Parametric Circuit Design Made Accessible Through Programming and Bidirectional Interfaces

LEO F MCELROY[1]

Independent Researcher, Massachusetts, leomcelroy@gmail.com

QUENTIN H BOLSEE

Vrije Universiteit Brussel / MIT Center for Bits and Atoms, Massachusetts, quentinbolsee@hotmail.com

We present an open-source web-based tool for parametric circuit board design. We developed a JavaScript-based system for describing circuits and a graphical editor which assists users in writing JavaScript programs through direct manipulation while providing full access to the textual representation of the board design. We aim to enable a future where users can programmatically generate and share boards as easily as programmers share code. We will share how programmatic representations and bidirectional interfaces have enabled us to create a tool used by both thousands of beginner circuit designers and researchers of novel electronics fabrication processes.

CCS CONCEPTS • Human-centered computing → Interactive systems and tools; • Applied computing → Computer-aided manufacturing; • Software and its engineering → Domain specific languages.

Additional Keywords and Phrases: Electronics, Electronic Design Automation, Hardware Description Languages, web-based

ACM Reference Format:

Leo McElroy and Quentin Bolsée. 2023. Parametric Circuit Design Made Accessible Through Programming and Bidirectional Interfaces. In *CHI2023* Workshop [WS2] - Beyond Prototyping Boards: Future Paradigms for Electronics Toolkits CHI '23, April 23–28, 2023, Hamburg, Germany. ACM, New York, NY, USA, 5 pages.

1 INTRODUCTION

Printed Circuit Board (PCB) design allows people to create functional electronic devices by connecting components in a controlled manner. Traditional Electronic Design Automation (EDA) tools typically rely on clear distinctions between schematic and layout phases of design. Users first create logical connections in schematic planning and then create physical connections in the physical layout phase. Both of the phases rely on direct modeling techniques where users manipulate Graphical Users Interfaces (GUI). The issue with this approach is that it can be difficult to reuse designs, and these designs are both non-hierarchical and

[1]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
This work is licensed under a Creative Commons Attribution 4.0 International License.

non-parametric. All of these issues can be addressed by introducing basic algorithmic concepts to circuit design. We did this by developing a JavaScript-based Hardware Description Language (HDL), and a web-based editor for developing board level designs in this language. In order to remain accessible and ergonomic we also developed a set of bidirectional editing features which allow users to manipulate both the textual and the graphical representations of circuit boards, with both representations remaining synchronized with one another.

We envision a future where circuit design is vastly more accessible because it's far easier to share and modify existing designs, to reuse portions of designs, and to transfer programming experience to circuit design. Additionally we aim to make programming itself more accessible by allowing people to "write" programs from graphical interfaces.

Our team consists of an educational technologist and open-source developer who has created accessible programming, design, and digital fabrication tools used by tens of thousands around the world, and a prominent member of the Fab Lab network (a global network of makerspaces) who also works as a researcher on digital fabrication tools at MIT's Center for Bits and Atoms.

2 RELATED WORK

Existing PCB design tools can be split into two main categories: graphical interface-based applications, and hardware description languages (HDL). Graphical interfaces are still the most popular way to work on small to medium scale boards [2]. Their main flaw is the rigidity of the interface, requiring manual steps and a heavy setup process even for simple projects. The file format produced by those tools is also not easily shared and customized by end users, as it requires fully installing the tool and component library before opening the files. Furthermore, the absence of a scripting language makes it challenging to procedurally generate complex hierarchical boards and wiring.

HDLs were initially created for describing low level integrated circuits [1] but have recently been shown to be applicable to PCB design as well [5], albeit mostly focusing on schematic design and rarely on physical board layout. Another flaw of most HDL scripting languages is the lack of advanced features found in fully-fledged programming languages. To counter this, Lin *et al.* [3] proposed to use a Python programming environment for describing PCB components as classes, but this system still involves manual layout and routing in a separate editor once the netlist is generated.

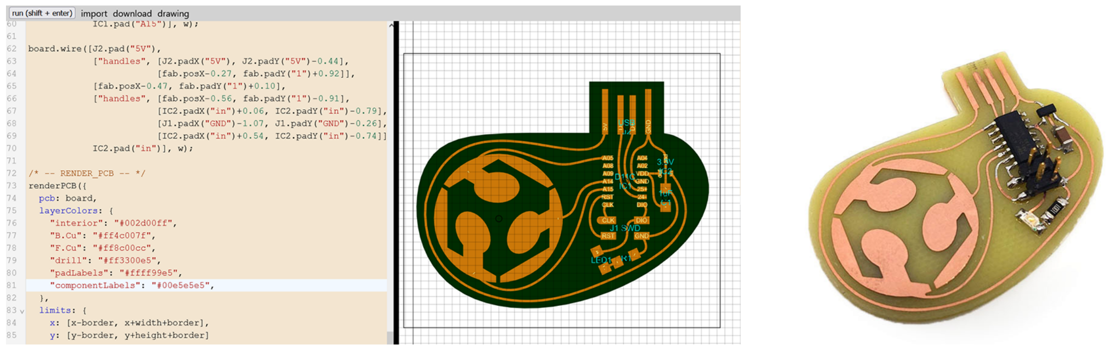
Scripting is available in several modern EDA tools such as EAGLE [6]. These scripting tools are generally designed to automate small sets of sequential procedures whereas our tool describes the complete design programmatically.

3 OUR APPROACH

Our approach [4] revolves around four main design objectives:

- Allow creation of fully parametric PCBs with formally defined component connectivity and geometric placement.
- Leverage familiarity with a fully-fledged programming language to accommodate procedural and programmatic designs.
- Create an easily shareable self-contained representation of PCBs which contains footprint declarations, component placement, and wire geometry.
- Allow users to leverage the flexibility and power of programmatic representations with the ease of direct manipulation interfaces through bidirectional editing. This means rendering the board in an interactive GUI that users can manipulate to modify code.

We have had hundreds of users make use of our circuit design tool for end-to-end board design in university courses, global online courses, and makerspaces. Figure 1 provides an illustration for a board fully designed in our tool, then milled and hand soldered by the designer.



```
              IC1.pad("A15")], w);

board.wire([J2.pad("5V"),
            ["handles", [J2.padX("5V"), J2.padY("5V")-0.44],
                        [fab.posX-0.27, fab.padY("1")+0.92]],
            [fab.posX-0.47, fab.padY("1")+0.10],
            ["handles", [fab.posX-0.56, fab.padY("1")-0.91],
                        [IC2.padX("in")+0.06, IC2.padY("in")-0.79],
                        [J1.padX("GND")-1.07, J1.padY("GND")-0.26],
                        [IC2.padX("in")+0.54, IC2.padY("in")-0.74]]
            IC2.pad("in")], w);

/* -- RENDER_PCB -- */
renderPCB({
  pcb: board,
  layerColors: {
    "interior": "#002d00ff",
    "B.Cu": "#ff4c007f",
    "F.Cu": "#ff8c00cc",
    "drill": "#ff3300e5",
    "padLabels": "#ffff99e5",
    "componentLabels": "#00e5e5e5",
  },
  limits: {
    x: [x-border, x+width+border],
    y: [y-border, y+height+border]
```

Figure 1: Text editor (left), graphical interface (middle) and homemade board (right).

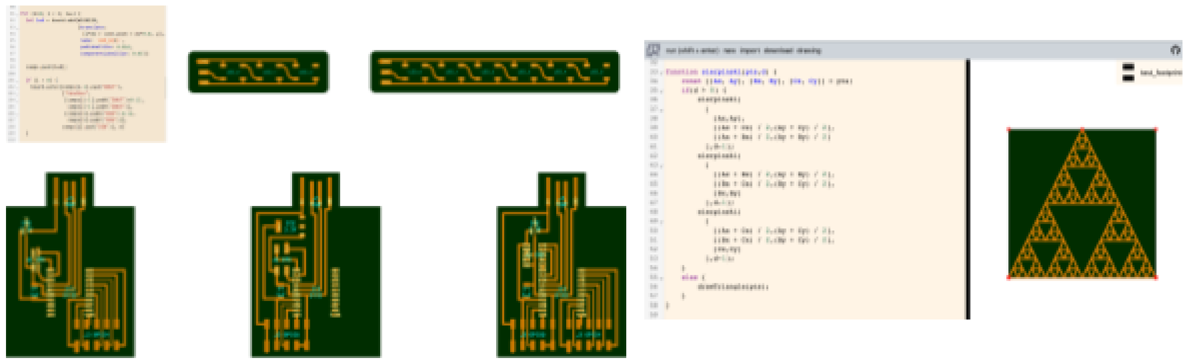


Figure 2: Usage of a loop to generate a flexible LED strip (left-top), conditionals to make boards which can accommodate substituting components (left-bottom), and a function to generate a recursive wire pattern (right).

In our system, basic programming constructs (and all of the associated benefits) are first class tools in circuit design. Concepts like loops, conditionals, and functions allow people to produce parametric and customizable designs which would otherwise be infeasible or impossible to generate in existing tools. Figure 2 provides examples of boards relying on those features.

Additionally, because our tool is web-based and each design file is self-contained with footprint definitions and layouts it is easy to share designs as links. This ease of sharing is playfully demonstrated in Figure 3's self-documenting board.

One important feature of our tool is the real-time rendering of the board in the graphical interface, seen on the right in Figures 1 and 2. This visualization is not just a way for users to immediately see changes upon editing code, but also includes interactive graphical tools such as handles for moving components, a panel for picking and adding new components, and the ability to create new wires using mouse clicks only. Changes made in the graphical interface affect the code, and vice versa. This makes our tool fully bidirectional between the GUI and the code pane.

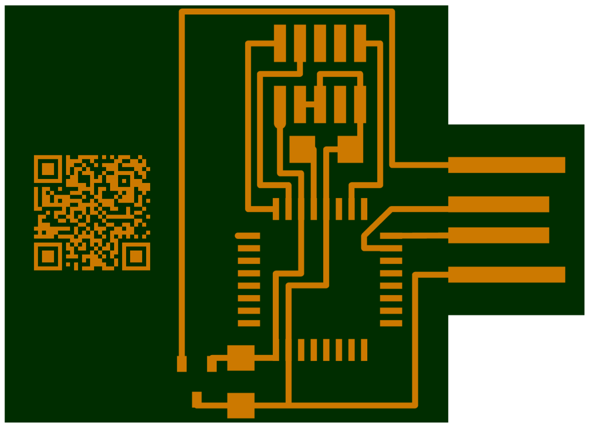


Figure 3: Embedding a link to the complete board design as a QR code into the copper layer of a PCB.

4 FUTURE WORK

Our future work aims to better integrate our circuit design tool with existing PCB design toolchains. A basic step in this direction is supporting high quality exports in both Gerber format (which currently has rudimentary support) and as KiCad Modules.

A significant concept missing from our current board design paradigm is netlists, or specifications of how boards should be logically connected. There is no limitation to adding support for netlists. These netlists would provide necessary information for developing more advanced features such as design rule checking and autorouting.

We also intend to improve the number and quality of graphical manipulation handles. Currently we support dragging points, drawing wires, drag-and-drop components, color/opacity pickers, and various types of control inputs (i.e. sliders). Future improvements will entail more sophisticated curve manipulation tools (such as bezier handle control common in vector graphics editing software) as well as constraint inference.

The core of our bidirectional manipulation system is an incremental parser which automatically recognizes changes to program structure and passes associated information on the concrete syntax tree to the program runtime. This allows us to associate dynamic values generated at execution with information on program structure and to generate manipulation handles which can modify the source program at interactive rates. We plan to articulate and share how others can leverage these techniques to create powerful programmatic systems with friendly accessible interfaces.

REFERENCES

[1] Flake, Peter, et al. "Verilog HDL and its ancestors and descendants." *Proc. ACM Program. Lang.* 4.HOPL (2020): 87-1.

[2] Kanagachidambaresan, G. R. "Introduction to KiCad Design for Breakout and Circuit Designs." *Role of Single Board Computers (SBCs) in rapid IoT Prototyping*. Cham: Springer International Publishing, 2021. 165-175.

[3] Lin, Richard, et al. "Weaving Schematics and Code: Interactive Visual Editing for Hardware Description Languages." *The 34th Annual ACM Symposium on User Interface Software and Technology*. 2021.

[4] McElroy, Leo, et al. "SVG-PCB: a web-based bidirectional electronics board editor." *Proceedings of the 7th Annual ACM Symposium on Computational Fabrication*. 2022.

[5] Nelson, Brent, Brad Riching, and Richard Black. *Using a Custom-Built HDL for Printed Circuit Board Design Capture*. No. SAND2012-1360C. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2012.

[6] Wang, S., Merrill, D., Taylor, C., & Swanson, S. (2016). Echidna: Programmable Schematics to Simplify PCB Design. *UC San Diego: Department of Computer Science & Engineering*. Retrieved from https://escholarship.org/uc/item/8wn4m97p