ARL:UT's Experiences in the Free Open-Source VLSI EDA Landscape

Russell Friesenhahn
The Applied Research Laboratories
10000 Burnet Rd
Austin, Texas 78758
russellf@arlut.utexas.edu

Johnathan York
The Applied Research Laboratories
10000 Burnet Rd
Austin, Texas 78758
york@arlut.utexas.edu

ABSTRACT

Historically at ARL:UT, we have almost exclusively developed digital applications on commercially available CPUs and FPGAs. However, our evolving application set demands higher performance with lower Size, Weight, and Power (SWaP) than commercially available CPUs or FPGAs provide. This has driven us to ASIC design. We have noticed that fabrication costs of "obsolete" technology nodes have fallen, yet the cost of commercial EDA tools remains agnostic of the targeted technology node. This has pushed us to explore the Free and Open-Source (FOSS) VLSI EDA tools. We will discuss our experiences in using and collaborating with the design flow provided by Qflow, which works well for the intended design space. We then present our thoughts on what would benefit the FOSS VLSI EDA community, and why we believe the development tools we choose should be as FOSS as possible.

1 INTRODUCTION

Applied Research Laboratories, The University of Texas at Austin (ARL:UT) is a University-Affiliated Research Center (UARC) that has performed research and development for United States Government sponsors since 1945 [7]. Historically, we have almost exclusively developed digital applications on commercially available CPUs and FPGAs. However, our evolving application set demands higher performance with lower Size, Weight, and Power (SWaP) than is available in either commercially available CPUs or FPGAs. This has led us to explore ASIC design options.

We have noticed that while Moore's Law marches on to single digit nm feature sizes, the financial barriers to entry of "obsolete" technology nodes have fallen providing researchers, startups, and hobbyists unprecedented access. However, total cost of ownership for commercial electronic design automation (EDA) tools remains agnostic of the targeted technology node. Thus while access to fabrication is affordable, access to design is not. This has pushed us to explore the Free and Open-Source (FOSS) VLSI EDA tools.

As both a University and as a UARC, ARL:UT is chartered to serve the public interest, and so the use of and contribution to FOSS tools open to all is a natural course for us. If we believe a tool is not working as expected, we dig into the source to find and fix the problem. Our belief in FOSS is seen in our use of Linux, Python, NumPy, SciPy, KiCad, Icarus Verilog, and in our long history of contributions back to the community, among them the GPS Toolkit [9] [4] and Ganymede [6] [3].

Using internal R&D funds, we have spent the previous two years working with the FOSS VLSI EDA ecosystem automated by Qflow. We will discuss our experiences with Qflow and related tools and our contribution to the effort. We then present our thoughts on what would benefit the FOSS VLSI EDA community and why we believe the development tools we choose should be as FOSS as possible.

WOSET 2018, San Diego, CA

2 THE QFLOW ECOSYSTEM

After evaluating the available FOSS VLSI EDA tools, we selected Qflow because the flow receives heavy development to expand the feature set, maintainers of several of the tools are active in multiple forums and platforms and open to community collaboration, the flow uses standard file formats whenever possible, and we found it incredibly quick and easy to stand-up the tools and execute the RTL to GDSII process on a small circuit.

Qflow mostly automates a textbook design flow of taking an RTL design to physical GDSII output. Table 1 lists the tool that implements each design flow process. OpenRAM provides an SRAM memory compiler capability and is included in this list even though it is not used in Qflow because of the criticality of this feature. We argue without this, a large set of designs would be unrealizable.

Many of the individual tools have some idiosyncrasies, but, in reality, each tool is typically the only option if a fully automated flow is desired. For this reason, we will provide evaluation of the tools as a suite.

2.1 Qflow Efficacy

Most of the tools were originally developed individually, some decades in time apart, and for a particular function or project without the present design flow in mind. However, since the tools are FOSS and naturally use openly defined file formats or industry-standard file formats, the tools interact with each other without issue. And where one tool provides an output format that the next tool in the process chain does not natively accept as an input, Qflow provides a glue script for translation.

Overall, the design flow works excellently, especially when using the flow as the Qflow commands use it. A designer can easily and reliably take a Verilog RTL design, run it through Qflow, and receive GDSII output of the design. Furthermore, if an engineer requires more control over individual steps, a custom flow can be created using a combination of a Makefile and Qflow which we have done

Table 1: Summary of design flow processes and the associated FOSS tools

Design Flow Process	FOSS Tool
Behavioral Simulation	Icarus Verilog
RTL Synthesis	Yosys
Gate-Level Simulation	Icarus Verilog
Logic Optimization	ABC
Technology Mapping	ABC
Placement	Graywolf
Routing	Qrouter
Static Timing Analysis	Vesta
Physical Layout	Magic
Design Extraction	Magic
Design Rule Check	Magic
Layout Versus Schematic	Netgen
GDSII Output	Magic
Misc. "Glue" Scripts	Qflow
Memory Compiler	OpenRAM

for our project. This approach enables maximum flexibility while leveraging as much of Qflow's ability to automate the design flow.

Also encouraging, the design flow does not have an upper limit on the design size, at least that we have encountered. Anecdotally, we are aware of other design teams using the tools to create much larger designs than we have, to date, created, including small processor designs and systems-on-a-chip. The trade-off we have noticed is, not unexpected, the design runtimes. Placement, routing, and layout extraction have significant higher execution times as the design size increases. However, we have seen effort put towards eliminating performance bottlenecks in the past 12 months.

2.2 Qflow Limitations

Despite working well together, the design flow does have some limitations. For starters, only 0.5μm to 180nm fabrication processes are currently supported. This has not been fully investigated, but we believe that almost all the tools would support the smaller feature sizes except Magic's DRC function. As we all know, as fabrication processes shrink in size, new DRC rules arise to properly constrain the design to accommodate the new fabrication tools' limitations. Magic has software support for DRC rules down to the typical 180nm processes. Beyond that, new software development is required to support new fabrication processes. In summary, the design flow could immediately use smaller fabrication processes at the risk of not being able to fully verify the manufacturability of the completed design.

Another shortcoming that we have encountered is a lack of support for hierarchical design. In all fairness, this is not the original goal of the design flow and Qflow, but we believe this capability is a natural progression that the tools must support to remain competitive.

2.3 Qflow Final Thoughts

In summary, there exists a FOSS VLSI EDA design flow implemented by the tools listed in Table 1. ARL:UT has been using the flow, and it works reasonably well. The development community surrounding Qflow appears strong: tools maintainers are responsive to bug reports, continue to add new features, are open to coordination. At least one start-up company has based much of their backend technology on the design flow presented by Qflow.

3 ARL:UT'S CONTRIBUTIONS

Beyond developing in-house ASIC designs, our project's goals include participating and contributing to the open-source community. In addition to providing bug reports, we have contributed back with two features which we will expand upon.

(1) Add interconnect delay to Vesta Static-Timing Analysis in Qflow
(2) Create a Continuous Integration test for the design flow

3.1 Interconnect Delay

This internal R&D project originated to further Johnathan York's dissertation work, which evaluates the trade-off between a minimally reconfigurable circuit design and performance [10]. Depending on the design space region, interconnect delay can be a dominating factor in a design's performance. Interconnect delay is the amount of time required for a signal to propagate its value from a gate's output to the input(s) of the downstream gate(s). Essentially, this is the effect the wires have on the circuits operational speed.

The easiest method to determine the interconnect delay's effect on the design is to include the interconnects' delay values in the static timing analysis (STA) which provides the worst case operating frequency for a design. As shown in Table 1, Vesta is the STA tool included in Qflow. However, at the outset of this effort, Vesta did not take interconnect delay values as an input into its processing. Furthermore, the design flow did not have a reliable method to calculate interconnect delay.

In coordination with the maintainer of Qflow and Qrouter, we planned a development path to include interconnect delay in the STA process. First, Qrouter's maintainer expanded the functionality such that when a route is completed, Qrouter would also model the resulting wire as a resistor-capacitor (RC) tree. The RC tree information is then written to a text file in a newly specified format.

We developed software to ingest the RC tree file, calculate the Elmore delay of each interconnect, and write the result to a file in a format that Vesta understands. Vesta was then modified to accept the delay file as an optional input and to use the interconnect delay values when performing STA on the related net. This feature addition provided the design flow with a first-order approximation of the effect interconnect delay has on a routed design.

3.2 Continuous Integration

The design flow uses at least ten FOSS tools maintained by at least five different groups or individuals. Though the tools' developers do work together when issues arise, it is challenging for them to maintain full compatibility at all times. New patches and features are constantly being added to the most heavily developed tools. Though a few of the tools have extensive regression testing, the entire design flow does not which would ostensibly fall under the *Qflow* repository. Though this constant state of flux gave us confidence in the active development of the tools, it also presented a challenge in

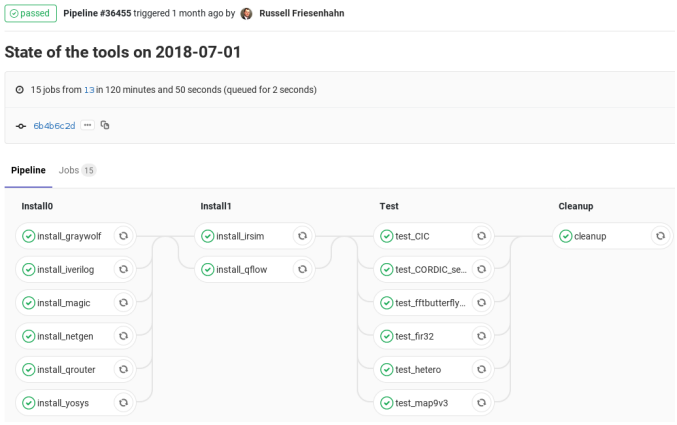

Figure 1: Gitlab Continuous Integration Pipeline Display

using the tools. With each problem encountered, the question arose, "are we debugging our design or the design flow?" We realized we needed the following:

(1) a method to version all of the tools as a single unit
(2) a regression test for the design flow
(3) the automation (as much as possible) of the first two

The solution to the first requirement was to gather all the tools as submodules into a Git superproject. Within the superproject, we create tags when a stable point across all the tools is found. Stables points are identified by a successful regression test which is also part of the superproject. The regression test consists of the design flow correctly operating on five Verilog RTL designs from synthesis through a passing LVS run. Though certainly not exhaustive, the regression test has found issues in the design flow operation which are fed upstream to the maintainers. Many times, the responsible maintainer has a patch committed within 1-2 days.

To implement continuous integration (CI), we created scripts to automate the install of the tools and the operation of the regression test. A user runs another script to update the superproject with new commits from the submodules. Upon committing to the superproject, the built-in CI feature in GitLab takes over. The tools are installed at the superproject commit under test and the regression test is executed. The steps of the CI process are shown in Figure 1 which is a screenshot of the GitLab website for the superproject. We then create a tag if the regression test is successful or begin debugging upon regression failure. In addition to maintaining this superproject internally, we also host it on GitLab.com calling it *asic_tools* [2].

3.3 Future Work

There are a lot of tools out there, and we expect that number to grow. Ensuring that all those tools work together and remain compatible as they grow in features is how FOSS EDA tools grow and mature as a viable option. This is not a trivial endeavor.

We see that we can continue to refine and grow the continuous integration framework. We welcome the addition of new tools so that developers can be assured that not only does their tool work standalone, but it works on a standard suite of test designs, works in conjunction with other tools, and does so not only today, but continues to work tomorrow. However, the greatest benefit from this type of testing is learning immediately when tools have diverged. This knowledge can be acted on before it becomes too costly to maintain compatibility.

4 RECOMMENDATIONS

4.1 Success at Every Stage

We are encouraged and optimistic by the OpenROAD [8] effort led by UC San Diego. The desired outcome of complex ASIC and SoC design in under 24 hours with no human-in-the-loop is impressive. To achieve this, we imagine much of the standard design flow may be turned on its head. However, we also believe that this new approach will have steps that overlap with the traditional methods. We suggest that the tools developed that reach towards the new might also be useful for plain ol' regular VLSI design as well. One of our frequent government sponsors has a saying for project planning: Success at every stage. Though the intended goal is a turnkey process, we believe it would be beneficial to the community for the developed tools to support stand-alone operation as well as using standard file formats. Perhaps this is already the planned approach as we have seen several tools pushed up to the OpenROAD GitHub account [5] already. Nevertheless, we wanted to express our desire for this.

4.2 As FOSS as Possible

Answering the question of "what is open-source?" in the software realm is relatively simple: Use developer tools that have a compatible FOSS license and apply an appropriate license to the developed software. This process is as pure FOSS as possible from beginning to end. Unfortunately, this is simply not the case in ASIC development.

At the very least, due to intellectual property (IP) protections secured via non-disclosure agreeements (NDAs), the fabrication process is not open. It is unique to the chosen fabricator also making the final design output not portable. With the *de facto* end of MOSIS' Scalable CMOS about ten years ago as the technology scaled into very deep submicron, the standard cell libraries are also under NDA protection. A standard cell library, to be compatible with a fabrication process, almost certainly requires IP knowledge of that same process. With the cell library in hand, a full design can be implemented using tools of the engineer's choice. However, to get to this point, the team or its organization had to negotiate and sign one or more NDAs. Depending on the organization, this is a significant barrier (and sometimes an impossible one as one of our team members experienced at a previous organization).

At least one start-up company, efabless [1], has recognized that this barrier exists for academics, hobbyists, and fellow small start-ups and has worked around it. Efabless provides a cloud platform of FOSS design tools that target a contracted fabricator's process. The designer never sees or has access to the GDSII which eliminates the IP access issue thus obviating an NDA. This may be an excellent option for a hobbyist but is not acceptable for us. We need to have access to and own our final design products.

For this reason, we have embraced using FOSS for all pieces after the cell library for maximum long-term flexibility. We believe that the benefits of FOSS also apply to memory compilers, metal fill

tools, and similar "peripheral" design support tools. Mainly that FOSS enables an organization to more fully own their completed designs.

5 CONCLUSION

In the past twenty-four months, ARL:UT has evaluated the available FOSS VLSI EDA tools and selected a design flow we have the most confidence in. We have contributed new features, discovered software bugs, and provided patches upstream. We have configured and now manage a continuous integration process to facilitate testing updates to the tools. Though the tools have some idiosyncrasies, through development, testing, and curating expertise in operating the tools, we are confident in our ability to create in-house ASIC designs and plan to exercise that capability through to fabrication over the next 12 months.

We are excited by the OpenROAD goals and are anxious to see the project's results. If possible, we believe keeping the old design methods in mind when developing tools for OpenROAD will greatly assist the current FOSS VLSI EDA tools.

We look forward to collaborating with others towards the advancement of free and open source VLSI EDA tools and are actively soliciting ways to do so. Specifically, we're looking to

- provide early-adopter feedback on new tools using designs we're currently working on
- facilitate setting up continuous integration, regression testing, and compatibility testing with old and new tools

BIOGRAPHY

Russell Friesenhahn is an Engineering Scientist at the Applied Research Laboratories (ARL:UT) since 2014. He holds a B.S. in Electrical Engineering from the University of Texas at Austin and a M.S. in Electrical Engineering from University of Southern California. Mr. Friesenhahn specializes in digital design targeting FPGAs and ASICs. He has been the lead designer on five research ASICs since 2009.

Johnathan York received a BS, MS, and a Ph.D in Electrical Engineering at the University of Texas at Austin. He has worked at the Applied Research Laboratories (ARL:UT) since 2001, working primarily with high-throughput real-time Digital Signal Processing applications. Dr. York's research focus is heterogeneous reconfigurable computing.

ACKNOWLEDGEMENTS

Russell and Johnathan would like to acknowledge their student interns whom without this work would not have been possible. Michael Koger Darden for the first iteration of the continuous integration setup and regression test. Alihussein Momin for his investigation in hierarchical design options. Shrey Sachdeva and Timberlon Gray for their work on our first ASIC design.

REFERENCES

[1] 2018. efabless.com. (2018). Retrieved August 28, 2018 from https://efabless.com
[2] 2018. FOSS Hardware Tools - asic_tools. (Aug 2018). Retrieved August 28, 2018 from https://gitlab.com/fossht/asic_tools
[3] 2018. Ganymede Network Directory Management System. (Aug 2018). Retrieved August 28, 2018 from https://github.com/jonabbey/Ganymede
[4] 2018. GPS Toolkit. (Aug 2018). Retrieved August 28, 2018 from https://github.com/SGL-UT/GPSTk
[5] 2018. OpenROAD (abk). (2018). Retrieved August 28, 2018 from https://github.com/abk-openroad
[6] Jonathan Abbey and Michael Mulvaney. 1998. Ganymede: An Extensible and Customizable Directory Management Framework.. In *LISA*. 197–218.
[7] ARL:UT. 2018. About Us. (Aug 2018). Retrieved August 28, 2018 from http://www.arlut.utexas.edu/about.html
[8] UC San Diego. 2018. OpenROAD - Foundations and Realizatio of Open and Accessible Design. (Aug 2018). Retrieved August 28, 2018 from http://theopenroadproject.org
[9] Tom Gaussiran, D Munton, Ben Harris, and B Tolman. 2004. An open source toolkit for GPS processing, total electron content effects, measurements and modeling. In *International Beacon Satellite Symposium, Trieste, Italy.*
[10] Johnathan York. 2011. *Multiple personality integrated circuits and the cost of programmability.* Ph.D. Dissertation. The University of Texas at Austin.