Contributions to OpenROAD from Abroad: Experiences and Learnings

Invited Paper

Mateus Fogaça[1,3], Eder Monteiro[3], Marcelo Danigno[5], Isadora Oliveira[1,3], Paulo F. Butzen[1,4] and Ricardo Reis[1,2,3]

[1]PGMicro/[2]PPGC, [3]Inst. de Informática, [4]Dep. de Elétrica, Universidade Federal do Rio Grande do Sul
[5]Centro de Ciências Computacionais, Universidade Federal do Rio Grande - FURG
{mpfogaca,emrmonteiro,isoliveira,reis}@inf.ufrgs.br,marcelo@furg.br,paulo.butzen@ufrgs.br

Abstract

The OpenROAD project is an ambitious initiative seeking to develop an automated, open-source RTL-to-GDSII flow. To build its complex toolset, OpenROAD brings together a team of industry experts, veteran scholars, and enthusiastic students from different schools and different countries. This paper first presents our path to becoming OpenROAD contributors, highlighting the nature of the OpenROAD project, the recruitment process, and the necessary logistics. We then summarize the contributions of the Brazilian team to the OpenROAD project; these comprise the development of five tools and more than 10K lines of released code, along with authorship or co-authorship of two publications in the research literature. We also summarize our experiences from working in a large software project: (i) working environment and relationship with people from around the world; (ii) task management and short turnaround times; (iii) continuous integration and testing; etc. Finally, we highlight the challenges of "refurbishing" academic research codes for use in the design of production ICs.

Keywords: Open-source, VLSI CAD, electronic design automation, RTL-to-GDSII, floorplan, placement, clock tree synthesis, routing

ACM Reference Format:
Mateus Fogaça[1,3], Eder Monteiro[3], Marcelo Danigno[5], Isadora Oliveira[1,3], Paulo F. Butzen[1,4] and Ricardo Reis[1,2,3]. 2020. Contributions to OpenROAD from Abroad: Experiences and Learnings: 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, 8 pages. https://doi.org/10.1145/3400302.3415742

1 Introduction

Modern technologies introduce an ever-increasing set of design rules and ever-more demanding power, performance, and area targets. The cost of IC design continues to increase, as complex SOC products require more commercial EDA tool licenses and larger teams of expert engineers. The largest semiconductor companies have extensive CAD and design organizations and can rely on close support from their EDA suppliers. On the other hand, small companies must work with a reduced number of licenses and less vendor support. These can discourage companies from taking risks, blocking innovation in the market. Small design teams or early what-if product conception can benefit from *open-source* EDA tools in two ways. First, open-source tools are free, and second, companies can customize or extend open source to achieve their goals. However, there are only a few available open-source EDA tools, most of which are developed for academic purposes and cannot handle real production ICs.

The OpenROAD project is an ambitious project launched in mid-2018 by the U.S. Defense Advanced Research Projects Agency (DARPA); it aims to develop an open-source 24-hour no-human-in-the-loop RTL-to-GDSII flow. (Unlike commercial EDA tools, OpenROAD must automatically generate DRC-clean, manufacturable layout.) To achieve its goals, the project's strategy relies on three *base technologies* seen in Figure 1. *Machine learning* allows the prediction of tool outcomes and enables the flow to auto-tune itself. *Extreme partitioning* affords problem decomposition and a divide-and-conquer approach to stay within turnaround time limits. *Parallel and distributed optimization* targets cloud deployment to explore multiple solution paths concurrently, maximizing design outcomes while mitigating noisy behavior of complex heuristics. A fourth element, *restricted layout*, simplifies the tool via "freedoms from choice".

To develop the complex set of tools that serve as building blocks of commercial RTL-GDSII flows, EDA companies rely on hundreds or thousands of R&D engineers. EDA tool developers must be well-acquainted with advanced programming techniques (especially in C and C++) as well as computer science and computer engineering foundations. Even the largest EDA companies struggle to find engineers with these

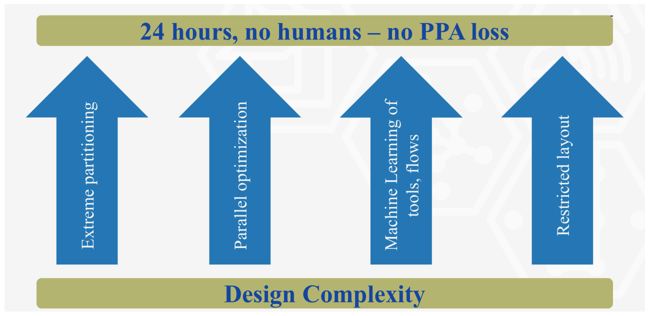

Figure 1. OpenROAD's strategy to conquer design complexity in a 24-hour, autonomous RTL-to-GDSII flow.

skills. Additionally, the number of graduate students in the field has decreased in recent years. Given this, as well as the goal of seeding a free open-source EDA software ecosystem, it is natural for OpenROAD to seek talents not only in the U.S., but also overseas. In this paper, we describe the experience of being part of the OpenROAD project, through the eyes of foreign students.

- We present how to become a contributor to the OpenROAD project. We discuss skills desired in a contributor, logistic challenges, and what to expect while working in the project. Notably, there are key differences between OpenROAD and a typical academic research project, bringing both advantages and disadvantages.
- We present the tools developed by us in the OpenROAD project, along with lessons learned from developing open-source code that supports the design of real production ICs.
- We present challenges experienced when working in a large project from abroad. We describe the experience of working with the industry veterans that lead the project. We also discuss the difficulties of managing many tasks with reduced manpower, and the best practices adopted for task management and continuous integration.

The remainder of our paper is organized as follows. Section 2 discusses the process and logistics of becoming a contributor to the OpenROAD project. Section 3 introduces our contributions to the project and discusses challenges we have faced as developers, while Section 4 describes initial studies and implementation toward the "extreme partitioning" base technology of OpenROAD. Section 5 describes some important learnings from our experiences in the project, and we conclude in Section 6.

2 Becoming an OpenROAD Contributor

We now discuss how the OpenROAD project tackles issues such as recruitment and logistics. We also note how OpenROAD differs from typical academic and industry contexts.

2.1 Not Research As Usual

The OpenROAD project deals with topics that are well-researched in academia. But, working for OpenROAD is not the same as working on an academic project. Research is an important element in the development of OpenROAD, and this can be quite similar to how universities and companies around the world tackle it. However, every other step is different from what is typically seen in an academic research group. While developing, the use and creation of unit tests is mandatory. Once code is finished and working, it must be checked to make sure it obeys the project's coding guidelines. Even with that, each addition must pass through the contribution flow, which requires the testing of results, removing any memory errors and leaks, checking naming standards, and review. This is much more than is normally done with student code (where fast development often ends with export of results into tables of a paper), but it is indispensable in any big project like OpenROAD – especially if it intends to be "built to last".

OpenROAD is based on software deliveries. Each contribution represents either an important part of the flow (e.g., a clock tree synthesis tool) or a specific fix. Once a contribution is made, the developer moves on to the next delivery. This approach, alongside an extremely aggressive schedule and shifting requirements from the sponsoring agency, can easily cause technical debt: (i) tools support narrow fields of use and functionality driven by deliverables; (ii) necessary functionalities may be deferred to meet delivery times; or (iii) GitHub issues and even failures in continuous integration may be "deferred" if they are not on any critical path. In OpenROAD, these technical debts are typically diagnosed and documented to queue them up for work in the next reporting period; this occurs in monthly project report documents that review the status of each tool. Regular, focused reports help to see which parts of the project need help and how to prioritize tasks. The approach is very different from usual academic practice, where the frequency of updates and reports depends on the researcher/student.

Planning and organization are also extremely important to coordinate multiple teams. Each contribution is specified either by an error report from a team member (usually, on an important testcase or "proof point" for the tool) or a known missing feature in a tool. Guiding implementations in this way allows OpenROAD technical leaders to make sure no effort is wasted and that every member is working with knowledge of the latest version of the code. We further discuss OpenROAD's working environment and project management in Section 5 below.

2.2 Recruiting

The ideal candidate to work in the OpenROAD project is someone with excellent programming skills and deep knowledge and experience on VLSI. Good luck finding many of these, as such candidates are lacking even in the EDA industry itself. Especially given the "not research as usual" characteristics of OpenROAD, graduate students who have the right skills and background could consider OpenROAD as misaligned with academic careers, due to the engineering-focused work and lack of emphasis on publications.

Our team bypassed this problem by recruiting *undergraduate* students with good programming skills and introducing them to the concepts of VLSI CAD. Senior project members who have decades of EDA industry experience work alongside these eager-to-learn students, providing them with unique learning and professional opportunities.

Recruiting talented undergraduates to the project brings longer learning curves for both programming skills and the basic concepts of this newly-introduced field. With undergraduates, time must be well-managed between their project obligations and their day-to-day classes and other ongoing education. But, in our experience bringing new talents to the EDA field has proved itself to be a smart decision.

2.3 Logistics

Programming skills and VLSI knowledge are essential for an OpenROAD contributor, but this is just the beginning. The development of the OpenROAD project requires a suitable infrastructure in terms of programming tools, computational power, and technology access. The OpenROAD flow today is composed of three main tools: Yosys [21], OpenROAD [18], and TritonRoute [20]. Each has its own requirements to build and execute, including compilers, interpreters, and third-party packages. A special subteam of OpenROAD, the "Internal Design Advisors" students and post-doc at the University of Michigan, uses commercial EDA tools (e.g., Mentor Calibre) to validate that OpenROAD outputs meet requirements. For communication, and to otherwise address the project's needs, a common workspace is helpful.

As Brazilian students of public universities, we face multiple infrastructure limitations. We do not have powerful servers where a large group of people can work. We lack access to design enablements and tools that are common elsewhere. This reflects the shortage of financial resources that is currently endemic in our universities. To overcome these barriers and to improve the velocity of development, we have taken the extraordinary step of obtaining appointments as visiting students (research collaborators) at the University of California, San Diego (UCSD).

For us to become visiting students at UCSD, it was mandatory to travel and stay in San Diego. During our stay, hosted by the OpenROAD PI, we had the opportunity to meet some project members and experience the work method of UCSD students. As visiting students at UCSD, we have access to a shared workspace along with other OpenROAD project members. This enables us to be productive developers in the OpenROAD project.

However, we still have particular circumstances of working in the OpenROAD project from abroad. It is not easy to have quality internet service in Brazil, and we depend on it to access the UCSD infrastructure. It is common to have our work negatively affected by internet connection issues. Support logistics also required some extra paperwork.

Finally, a small concern for some members of the Brazilian team was the language barrier. The communication in a different language with the other members of the OpenROAD project raised the necessity to improve English speaking and understanding. (Also, the project leadership often lives by the mantra of "if it is not written down, it does not exist".) While this was never an issue for us, we emphasize the importance of developing both spoken and written English communication skills.

3 Our Contributions to OpenROAD

We now give details of the tools developed and maintained by the Brazilian team. For each tool, we explain the addressed problem, optimization objectives and constraints, and integration with other tools in the flow. We conclude each of these discussions by noting near-term goals for the tool. Our tools are integrated into the OpenROAD app [18], a unified repository with almost all physical synthesis tools except for the TritonRoute detailed router. We also discuss how we have integrated the tools adapted from third-party codes with the OpenROAD app.

In the OpenROAD project, all the physical design steps are controlled by one environment: the OpenROAD flow [19]. It consists of a set of tools that implement all steps from logic synthesis to layout extraction. The inputs to this flow consist of an RTL Verilog file, design constraints, and design enablement elements such as Liberty, LEF and technology files. The output is a tapeout-ready GDSII file. Figure 2 depicts the main steps of the OpenROAD flow: logic synthesis, floorplan/power delivery network (PDN), placement, clock tree synthesis (CTS), routing, and layout finishing. Parasitic estimation and static timing analysis are also run to check that constraints are satisfied. From each step, the OpenROAD flow extracts log, design and metrics information; this can be used in project management dashboards, in diagnosing problems, or in simple learning applications. Contributions from the Brazilian team can be found in floorplan (ioPlacer, tapcell), clock tree synthesis (TritonCTS), and global routing (FastRoute).

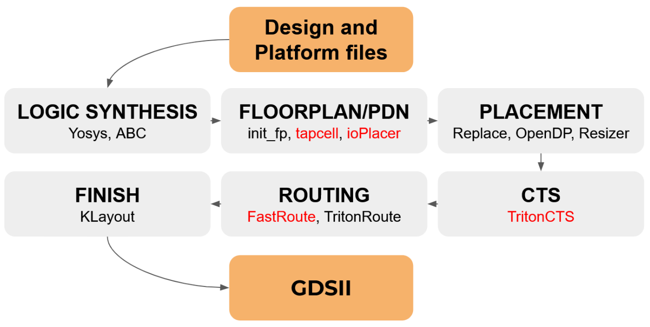

Figure 2. The OpenROAD flow. Tools in red are contributions from the Brazilian team.

3.1 Floorplan

Floorplanning may be one of the most incompletely-studied topics in physical design, with only macro placement and power-ground distribution having significant research literatures. We have worked on the tapcell insertion and I/O pin assignment, as described next.

Tapcell. *Welltap and endcap insertion* is a relatively straightforward task, which explains the lack of research papers and available academic codes on the subject. However, a usable RTL-to-GDSII flow must provide welltap and endcap insertion, which we accomplish in the *tapcell* code. Moreover, each technology node, memory generator and cell library brings its own set of design rules that must be properly observed. Our first contribution in floorplanning is a tapcell insertion script, implemented in Tcl, that supports *real design rules* for nodes from 130nm down to 14nm industrial enablements. The logic of tapcell, and its consideration of numerous corner cases (location of vertical macro edges relative to the default welltap "checkerboard", odd- or even-row location of bottom macro edges, N or FS orientation of bottom row in site map, etc.), is as specified by an experienced physical design methodologist in the OpenROAD team.

ioPlacer. Another critical but neglected floorplanning task is *I/O pin placement*. In chip-level design, PAD and I/O pin locations are commonly planned by the designers. However, for block-level design, the OpenROAD flow must perform automatic placement of I/O pins (i.e., terminals). This is essential if an analytic global placement engine is used, as is the case with OpenROAD. Our second contribution to the floorplanning stage is *a new I/O pin placement tool* called ioPlacer. As described in [4], ioPlacer addresses the following problem. *Given* the design netlist, the core area, one horizontal metal layer, one vertical metal layer and the routing tracks, ioPlacer *finds* on-track locations for each I/O pin from among a given set of available locations (called *slots*) along the design's core boundary. The ioPlacer objective is to *minimize* the total Manhattan wirelength of I/O nets. Novel techniques include speedups of the well-known Hungarian assignment algorithm, and heuristics for non-random, macro placement-aware I/O placement.

The future of tapcell and ioPlacer. The existing tapcell scripts are fast and stable. Therefore, going forward, our only goal with tapcell is to add support for more technologies. On the other hand, the current ioPlacer implementation has two limitations. First, we do not consider timing information while performing I/O pin placement. Second, we only place I/O pins in two metal layers (one horizontal and one vertical). We have received feedback from experts saying that the latter limitation is more critical to address. However, due to the high demand of work in OpenROAD and the fact that the current implementation is very stable, addressing this limitation has not yet been scheduled.

3.2 Clock Tree Synthesis

The clock tree synthesis tool, TritonCTS, is our oldest contribution to OpenROAD. Development started with the project itself, in June 2018. The starting goal was to build an open-source tool based on the original scripts and code of the Generalized H-Tree (GHTree) [5] approach. This brought many challenges since the original implementation relied on commercial tool scripts to perform various steps: (i) a commercial P&R tool parses the design and writes clock sink locations to a text file with a simple syntax; (ii) the clock tree topology is created using dynamic programming; (iii) clock sink clustering and clock buffer placement are performed via an integer linear programming (ILP) formulation and a commercial solver; (iv) the OpenAccess database [6] is used to add clock tree subnets back into the design; and (v) finally, the clock tree routing is done using the commercial tool.

Removing commercial code. To release the GHTree approach under a BSD-3 license, we first had to remove all commercial tool dependencies. For example, we replaced the P&R tool-based parsing by our own in-house DEF parser, operating on a tight schedule to meet timeline needs. This proved to be a bad choice, because time gained with the simple implementation was lost in many fixes to handle corner cases and possible syntaxes. (We now understand that a golden LEF/DEF parser should have been adopted from the start – a lesson that applies to many other basic EDA components.) Usage of the OpenAccess database was replaced by our own Verilog and DEF writers. Because open-source ILP solvers are not sufficiently performant, we replaced the original authors' use of commercial ILP solvers with min-cost flow based heuristics. Finally, the commercial router was replaced by our own OpenROAD global and detailed routers. Thus, after removal of commercial P&R and ILP tools, as well as OpenAccess, it was possible to release the first version of our CTS tool, named **TritonCTS**, on GitHub.

Technology characterization. When first released in GitHub, TritonCTS required a *technology characterization file* with a lookup table containing power, delay and capacitance information for buffered wires. We also released scripts to generate the characterization files; however, many users found it difficult to understand the characterization input arguments and to check whether characterization files were correctly generated. Learning from this, we have made the characterization automatic in the OpenROAD flow. In our current implementation, the characterization is called automatically in the beginning of CTS. To avoid recomputing the same numbers for every run made in a given enablement, the user can save the characterization file and use it in subsequent runs.

Starting from scratch: TritonCTS 2.0. The first year of TritonCTS release exposed many issues arising from the complex, "academic" nature of the GHTree approach. GHTree had been designed to work best using *even* sink placement distributions. Also, runtime and memory consumption of the high-dimensional dynamic programming scaled poorly with respect to the core area. With incorrect setup, the dynamic programming could run for up to an hour and consume 150GB peak memory, in a design with less than 200K clock sinks. Additionally, the original code would require significant changes to support multiple clocks, clock gating cells (CGCs) and generated clocks. With this in mind, we embarked on rewriting a "Simple CTS" from scratch. Within one month, we created a new tool, fully integrated with the OpenROAD app, called TritonCTS 2.0. Our new code has benefited from tight integration with the project's database (OpenDB) and timer (OpenSTA). This experience is a good example of how important a good base infrastructure (in our case, the OpenROAD app) is to the quality of the tool. The current implementation has a simple top-down flexible H-tree heuristic (i.e., a *non*-generalized H-tree!), but is robust and produces decent results in internal testcases.

The future of TritonCTS. The current version of TritonCTS still has obvious weaknesses, such as difficulty in achieving desired skew or insertion delay metrics. This is the subject of research toward a future new version of TritonCTS. One example weakness: the current use of a top-level H-tree brings high wirelength and latency, with a large number of buffers placed in any clock root to sink path. There are intrinsic challenges as well, notably that CTS sits between upstream placement and downstream routing, which makes fine-tuning difficult. Not only does testing and tuning of a new CTS solution requires the full flow context, but the CTS outcome is intimately tied to steps outside of TritonCTS such as global placement, placement legalization, and slew repair. Some post-CTS steps can unbalance the CTS solution, and need to be taken into better consideration by TritonCTS; other pre-CTS steps should better understand what it means to be "friendly" to TritonCTS.

3.3 Global Routing

FastRoute was developed originally by Pan et al. [8]. It is an academic global routing tool that uses a series of techniques – including congestion-driven and via-aware Steiner tree construction, and multi-source multi-sink maze routing – to achieve high-quality routing solutions. There are five versions of FastRoute, each adding new techniques to improve quality of results [9] [13] [12] [10]. In the OpenROAD flow, we have adapted FastRoute4.1 [10], the last version of the tool and the version open-sourced under BSD-3 license in November 2018 [15].

The original code of FastRoute4.1 uses the input and output file formats from the ISPD 2008 Global Routing Contest. These formats are academic and do not match the formats adopted in the OpenROAD flow, i.e., LEF/DEF as input and guides [7] as output. Therefore, a starting step was to create an interface between the formats used in the project and in FastRoute4.1, and integrating with OpenDB [17], which is the OpenROAD project's database. Other enhancements and new features are described in the following.

Scalability was perhaps the most significant issue in the original FastRoute implementation that required fixing for use in OpenROAD. Initial limitations of FastRoute which have now been removed include:

- A net degree should be less than 1000.
- Designs should have at most eight routing layers.
- The preferred direction of routing layers was hardcoded.
- The routing grid was limited to a constant value, restricting the designs' area.
- The layer assignment had hardcoded pin layers.

The routing resources model in FastRoute was adapted from the ISPD 2008 Global Routing Contest to the OpenROAD flow environment. Beyond this, it was necessary to implement a correct calculation of "true routing resources" according to both *technology attributes* (e.g., spacing rules, transition layers, routing track pitches) and *design attributes* (e.g., routing obstacles, macro blocks, pin geometries). Correct understanding of routing resources allows the global router to avoid congested areas, creating quality results for the detailed router.

A critical functionality required in the OpenROAD project is fine-grain control of the global routing resource model. Passing a realizable global route to TritonRoute is important not only because of the DRC-clean output requirement, but also because OpenROAD's flow is currently non-reentrant and non-iterative. After computing routing resources ("supply") from technology and design attributes, FastRoute allows reduction of the resources on a per-layer and per-region basis. We have also performed studies and experiments to find proper resource configurations for each technology studied in the project, evaluating how different resource configurations impact quality of the final detailed routing solution.

Three new features have been added in the global router: clock net routing, antenna repair, and parasitics estimation. The **clock net routing** feature allows specifying a different configuration for clock nets. The *routing layer range* sets the minimum and maximum routing layers on which these nets will be routed, and the *routing topology* of these nets can be generated using FLUTE [2], the default Steiner tree constructor of FastRoute, or using PDRev [1]. The **antenna repair** feature is a preemptive approach to mitigate antenna rule violations [11]. The long routing segments that may create antenna violations in the detailed routing can be inferred in the global routing result. Therefore, we evaluate the global routing result to find nets with antenna violations and fix the violated nets by inserting diodes. The antenna checking/fixing flow involves calls to the antenna checker, netlist updates, and placement legalization of inserted diodes – but in our initial implementation it is driven by FastRoute. The **parasitics estimation** feature computes capacitance values for pins and wires of the nets using the global routing results. It is used by the Resizer tool (with OpenSTA) to identify and fix nets with maximum transition and maximum capacitance violations in the place-and-route flow. Finally, we have implemented an API that allows the integration of FastRoute with other tools inside the OpenROAD Project.

Most of the above issues and enhancements were identified during the tool's execution in the OpenROAD-flow. Often, the pending solutions were of highest priority. Thus, we have typically had only short periods to implement issue fixes and new features, leading to a backlog of missing documentation and poor implementation design choices. We are now improving the architecture of the tool with close guidance of a very experienced software engineer in the project, aiming for better maintainability and efficiency.

The future of FastRoute. We are currently using FastRoute only to generate routing guides for the detailed router, as well as the above-mentioned purposes. We intend to use FastRoute in other flow steps such as congestion- and timing-driven placement, and in (learning-enhanced) parasitic estimation during global placement through clock tree synthesis. We also intend to solidify the tool's support for different technology nodes, and implement any future enhancement that the OpenROAD project may require in the future.

4 Towards Extreme Partitioning

One of the foundations, or *base technologies*, for OpenROAD's scalability is the use of extreme partitioning to cope with design complexity. Partitioning tools break a given problem into smaller pieces that have some amount of independence and potential for parallelism. Core implementation algorithms can then process these subproblems individually, thus tackling runtime scaling.

4.1 Harnessing Clustering and Partitioning Tools

Over the years, many partitioning and clustering tools with different, but equally well-motivated, features have arisen. To access this diversity of techniques, we developed *partclusmanager*, a module in OpenROAD that integrates three well-known multilevel partitioning tools under a single Tcl API. Adapting three completely different frameworks to work under the same management tool was challenging, especially considering the chosen tools: (i) MLPart, a classic *hypergraph* partitioner implemented for VLSI CAD applications in the early 2000s; (ii) gpmetis [16], a highly regarded *graph* partitioner from the METIS family; and (iv) Chaco [14], a *graph* partitioner with many unique parameter options (e.g., architecture and terminal propagation).

Standardization. In VLSI CAD, netlists are naturally modeled using hypergraphs, where vertices are instances and hyperedges are nets. However, in many applications ranging from analytic placement to scientific computing, graph-based algorithms are much more common than hypergraph-based ones. Therefore, it is very common for the same hypergraph to graph decomposition algorithms to be implemented each time a new graph-based tool is to be used within the same research group. This generates possible discrepancies among the implementations and, consequently, among the achieved results. In PartClusManager, we offer graph customization options comprising different models (e.g., clique, star, hybrid), seven different formulas to calculate edge weights, and two options for vertex weights. This permits a standardized input for all partitioners managed by our tool, along with the possibility to dump the generated graph to a file for use in different tools. Besides input standardization, a common method to evaluate the results generated by different tools is needed. We implemented an evaluation function with three different assessment metrics (number of hyperedge cuts, number of terminals, runtime) to process the generated results. This feature can also be used to evaluate alternative partitioning solutions read from files.

Clustering. Each of the selected tools uses recursive coarsening as the first step in its overall multilevel partitioning process. We encapsulate the results of this step as a method for creating coarsened netlists, where we consider clusters as instances and infer their connection topology from the flat netlist. To achieve this, a deep understanding of the original third-party codes was needed to modify the implementation and extract these results without interfering with the usual flow of each tool.

4.2 Modularity-Driven Clustering

In OpenROAD, we have also studied a novel category of *modularity-driven* clustering algorithms. Unlike traditional VLSI clustering methods, modularity-driven clustering finds *natural clusters* in a graph using the *modularity* criterion. Thus, modularity-driven algorithms do not require user-input parameters such as target number of clusters, balance constraints, or coarsening ratio. In particular, the Louvain algorithm implements a fast and effective heuristic. In our studies, published in [3], we use Louvain to predict groups of logic gates that will stay together in the implementation flow. Comparing Louvain with traditional VLSI partitioning and clustering tools, we find that Louvain clusters better correlate with flat instance placements, using less runtime. We also propose a technique that performs a fast, "seeded blob-placement" of netlists based on modularity-driven clusters. This approach has shown up to 50% speedup in placement with less than 1% post-route wirelength degradation for multi-million instance netlists. We believe that such a technique can speed up placement of large netlists in OpenROAD in the future.

5 The OpenROAD Experience

In this section, we highlight important aspects of, and learnings from, working remotely for a large project. We describe how the OpenROAD project has evolved to facilitate the communication of multiple teams spread in many geographic locations and time zones. Some best practices and tools for task management and team communication are also noted. We conclude by discussing the importance of the OpenROAD automatic testing methodology that allows incremental modifications without breaking the OpenROAD flow – enabling what is called continuous software integration (CI).

5.1 Working Environment

The OpenROAD project has characteristics that differentiate it from a regular EDA project. We have stated it before: this is not research. However, this is not a company either. The OpenROAD project has a team composed of dozens of contributors, mostly students from different universities, who are located in different places around the world. The teams are divided according to the universities, and we also have a few experienced designers, researchers and programmers who give us support for tools, guidance in coding and design choices, and infrastructure management.

Our team in Brazil also has unique characteristics. We have had six students in our team during our participation in the OpenROAD project, where four of them were undergraduate students. Currently, our team is composed of two undergraduate students and one master's student. We must manage our time across university obligations, such as classes and exams, and the project tasks. The OpenROAD project has also been the first experience in a "real world" project for most of us. It is therefore the first step of our future careers, even when our perspectives are not aligned with the EDA area. The learnings we get from this large scale project in task management, team organization, and software development are valuable for any industry area where we eventually intend to work.

One of the most significant experiences we have from the project is working with people from all around the world. For most of our team, the OpenROAD project provided the first contact with people from other countries, with different backgrounds and knowledge. This has allowed us to see different points of view of academic research, understand industry expectations, and work with experienced engineers. However, we have also seen some conflicts with the different cultures and backgrounds of the project members. Sometimes, senior members of the project expect everything to be the way they *think* is correct, resulting in curt comments about our code. On the other hand, senior project members have been kind in willing to share their experience with us.

5.2 Teams Organization and Task Management

Due to OpenROAD being an international project, certain challenges come up when discussing organization and task management. Working with people from different countries, and therefore with different time zones, is a significant challenge, and especially impacts communication. Emails are a sure way of reporting tasks and issues, as well as keeping track of them. However, since there are no common work shifts, the number of email iterations can quickly rise, even among members of the same team. For example, no matter whether we are working in the morning or afternoon or night, communication to certain countries will be impossible.

In the beginnings of the OpenROAD project, email was more heavily used to define tasks and discuss implementations. Because of the issues presented above, as well as cc lists leading to confusion and lack of documentation, the OpenROAD project adopted Kanban-based project and issue tracking using Jira. Task documentation has become simpler and cleaner, issues are easy to find and organize, and threads can follow a single history that multiple team members can see and comment on. GitHub issues can also be used when discussing issues with the users of the OpenROAD app.

While task and issue tracking is now easy to work with and fairly straightforward, there is still some confusion with communication when dealing with different steps of the flow. This is for two reasons. First, there is not a strong team hierarchy concept in the project, since the universities and companies in the project each have their own "principal investigators" and internal management, which is not very transparent. Second, the project structure is contractually organized according to "Tasks" such as parasitic extraction or timing or placement (these three Tasks were originally spread across four different entities). The result is that an issue can stay on standby for quite some time without anyone assigned to it, and error reports for specific parts of the flow can end up with the same fate. This is where live meetings and emails come into play. When multiple members are discussing and screen-sharing issues together, tasks are assigned much faster and, consequently, are finished in less time, hopefully removing blockers for other team members.

5.3 Continuous Integration

The OpenROAD's flow is mainly composed of tools that are in constant need of expansions and improvements. Most of these tools were completely independent projects belonging to different work teams with distinct coding conventions. Each of these tools was modified to provide an integration friendly version. Additionally, multiple integrable versions were created with different features and levels of stability. Therefore, a unified repository was created to guarantee that all users have access to the most recent stable version of each tool. To guarantee the stability of this repository, both individual unit tests and top-level tests were proposed to be performed using an automation server to build and test software reliably. This would assure that the added modifications did not compromise the correctness of the extended tool or impacted the functionality of others.

We have seen that with the constant expansion of the tools for the OpenROAD project, the ideal repository came to be challenging to maintain. Continuing to synchronize the integrated and the standalone versions of the tools became less of a priority, as the original tools were barely used anymore. Managing dependencies between several repositories was also too much overhead. Our tools have been merged into one single repository to improve software quality and provide more standardized and organized code practices. Although monorepos are easier to break and more difficult to test, they are easier to maintain and can speed up the development process as features or components that are standard to multiple tools do not need to be implemented several times. As we write this, even though an automation server is properly integrated into our repository, it is not sufficiently robust and fail warnings may be deliberately ignored especially if project priorities have been changed. The OpenROAD project is slowly moving toward having a good continuous integration process, but much is yet to be improved and reorganized.

6 Conclusions

In this paper, we have presented the experience of being a contributor to the OpenROAD project from the perspective of foreign students. We discuss the nature of the OpenROAD project, highlighting the characteristics that differentiate it from typical academic research, such as software development standards and coordination of multiple teams. We describe the desirable skills to contribute to the project, such as programming skills and VLSI experience, and the challenges and workarounds we face by working from abroad, especially regarding infrastructure and technology access.

We also describe our contributions to the OpenROAD project, which include four tools used at the floorplan, clock tree synthesis, and global route stages of the OpenROAD-flow, as well as two additional tools that can underlie the extreme partitioning strategy. We also describe considerations inherent in open-sourcing academic code for the design of production ICs, and development of brand-new tools for subjects neglected in the VLSI CAD literature.

Finally, we discuss our experiences and learnings from working on a large scale project. We emphasize that the OpenROAD project is not normal academic research, and it is not a company either. We have several teams composed of students from different universities, and a few engineers with industry experience. We present our team's unique characteristics and relations with other project teams, especially with the senior members. We describe the challenges and solutions for team organization and task management, showing the evolution of communication and documentation methods and the open issues we still face. Last, we describe methodology for the development and maintenance of our tools using unit tests and top-level tests, to improve code stability and reliability.

Our team plans to keep contributing to the OpenROAD project in the best ways possible. We are currently supporting the five tools described in this work. The abovementioned improvements of FastRoute, which achieve better maintainability and development efficiency, are being propagated to all the tools that we have worked on. And, a near-term roadmap of functionality is clear – e.g., ioPlacer requires enhancements for timing-driven multi-layer I/O pin assignment; TritonCTS must improve its ability to meet strict timing requirements; and the partitioning tools under partclusmanager need to be integrated with the OpenROAD flow as it provides more support to chip- and block-level planning use cases. We hope that this paper will motivate more talents to join OpenROAD, which would in itself expand our contributions to the project.

Acknowledgments

This work was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, CNPq, FAPERGS and by U.S. DARPA (HR0011-18- 2-0032).

References

[1] C. J. Alpert, W.-K. Chow, K. Han, A. B. Kahng, Z. Li, D. Liu and S. Venkatesh, "Prim-Dijkstra Revisited: Achieving Superior Timing-driven Routing Trees", *Proc. ISPD*, 2018, pp. 10–17.
[2] C. Chu and Y.-C. Wong, "FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design", *IEEE TCAD*, 27 (2008), pp. 70–83.
[3] M. Fogaça, A. B. Kahng, E. Monteiro, R. Reis, L. Wang and M. Woo, "On the Superiority of Modularity-Based Clustering for Determining Placement-Relevant Clusters", *Integration: The VLSI Journal* 74 (2020), pp. 32–44.
[4] V. Bandeira, M. Fogaça, E. Monteiro, I. Oliveira, M. Woo and R. Reis, "Fast and Scalable I/O Pin Assignment with Divide-and-Conquer and Hungarian Matching", *Proc. NEWCAS*, 2020, pp. 1–4.
[5] K. Han, A. B. Kahng and J. Li, "Optimal Generalized H-Tree Topology and Buffering for High-Performance and Low-Power Clock Distribution", *IEEE TCAD*, 2020, pp. 478–491.
[6] M. Guiney and E. Leavitt, "An Introduction to Openaccess an Open Source Data Model and API for IC Design", *Proc. ASPDAC*, 2006, pp. 434–436.
[7] S. Mantik, G. Posser, W.-K. Chow, Y. Ding, and W.-H. Liu, "ISPD 2018 Initial Detailed Routing Contest and Benchmarks", *Proc. ISPD*, 2018, pp. 140–143.
[8] M. Pan and C. Chu, "FastRoute: A Step to Integrate Global Routing into Placement", *Proc. ICCAD*, 2006, pp. 464–471.
[9] M. Pan and C. Chu, "FastRoute 2.0: A High-quality and Efficient Global Router", *Proc. ASPDAC*, 2007, pp. 250–255.
[10] M. Pan, Y. Xu, Y. Zhang and C. Chu, "FastRoute: An Efficient and High-Quality Global Router", *ACM VLSI Design*, 2012, pp. 14:1–14:1.
[11] H. Shin, C. King and C. Hu, "Thin Oxide Damage by Plasma Etching and Ashing Processes", *Proc. IRPS*, 1992, pp. 37–41
[12] Y. Xu, Y. Zhang and C. Chu, "FastRoute 4.0: Global Router with Efficient Via Minimization", *Proc. ASPDAC*, 2009, pp. 576–581.
[13] Y. Zhang, Y. Xu and C. Chu, "FastRoute 3.0: A Fast and High Quality Global Router Based on Virtual Capacity", *IEEE/ACM ICCAD*, 2008, pp. 344–349.
[14] B. Hendrickson and R. Leland, "The Chaco User's Guide", https://prod-ng.sandia.gov/techlib-noauth/access-control.cgi/1995/952344.pdf
[15] FastRoute, http://home.eng.iastate.edu/~cnchu/FastRoute.html
[16] G. Karypis and V. Kumar, "METIS - Unstructured Graph Partitioning and Sparse Matrix Ordering System, V. 2.0", https://dm.kaist.ac.kr/kse625/resources/metis.pdf
[17] OpenDB, https://github.com/The-OpenROAD-Project/OpenDB
[18] OpenROAD, https://github.com/The-OpenROAD-Project/OpenROAD
[19] The OpenROAD flow, https://github.com/The-OpenROAD-Project/OpenROAD-flow
[20] TritonRoute, https://github.com/The-OpenROAD-Project/TritonRoute
[21] Yosys, https://github.com/The-OpenROAD-Project/yosys