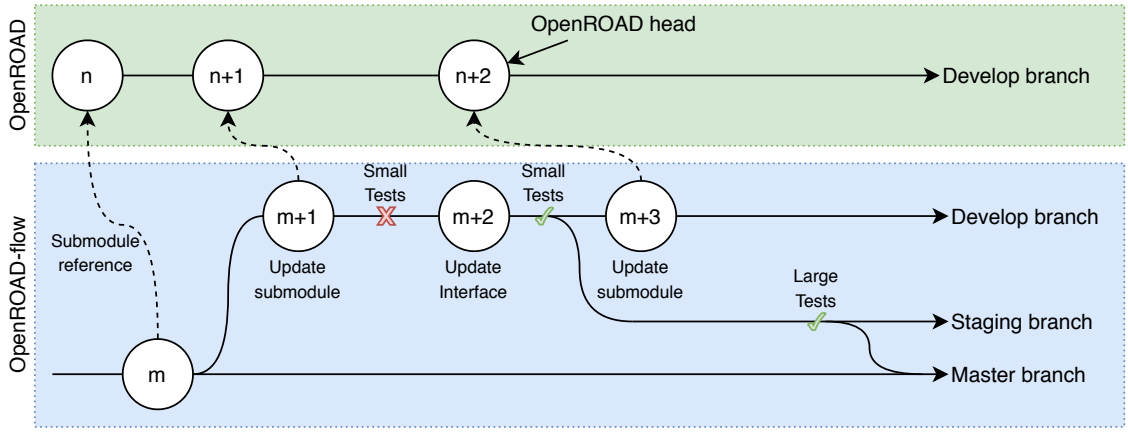

Figure 2: OpenROAD-flow submodule branching methodology

3.3 Challenges

Developing and maintaining OpenROAD-flow has led to several challenges along the way to meeting OpenROAD's development schedule goals. In this section, we discuss some of the most difficult aspects we faced.

3.3.1 Tool Synchronization. One of the main challenges for flow maintenance has been synchronization. OpenROAD-flow acts as a wrapper on top of OpenROAD and other tools. Therefore, whenever the underlying tools change, OpenROAD-flow must change as well. This problem bears resemblance to software libraries and packaging. However, OpenROAD developers frequently update the application programming interface (API) or introduce new features, and they look for feedback within on the order of hours or days. Therefore, synchronization occurs at a scale that is too fine-grain to rely on software packaging.

Our approach to this problem relies on asynchronous, regular updates to OpenROAD-flow. The current methodology is shown in Figure 2. OpenROAD-flow maintains a reference to a specific OpenROAD commit (and other tools) via git submodules, which allows OpenROAD-flow to maintain API synchronization with OpenROAD. When updates are committed to OpenROAD, we can asynchronously update OpenROAD-flow's submodule reference, perform API updates, and then merge the changes into the master branch. This model differs from the one described by Kahng [12] in a few regards. In particular, OpenROAD-flow separates tests into small pipe-cleaning tests and large quality of result (QoR) tests across separate Jenkins pipelines triggered by separate branches. The main reason for this strategy is to reduce build server load: if a large QoR test is triggered from every commit, the build server can quickly become overloaded. We instead use automatically triggered small tests for every commit to the development branch, and then large tests only on merges to the "staging" branch. Merges to the staging branch occur at a regular interval (i.e., nightly), and merges to the master branch are triggered by a successful QoR test.

3.3.2 Tool Workarounds. In cases where the root issue resided between tools, the design advisors would often be able to perform external processing as a workaround for lack of tool support. This allowed developers to focus on more critical issues. Some previous examples include fixing incorrect or inconsistent DEF outputs between tools, fixing technology-dependent tool issues, and adding partial support for yet-to-be-supported file formats (e.g., interconnect parasitics).

Adding workarounds remain a balancing act, however, as they often add technical debt which burdens future development. Several of these workarounds have enabled the development team to hit our schedule for 14nm node support, but now underlying issues must be thoroughly investigated to reduce technical debt.

3.3.3 Parameter Tuning. Virtually all EDA tools require human input to identify design intent and constraints. While OpenROAD's long-term goal is to automate much of this process, OpenROAD developers today need reasonably selected inputs in order to debug algorithms and identify smaller-scale problems. The design advisors have often provided the human intelligence to select and tune design parameters, such as design area and utilization, cell placement padding, and global routing settings. Parameter tuning is a normal part of designing with EDA tools and is quite straightforward for experienced designers. The more challenging aspect of parameter tuning is trying to identify tool pitfalls to the developers. For example, if decreasing global placement density and reducing global routing per-layer resource allocations do not result in reduced routing violations, then the issue may be that the detailed placer is causing pin access issues for the detailed router. As design advisors, we have to be familiar with the entire flow and process rules in order to narrow down issues appropriately.

In addition, the challenge becomes more difficult when applying this tuning across platforms. The design advisors are expected to maintain good configurations for each design across all supported nodes - from 130nm down to 14nm. The diverse routing rules, cell libraries, metal stacks, etc. across supported nodes dictate separate parameters across platforms, in addition to tuning designs.

3.3.4 Generic Node Enablement. Detailed routing for FinFET technology nodes is a significant challenge for academic research. A routing tool must understand all of the hundreds of complex design

rules in order to properly route designs without design rule violations (DRVs). To the best of our knowledge, no academic detailed router other than OpenROAD's TritonRoute [6] supports FinFET (sub-20nm) nodes. Even among commercial tools, few commercial routers successfully route FinFET-node designs without DRVs.

As internal design advisors, helping TritonRoute to achieve zero DRVs with limited time and resources was a significant challenge. Following the "generic node enablement" methodology [12], we made and enforced assumptions in routing rules to minimize the required design rule support:

- **Unidirectional and on-track routing.** Only on-track routing in the preferred direction is allowed. Bidirectional routing triggers complex design rules, such as spacing to convex or concave corners and color-aware design rules for multiple patterning technology layers.
- **Minimum-width routing.** Non-default routing (NDR) is not allowed. Wide metals trigger width-aware spacing rules, via enclosure rules and minimum numbers of vias.
- **On-track pins for macro cells.** Macro cells, such as SRAMs and other IPs, often have non-uniform pin widths and shapes. By creating "wrapped LEF" views with on-track, minimum-width pins, we can provide a view with simplified pin access without violating the two previous assumptions.
- **Routing-friendly P/G distribution.** Minimum-width routing is generally not an option for the power delivery network (PDN). In order to avoid forcing the router to consider NDRs, OpenROAD's PDN generator adds routing blockages and uses on-track, minimum-width stacked vias between power stripes. This allows the router to perform clean routing without any additional rule support.

4 TEST INFRASTRUCTURE

4.1 Design Suite

4.1.1 Original Intent. A critical aspect of making OpenROAD a usable tool is ensuring it is tested on real-world design data. The original intent of the design advisors was to curate a suite of test cases based on previous tapeouts of real designs. We quickly ran into several issues that prevented us from creating meaningful test cases for the developers.

Candidate designs that we identified for test cases proved too complex, and they overwhelmed the early OpenROAD tool capabilities. Our initial test cases had many advanced features not yet supported by OpenROAD, such as dense floorplans, multiple power domains, multiple clocks, etc. Additionally, some designs relied on circuit-level techniques which were incompatible with OpenROAD's all-digital design flow, or contained proprietary IP which prevented transmission to the developers.

To address this, we stripped down the test cases to accommodate the tool capabilities, project milestones and specific features we needed to evaluate. As the tools matured, we progressively tightened design constraints and added more test cases from both the design advisors and the open-source community.

4.1.2 Responsibilities. OpenROAD-flow now maintains a suite of designs containing source RTL and multi-platform constraints. The suite is a combination of open-source designs provided by both the design advisors and the community (summarized in Table 1). This suite provides several properties critical to maintaining an open-source flow:

- **Diverse** - The designs provided range from a few hundred instances to over 400k instances. Small designs allow developers to pipe-clean and debug tools quickly. In addition, users can run small designs quickly to validate their tool and flow setup. Large designs provide more complex developer test cases as well as benchmarks for QoR.
- **SoC-level** - OpenROAD-flow provides full-chip designs with I/O rings, enabling developers to test tools at the SoC level rather than only at the block level.[2]
- **Compact** - OpenROAD-flow provides a variety of test cases, but not so many that full regression testing becomes infeasible. The included designs are curated to be representative of a spectrum of real-world designs.
- **Real** - Almost all of the included designs have been taped out and are representative of real-world designs.
- **Cross-platform** - All designs are platform-independent and can be ported across processes. Macros must be regenerated for each platform, but the interfaces are designed to require no source changes.

In addition to design sources, we maintain flow configuration parameters for each design, including SDC and OpenROAD parameters. As the tools change, changing design parameters may be warranted. For example, placement density may increase over time as OpenROAD becomes more capable and can realize the benefits from shorter inter-cell distances without incurring DRVs.

4.1.3 Challenges. The main challenge to maintaining the design suite is likely tool compatibility. Many of our designs' sources are in SystemVerilog, which is only partially supported by Yosys. Our solution was to automatically convert the source to Verilog and maintain the generated source in the repository. This solution unfortunately loses the semantics and readability of the original designs, but it satisfied our use case of obtaining a test design. Many new open-source efforts in SystemVerilog parsing and conversion have arisen since the start of the OpenROAD Project, and our existing approach may be revisited as these efforts mature.

4.2 Continuous Integration

4.2.1 Original Intent. Continuous integration was another area which was not clearly enumerated in the original project task structure. Writing tests is often considered a responsibility of the programmers who write the code, and it was believed that regression testing could be handled by individual developers. Such a structure was sensible at the beginning of the project due to all of the tools being self-contained programs. However, several issues limited the scalability of this approach:

- **Issue reporting** - Even with developers being responsible for unit tests and design advisors for integration tests, problems are reported for the tool which fails and not necessarily the tool which creates the problem. This behavior can create

[2] A lack of open-source I/O cells limits what can be distributed publicly. The SkyWater130 platform plans on releasing I/O cells, and we expect to incorporate them shortly thereafter.

Design	Description	Source	Inst. count	Macro count	RTL taped out?
gcd	Greatest common denominator	Authors	250	0	Yes
dynamic_node	2D mesh router	[10]	8,090	0	Yes
vanilla5	Vanilla-5 CPU core	[11, 15]	12,300	4	Yes
ibex	Ibex CPU core	[4]	14,600	0	Yes
aes	AES encryption	[2]	15,000	0	Yes
bp_fe_top	Black Parrot CPU front-end	[14]	24,400	11	Yes
tinyRocket	Rocket generator CPU	[9]	25,100	2	No*
bp_be_top	Black Parrot CPU back-end	[14]	39,400	10	Yes
jpeg	JPEG encoder	[1]	52,600	0	Unsure
swerv	SweRV EH1 CPU (core only)	[3]	82,300	0	Yes
swerv_wrapper	SweRV EH1 CPU (with macros)	[3]	83,900	28	Yes
black_parrot	Black Parrot CPU core	[14]	121,000	24	Yes
ariane	Ariane CPU core	[17]	150,000	37	Yes
coyote	Coyote CPU core	[11, 15]	222,000	15	Yes
bp_multi_top	Black Parrot quad-core CPU	[14]	852,000	196	Yes

* several chips have been taped out using the Rocket generator, but tinyRocket RTL has not

Table 1: OpenROAD-flow design suite. Instance counts are collected post-synthesis from Yosys with a commercial cell library.

a "hot-potato" issue where developers have to pass the issue around to figure out the root cause.
- **Delayed feedback** - Lag time between introducing a change and receiving design advisor feedback inherently limits the rate at which developers can commit stable updates.
- **Portability** - Some code changes may only be stable in the developer's environment, where the code was tested, and unstable elsewhere.

With these problems, it became clear that investment in a continuous integration solution would be required to move forward. With CI for both unit tests and integration tests, developers can immediately pin down which code change broke the full-flow tests.

4.2.2 Responsibilities. The design advisors are partially responsible for maintaining the Jenkins CI infrastructure for the whole project, and wholly responsible for maintaining regression tests for the OpenROAD-flow repository. As mentioned in Section 3.3.1, we update the submodule reference to OpenROAD regularly and perform full-flow regression tests on the tools. Updates which pass all regression tests (which is currently every design in OpenROAD-flow's design suite) will be pushed to the master branch and be ready for use by the community.

4.2.3 Challenges. OpenROAD's CI has faced two main challenges.

Test scale. EDA tools are renowned for consuming significant computation time, and OpenROAD is no different. The largest designs in the OpenROAD-flow design suite can take more than 12 hours to run, meaning that running the full test suite on every push is not feasible. Instead, we adopt the approach mentioned in Section 3.3.1. A subset of test cases are triggered on every commit to the development branch, and the full test suite is only run nightly. The small tests are curated to run in a small amount of time, fail quickly if a change completely breaks the flow, and include designs both with and without macros. Our small test currently completes in under 30 minutes and includes gcd, aes, and tinyRocket. Our large test includes the full design suite and takes approximately 12 hours when fully parallelized.

Private tests. Using real, commercial platforms is critical to testing tool correctness. However, nearly all commercial platforms are regarded as trade secret and require significant security to ensure their privacy. Therefore, neither typical open-source CI practices nor closed-source CI practices fit our testing requirements. In order to ensure security, but still report test statuses back to the open-source repository, we ensure that the private CI server can send statuses, but not receive any web hooks. In addition, the CI server only tests on protected branches so that only trusted code can be executed on the CI server.

5 TOWARDS FULL AUTONOMY

While The OpenROAD Project has made long strides toward more automation, many complex challenges remain. Our main goals are to (1) focus on improving the user experience in the short term, and (2) focus on improving autonomy in the long term.

5.1 Improving User Experience

Achieving full autonomy is no small feat and we expect it to take a significant amount of time. In the short term, we aim to improve the user experience so that providing human input is more intuitive and tool issue resolution is less cumbersome. The following subsections detail key milestones we want to see from The OpenROAD Project as designers.

5.1.1 Improved Documentation. Significant developer effort was invested in achieving a tapeout-ready design in a 14nm FinFET node. After completing this goal, we propose providing several resources that users have come to expect from commercial tools:

- Documentation on all required OpenROAD-flow input files/parameters, and instructions on how to generate/select them.
- Tutorials for setting up new designs and new platforms.
- Documentation, uniformity, and adjustable filtering for all tool messages (info, warning, error, and critical).
- Generated documentation for OpenROAD code and APIs.

Common feedback from community members indicates that OpenROAD-flow's biggest hurdle is determining the source of errors – whether from user parameters, user designs, or tool bugs. We believe the points above, particularly the improvement of tool messages, will enhance users' abilities to resolve issues themselves.

5.1.2 Improved Access. Open software installation needs to be easy, fast, and widely available. Currently, OpenROAD's only officially supported OS is CentOS 7. We advocate adding official support to more operating systems, including CentOS 8 and Ubuntu 18, by testing builds in our CI system. All operating systems which support OpenROAD, KLayout, and Yosys dependencies should be able to build and run OpenROAD-flow, even without official support.

We also advocate making software packages available in the long term. The main difficulty with packaging is that OpenROAD-flow directly depends on OpenROAD's frequently changing API, as mentioned in Section 3.3.1. Our current solution of git submodules only works well when building from source. Versioning also becomes an issue, as matching a version of OpenROAD-flow with the corresponding version of OpenROAD would be cumbersome and unintuitive. This is currently an open problem, but we believe packaging is important to simplifying access to OpenROAD.

5.2 Reducing Manual Effort

The main goal of full autonomy is reducing the number of required user inputs. OpenROAD-flow is currently in-line with commercial workflows in terms of manually specifying parameters and fine-tuning a design with human guidance. To reach full autonomy, these human inputs will need to be replaced with machine intelligence.

OpenROAD-flow currently requires about 50 configuration parameters to be set per-process, not including the PDN, I/O, or design-specific configurations. For an inexperienced user, setting these parameters correctly can be difficult. We propose aggressively reducing the number of required manual parameters by replacing them with ones automatically extracted from platform data. This will allow OpenROAD-flow to reduce the level of experience needed to set up new platforms, while still allowing expert users to manually tune parameters if desired.

One such example parameter is the target design utilization (cell density). Setting this parameter can be a somewhat difficult task: a utilization target that is too high will blow up tool runtime and potentially provide an unroutable design; on the other hand, a utilization that is too low can turn competitive power-performance-area (PPA) results into non-competitive. The maximum utilization often correlates most strongly to the process, but can also be influenced by the design. To automate, OpenROAD will need heuristics to select a utilization which balances runtime with PPA based on process and design characteristics.

6 TOWARDS IMPROVED RESULTS

In contrast to many commercial tools, The OpenROAD Project sees autonomy as a primary constraint and QoR as secondary. This means that OpenROAD will focus on providing a clean and manufacturable design, without human intervention, over improvements to PPA. However, improving QoR is not always orthogonal to autonomy, as improved algorithms can lead to reductions in design rule violations. Improving QoR is also important for increasing OpenROAD's user base, as discrepancies in QoR can lead users towards closed-source proprietary tools.

6.1 Inter-tool Feedback

OpenROAD's shift from standalone binaries to an integrated app was a major advancement due to the common database substrate. This common substrate allows tools to interact with each other much more easily, but OpenROAD is only beginning to take advantage of this. We advocate focusing on mechanisms which allow feedback between tools to enhance QoR.

For example, the global and detailed placers focus mainly on half-perimeter wire length and cell displacement as their main optimization metrics. However, pin access is a primary constraint which can determine whether the router will be able to perform clean routing. Because of this limitation, users may try to reduce placement density, increase cell padding, and/or disable use of certain standard cells to avoid DRVs. Yet, macro placements in the floorplan, or the setup of global routing layer resources, could ultimately turn out to be "the culprit". We would like to see OpenROAD incorporate mechanisms to make upstream tools more aware of downstream problems so that users can save iteration time and achieve higher QoR. Simple versions of this might see the router's pin access analysis invoked by the detailed placer, or the global router being run under the hood of the placer. The common database substrate could also enable greater empowerments of OpenROAD's tools – e.g., the router curing a pin access-induced DRV by modifying the detailed placement.

6.2 Automatic Clock Gating

Automatic clock gating (ACG) is one of the most significant features that OpenROAD-flow does not yet support. Traditionally, ACG is implemented as part of synthesis, but Yosys does not currently support ACG. Due to Yosys being a third-party tool with separate infrastructure, addition of ACG may be difficult in that respect. We believe that the benefits to power and area are worth pursuing. We advocate that OpenROAD investigate a path towards implementing ACG, whether by upstreaming changes to Yosys, or by working within OpenROAD on a post-synthesis netlist.

6.3 Parasitic Extraction

Parasitic extraction is a key step which has been missing from OpenROAD-flow. We have worked around the issue by (1) extracting per-unit parasitics, (2) multiplying by wire length, then (3) derating to correlate with extracted parasitics. Parasitic correlation can be a cumbersome process, and more importantly, still relies on an external golden tool for accurate data. The project has invested significant effort toward bringing up a parasitic extraction tool, OpenRCX [5], which is expected to be introduced to OpenROAD soon.

OpenRCX will significantly aid in parasitic calibration as it reduces the trial and error from per-unit parasitics, but it still faces the issue of calibration from a golden model, which often uses data in an encrypted, unreadable format [12]. We advocate that OpenROAD explore two different paths for parasitic extraction:

- **Automatic golden correlation** - Proprietary golden model use is unavoidable for most commercial platforms. Commensurate with OpenROAD's vision, working with these golden models should be as automated as possible. For example, OpenRCX creates DEFs to provide to a 3D solver, and the user need only provide the corresponding parasitics (SPEFs) back to OpenRCX.
- **Full-stack solution** - The release of the open-source SkyWater130 platform provides a tremendous opportunity for OpenROAD. We advocate that OpenROAD develop a fully open-source parasitic extraction stack using the PDK.

7 CONCLUSION

The OpenROAD internal design advisors have greatly helped OpenROAD to get off the ground. Our expectations changed over time as we better understood the challenges facing open-source EDA flows, and we have learned important lessons along the way:

- **EDA is not typical open-source software** - Open-source projects typically have access to open data, whereas EDA must work with proprietary data to be practical. Proprietary data necessitates additional infrastructure and maintenance.
- **Test early and test often** - An RTL-to-GDS tool has an extraordinarily high minimum viable product of "placed and routed chip". Breaking down test cases into appropriate scope and complexity for early testing is incredibly important.
- **Expect the unexpected** - Even with significant EDA experience across our team, several unforeseen tasks arose. Project members stepping up to handle tasks outside their expertise significantly helped to keep the project moving forward.

We also have an optimistic outlook for OpenROAD's future, and we have identified some features that would offer the most benefit from a designer's perspective:

- **Improved user experience** - Intuitive, accessible, and documented software is important for reducing user effort. While full automation is a long-term strategy to improving the user experience, improving documentation and accessibility can provide quick returns on investment.
- **Improved quality** - The easier it is to get high-quality results from OpenROAD, the easier it will be to get community investment in it. Features such as inter-tool feedback, automatic clock gating, and accurate parasitic extraction can significantly improve QoR.

ACKNOWLEDGMENTS

This work was funded under the DARPA IDEA program (grant HR0011-18-2-0032). We would like to thank the rest of the OpenROAD team, and especially thank our early adopters and contributors. We hope that OpenROAD can continue to grow into a tool that reshapes EDA.

REFERENCES

[1] 2008. Video compression systems. https://opencores.org/projects/video_systems.
[2] 2018. AES (Rijndael) IP Core. https://opencores.org/projects/aes_core.
[3] 2019. SweRV RISC-V Core 1.1 from Western Digital. https://github.com/westerndigitalcorporation/swerv_eh1.
[4] 2020. Ibex RISC-V Core. https://github.com/lowRISC/ibex.
[5] 2020. OpenRCX. https://github.com/The-OpenROAD-Project/OpenRCX.
[6] 2020. TritonRoute. https://github.com/The-OpenROAD-Project/TritonRoute.
[7] Tutu Ajayi, David Blaauw, Tuck-Boon Chan, Chung-Kuan Cheng, Vidya A. Chhabria, David K. Choo, Matteo Coltella, Sorin Dobre, Ronald G. Dreslinski, Mateus Fogaça, Soheil Hashemi, Abdelrahman Hosny, Andrew B. Kahng, Minsoo Kim, Jiajia Li, Zhaoxin Liang, Uday Mallappa, Paul Penzes, Geraldo Pradipta, Sherief Reda, Austin Rovinski, Kambiz Samadi, Sachin S. Sapatnekar, Lawrence Saul, Carl Sechen, Vaishnav Srinivas, William Swartz, Dennis Sylvester, David Urquhart, Lutong Wang, Mingyu Woo, and Bangqi Xu. 2019. OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain. In *Proceedings of Government Microcircuit Applications and Critical Technology Conference* (Albuquerque, NM, USA) *(GOMACTech '19)*.
[8] Tutu Ajayi, Vidya A. Chhabria, Mateus Fogaça, Soheil Hashemi, Abdelrahman Hosny, Andrew B. Kahng, Minsoo Kim, Jeongsup Lee, Uday Mallappa, Marina Neseem, Geraldo Pradipta, Sherief Reda, Mehdi Saligane, Sachin S. Sapatnekar, Carl Sechen, Mohamed Shalan, William Swartz, Lutong Wang, Zhehong Wang, Mingyu Woo, and Bangqi Xu. 2019. Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project. In *Proceedings of the 56th Annual Design Automation Conference 2019* (Las Vegas, NV, USA) *(DAC '19)*. Association for Computing Machinery, New York, NY, USA, Article 76, 4 pages. https://doi.org/10.1145/3316781.3326334
[9] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. 2016. *The Rocket Chip Generator.* Technical Report UCB/EECS-2016-17. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html
[10] Jonathan Balkind, Michael McKeown, Yaosheng Fu, Tri Nguyen, Yanqi Zhou, Alexey Lavrov, Mohammad Shahrad, Adi Fuchs, Samuel Payne, Xiaohua Liang, Matthew Matl, and David Wentzlaff. 2016. OpenPiton: An Open Source Manycore Research Framework. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA) *(ASPLOS '16)*. ACM, New York, NY, USA, 217–232. https://doi.org/10.1145/2872362.2872414
[11] Scott Davidson, Shaolin Xie, Christopher Torng, Khalid Al-Hawaj, Austin Rovinski, Tutu Ajayi, Luis Vega, Chun Zhao, Ritchie Zhao, Steve Dai, Aporva Amarnath, Bandhav Veluri, Paul Gao, Anuj Rao, Gai Liu, Rajesh K. Gupta, Zhiru Zhang, Ronald G. Dreslinski, Christopher Batten, and Michael B. Taylor. 2018. The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric: Fast Architectures and Design Methodologies for Fast Chips. *IEEE Micro* 38, 2 (2018), 30–41.
[12] Andrew B. Kahng. 2019. Looking Into the Mirror of Open Source. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.
[13] Matthias Köfferlein. 2018. KLayout.
[14] Daniel Petrisko, Farzam Gilani, Mark Wyse, Tommy Jung, Scott Davidson, Paul Gao, Chun Zhao, Zahra Azad, Sadullah Canakci, Bandhav Veluri, Tavio Guarino, Ajay Joshi, Mark Oskin, and Michael B. Taylor. 2020. BlackParrot: An Agile Open Source RISC-V Multicore for Accelerator SoCs. *IEEE Micro* (2020).
[15] Austin Rovinski, Chun Zhao, Khalid Al-Hawaj, Paul Gao, Shaolin Xie, Christopher Torng, Scott Davidson, Aporva Amarnath, Luis Vega, Bandhav Veluri, Anuj others Rao, Tutu Ajayi, Julian Puscar, Steve Dai, Ritchie Zhao, Dustin Richmond, Zhiru Zhang, Ian Galton, Christopher Batten, Michael B. Taylor, and Ronald G. Dreslinski. 2019. A 1.4 GHz 695 Giga RISC-V Inst/s 496-core Manycore Processor with Mesh On-Chip Network and an All-Digital Synthesized PLL in 16nm CMOS. In *2019 Symposium on VLSI Circuits*. IEEE, C30–C31.
[16] Clifford Wolf, Johann Glaser, and Johannes Kepler. 2013. Yosys-a Free Verilog Synthesis Suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*.
[17] Florian Zaruba and Luca Benini. 2019. The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 11 (Nov 2019), 2629–2640. https://doi.org/10.1109/TVLSI.2019.2926114