Unleashing Electronics Prototyping (and beyond): computational design for circuit boards

Richard Lin*

University of California, Los Angeles, richardlin@ucla.edu

Rohit Ramesh

University of California, Berkeley, rkr@berkeley.edu

Prabal Dutta

University of California, Berkeley, prabal@berkeley.edu

Björn Hartmann

University of California, Berkeley, bjoern@berkeley.edu

Ankur Mehta

University of California, Los Angeles, mehtank@ucla.edu

While modern electronics prototyping systems – both breadboard-based and toolkits – have enabled even novices to be productive and able to build functional devices quickly, they also impose a low ceiling on capability by being limited to the ecosystem of breadboard- or toolkit-compatible parts. However, inspired by the success of software engineering and its library-driven development flow in being novice-friendly and scaling up to complex applications, we instead examine a hardware description language (HDL) approach to electronics design that can enable user-created libraries and user-defined design automation to make electronics design easier and more efficient while offering the flexibility and capability of custom circuit boards. In this position paper, we recap our prior work on this HDL and discuss extensions to better support prototyping as well as explore the broader trade-off space of electronics design tools.

CCS CONCEPTS • Hardware ~ PCB Design and Layout, Hardware description languages and compilation

Additional Keywords and Phrases: printed circuit board (PCB) design, circuit design, hardware description language (HDL)

ACM Reference Format:
Richard Lin, Rohit Ramesh, Prabal Dutta, Björn Hartmann, and Ankur Mehta. 2023. Unleashing Electronics Prototyping (and beyond): computational design for circuit boards.

This work was presented at the **CHI2023 Workshop [WS2] - Beyond Prototyping Boards: Future Paradigms for Electronics Toolkits**
CHI '23, April 23-28, 2023, Hamburg, Germany

1 INTRODUCTION

While modern electronics prototyping systems – commonly a combination of solderless breadboards, microcontroller development boards like Arduino, breakout boards like screens and motor drivers, and plug-and-play ecosystems like Qwiic and Jacdac [1] – have been transformative by enabling novices to build custom devices with a very low threshold to entry, they also impose a low ceiling on capability. Breadboards are large and fragile, which makes them unsuitable for where formfactor is a concern. Furthermore, breakout boards are only available for a tiny subset of all available electronics components. And as each device must be painstakingly hand-built, this becomes infeasible in even low volume production.

Typically, the next step beyond breadboarding is building a custom printed circuit board (PCB), though this can be daunting. While PCB fabrication today is affordable enough to be used for prototyping by even hobbyists, and open-source yet user-friendly design tools exist, a lot of knowledge is still required to produce a working board. No longer is the low-level electronics expertise encapsulated into an off-the-shelf breakout board, but the user must instead read datasheets and find and transcribe reference schematics. Plenty of gotchas also lie in wait for those not familiar with these lengthy datasheets and detailed tables, such as voltage limits and IO thresholds. And overall, PCB tools don't help with the actual circuit design – they're more computer-aided drafting tools for drawing schematics.

In contrast, the software development ecosystem offers a brighter vision for how design could not only be easier for novices, but also be more efficient for experts. While user-friendly programming tools and resources like StackOverflow help, much of the power comes from libraries which encapsulate significant complexity and functionality into easy-to-use packages. Furthermore, because of this separation of interface and implementation, even novice programmers can make use of these without understanding the (often significant) details of what they do under the hood.

We aim to bring that kind of power to hardware design by building design tools around these software engineering ideas. In the rest of this position paper, we recap our thread of work on a hardware description language (HDL) approach for PCB design to blend the best of software and hardware engineering, examine where it fits into the larger picture of device prototyping and design, and speculate on how this could be expanded in the future to not only further lower the threshold of entry but also raise the ceiling on capability.

2 RELATED WORK

Much of the academic work on board-level electronics focuses on supporting novices through tools for breadboards. These include design tools [6, 14] which blend aspects of schematic and breadboard design, built-in instrumentation [2] to aid debugging, and circuit synthesis tools [5] that produce breadboarding diagrams. However, as discussed earlier, breadboards impose significant limits on capability.

Academic work is more limited beyond breadboarding, though there is some needs-finding of bridging prototype to production [11] and a thread of work on synthesizing boards from high-level specifications [10, 3]. Some recent commercial work [13, 4] also examines high-level PCB design with a library-based modular block-based-design approach, though lack of focus on user library creation can also be limiting.

3 OUR APPROACH: HARDWARE DESCRIPTION LANGUAGE

In contrast, our aspiration is to pick three of three: low threshold (usable by novices, suitable for prototyping), high ceiling (scales to complex, production designs), and wide walls (useful for a wide range of applications) [12]. Since modern programming languages balance all of the above, we built a hardware description language for PCBs[1] in prior work [7, 8].

[1] Open-source at https://github.com/BerkeleyHCI/PolymorphicBlocks

Many software engineering concepts actually translate well to hardware design: for example, schematics are commonly designed with reusable subcircuits, which maps well to a library-based flow. As these subcircuits often need to be customized for each application, such as by modifying component values (e.g., different resistance for different input voltages), choosing different parts (e.g., 0603 vs. through-hole resistor), or even changing the structure (e.g., a 2x2-LED matrix instead of a 4x4 matrix), the conceptual reusability of the abstract circuit often does not translate into actual reuse of schematic files, since schematics in mainstream tools are static. By instead defining the circuit as a block of code via a hardware description language, the block of code can automatically generate the internal circuit details in a way that is responsive to parameters (e.g., LED matrix dimensions) or environment (e.g., calculate resistance based on input voltage).

Furthermore, the idea of "jellybean parts" (a generic component for which compatible parts are widely available, e.g., 1kOhm resistor) can be formalized with a type system where abstract supertypes represent the high-level generic type while concrete subtypes are actual purchasable parts. Using abstract parts in libraries further enhances reusability and generality, preserving the option of concrete compatible parts. The type hierarchy can also be helpful to novices, who may be more comfortable with these high-level specifications while the system then resolves ambiguity through smart defaults.

The HDL can then be compiled down to a netlist, which defines the components and connectivity and can be imported into a mainstream PCB layout tool like KiCad. Over the years, we've built increasingly complex boards with higher automation, as shown in Figure 1.

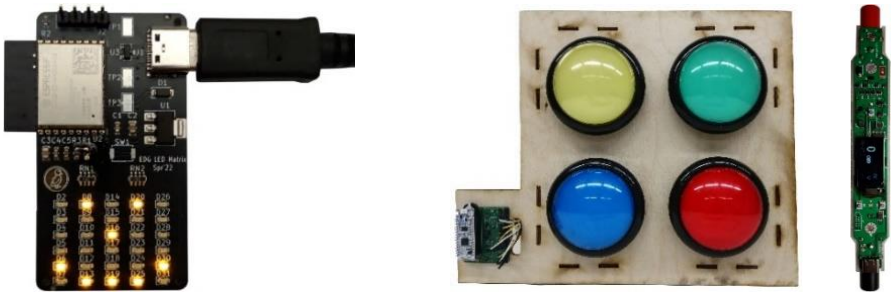

Figure 1: Example boards we have built in our HDL system. On the left is an IoT charlieplexed LED matrix, which enables the low-pin-count ESP32C3 microcontroller to drive a 5x6 grid of LEDs. The charlieplexing LED matrix library block encapsulates the details of the nontrivial circuit topology generator and packages it to be easily used by non-experts with a width and height parameter. Internal, lower-level blocks like resistors and capacitors are similarly generators that can automatically select matching parts from a table. On the right are two other boards we've built: a Simon memory game demonstrating a 5-to-12-volt boost converter generator to power the large dome button LEDs, and a compact multimeter demonstrating custom analog circuits.

Supporting tooling as an IDE [9] with block diagram visualizer and supporting schematic-like edit actions (e.g., insert block, connect ports) that generate corresponding HDL further helps working with this representation.

4 FUTURE WORK

While this HDL-based PCB design flow produces boards with higher-level design inputs and greater automation, future work can extend this to better support novice users and prototyping.

4.1 Bridging Prototyping and PCBs

Fundamentally, what we've built is a circuit design tool that supports user-defined automation and user-defined libraries. That it generates into PCBs is somewhat of an implementation detail: there's no reason an output couldn't be a

breadboarding diagram and a list of breakout boards to be purchased. After physical prototyping and testing, the same high-level design might be automatically specialized to a PCB for a production run or higher-fidelity prototype.

Extending this, such a system might also generate supporting artifacts like consistent hardware abstraction layer code for both prototyping and production platforms, further easing the transition by enabling the use of similar firmware.

4.2 Prototype-focused Tooling

While an HDL is powerful by enabling user-defined automation, code can be less intuitive and discoverable compared to modern graphical tools or prototyping with a collection of physical parts. However, this system can still usefully serve as infrastructure for more novice- and prototype-focused tooling, enabling experts to define libraries while more casual users can use those from a graphical frontend like existing modular design tools.

Furthermore, even breadboarding requires some of the same skills as a PCB designer would, such as understanding the functionality of breakout boards. The concept of a parts type system defines a *design space*, a set of all satisfying designs, through choices of subtypes to implement each abstract type. This idea could perhaps also extend to non-PCB prototyping tools, which allows users to define designs at a higher, ambiguous, level, while the system helps users make those more concrete and navigate trade-offs.

4.3 Overall Trade-off Space

In contrast to much existing work that takes a bottom-up approach by focusing on novices, we take a center-out approach by focusing on intermediate designs and incrementally expanding support for both novices and more complex designs. We believe that custom PCBs offer the widest capability while having a reasonable (though still significant) learning curve. In Figure 2, we explore the rough conceptual trade-off spaces of various electronics design tools, looking at metrics like learning curve, capability, and latency – and we hope that our approach is a 'happy medium' instead of a 'no man's land'.

But beyond these, what other trade-offs and objectives are important? Is there a one-size-fits-all solution, or does electronics design necessarily need to be a landscape of fragmented (but interoperating) tools? Or perhaps better interoperability is how we can bridge the gap between the different levels of prototyping and even production?

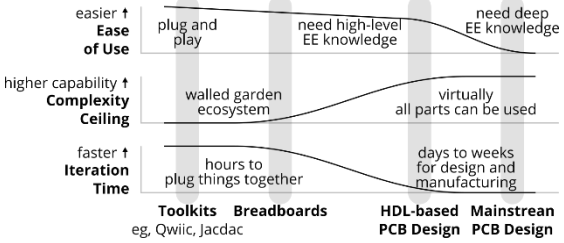

Figure 2: Rough concept of the trade-off space between toolkit-based design, breadboarding, HDL PCB design, and conventional PCB design. Our hope is that this design approach enables most of the capability of PCBs while being significantly easier to design with.

ACKNOWLEDGMENTS

This work was supported in part by DARPA grants HR00112110008 and FA8750-20-C-0156 (SDCPS). The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, nor does this imply any official endorsement. Approved for public release; distribution is unlimited.

REFERENCES

[1] James Devine, Michal Moskal, Peli de Halleux, Thomas Ball, Steve Hodges, Gabriele D'Amone, David Gakure, Joe Finney, Lorraine Underwood, Kobi Hartley, Paul Kos, and Matt Oppenheim. 2022. Plug-and-Play Physical Computing with Jacdac. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 6, 3, Article 110 (sep 2022), 30 pages. https://doi.org/10.1145/3550317

[2] Daniel Drew, Julie L. Newcomb, William McGrath, Filip Maksimovic, David Mellis, and Björn Hartmann. 2016. The Toastboard: Ubiquitous Instrumentation and Automated Checking of Breadboarded Circuits. In Proceedings of the 29th Annual Symposium on User Interface Software and Technology (Tokyo, Japan) (UIST '16). ACM, New York, NY, USA, 677–686. https://doi.org/10.1145/2984511.2984566

[3] Jorge Garza, Devon J. Merrill, and Steven Swanson. 2021. Appliancizer: Transforming Web Pages into Electronic Devices. In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (Yokohama, Japan) (CHI '21). Association for Computing Machinery, New York, NY, USA, Article 415, 13 pages. https://doi.org/10.1145/3411764.3445732

[4] Gumstix. 2018. Geppetto. www.gumstix.com/geppetto/

[5] Rushil Khurana and Steve Hodges. 2020. Beyond the Prototype: Understanding the Challenge of Scaling Hardware Device Production. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. 1–11.

[6] André Knörig, Reto Wettach, and Jonathan Cohen. 2009. Fritzing: A Tool for Advancing Electronic Prototyping for Designers. In Proceedings of the 3rd International Conference on Tangible and Embedded Interaction (Cambridge, United Kingdom) (TEI '09). ACM, New York, NY, USA, 351–358. https://doi.org/10.1145/1517664.1517735

[7] Richard Lin, Rohit Ramesh, Connie Chi, Nikhil Jain, Ryan Nuqui, Prabal Dutta, and Björn Hartmann. 2020. Polymorphic Blocks: Unifying High-Level Specification and Low-Level Control for Circuit Board Design. In Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology (Virtual Event, USA) (UIST '20). Association for Computing Machinery, New York, NY, USA, 529–540. https://doi.org/10.1145/3379337.

[8] Richard Lin, Rohit Ramesh, Prabal Dutta, Bjoern Hartmann, and Ankur Mehta. 2022. Computational Support for Multiplicity in Hierarchical Electronics Design. In Proceedings of the 7th Annual ACM Symposium on Computational Fabrication (Seattle, WA, USA) (SCF '22). Association for Computing Machinery, New York, NY, USA, Article 1, 11 pages. https://doi.org/10.1145/3559400.3561997

[9] Richard Lin, Rohit Ramesh, Nikhil Jain, Josephine Koe, Ryan Nuqui, Prabal Dutta, and Bjoern Hartmann. 2021. Weaving Schematics and Code: Interactive Visual Editing for Hardware Description Languages. In The 34th Annual ACM Symposium on User Interface Software and Technology (Virtual Event, USA) (UIST '21). Association for Computing Machinery, New York, NY, USA, 1039–1049. https://doi.org/10.1145/3472749.3474804

[10] Devon J. Merrill, Jorge Garza, and Steven Swanson. 2019. Echidna: Mixed-Domain Computational Implementation via Decision Trees. In Proceedings of the ACM Symposium on Computational Fabrication (Pittsburgh, Pennsylvania) (SCF '19). Association for Computing Machinery, New York, NY, USA, Article 5, 12 pages. https://doi.org/10.1145/3328939.3329004

[11] Rohit Ramesh, Richard Lin, Antonio Iannopollo, Alberto Sangiovanni-Vincentelli, Björn Hartmann, and Prabal Dutta. 2017. Turning Coders into Makers: The Promise of Embedded Design Generation. In Proceedings of the 1st Annual ACM Symposium on Computational Fabrication (Cambridge, Massachusetts) (SCF '17). ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3083157.3083159

[12] Mitchel Resnick, Brad Myers, Kumiyo Nakakoji, Ben Shneiderman, Randy Pausch, Ted Selker, and Mike Eisenberg. 2005. Design principles for tools to support creative thinking. (2005).

[13] Sparkfun. 2020. À La Carte. https://alc.sparkfun.com/

[14] Chiuan Wang, Hsuan-Ming Yeh, Bryan Wang, Te-Yen Wu, Hsin-Ruey Tsai, Rong-Hao Liang, Yi-Ping Hung, and Mike Y. Chen. 2016. CircuitStack: Supporting Rapid Prototyping and Evolution of Electronic Circuits. In Proceedings of the 29th Annual Symposium on User Interface Software and Technology (Tokyo, Japan) (UIST '16). ACM, New York, NY, USA, 687–695. https://doi.org/10.1145/2984511.2984527