SVG-PCB: a web-based bidirectional electronics board editor

Leo McElroy
Independent researcher
USA
leomcelroy@gmail.com

Quentin Bolsée
Vrije Universiteit Brussel
Belgium
quentinbolsee@hotmail.com

Nadya Peek
University of Washington
USA
nadya@uw.edu

Neil Gershenfeld
Massachusetts Institute of Technology
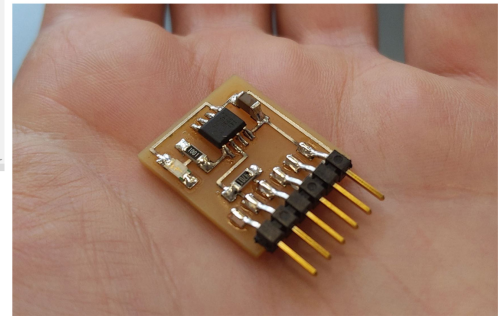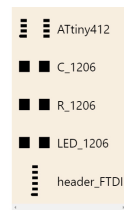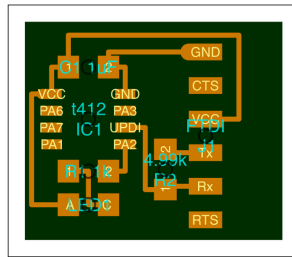USA
neil.gershenfeld@cba.mit.edu

Figure 1: Web-based editor (left) and resulting circuit board (right).

ABSTRACT

We present an open-source, web-based, client-side editor for parametric printed circuit board (PCB) design which supports bidirectional editing between our JavaScript hardware description language and a direct manipulation graphical interface. We developed a JSON format for describing component pads as SVG path data strings, referential component placements, and wire descriptions with curves, arbitrary degree Bezier splines, fillets, and chamfers. Boards can be exported in their JavaScript representation, as SVGs, or in Gerber format. The web-editor also supports interactive elements which update PCB designs in real-time such as number sliders, component translation handles, and drag-and-drop component libraries. Our tool was successfully used for developing and sharing basic boards in a distributed global class on digital fabrication and by researchers to produce procedurally generated designs. SVG-PCB offers the power and flexibility of a general purpose programming language for designing boards with the ease of use of a graphical user interface.

CCS CONCEPTS

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

KEYWORDS

Electronics, EDA, HDL, web-based

1 INTRODUCTION

Printed Circuit Board (PCB) design allows people to create non-trivial electronic devices. Designing PCBs is traditionally done in Electronic Design Automation (EDA) software, popular examples of which include Altium, Cadence, and free and open-source alternatives [Save et al. 2013][KiCad 2017]. Designing and making PCBs has become increasingly accessible; PCB manufacturing techniques using entry-level digital fabrication tools lower the barrier to entry and enable new form factors and applications [Mellis et al. 2013; Steimle 2015]. Further broadening access to PCB design and fabrication tools enables the creation of a long tail of complex electronics prototypes and eventually devices [Khurana and Hodges 2020].

However, there are limitations to current approaches to PCB design [Lin et al. 2019]. Currently, PCB design software typically distinguishes between schematic and layout phases. In the schematic phase, users specify the components (such as passives or specific microcontrollers) and the electrical topology of their circuit. In the layout phase, users define the component packages and their placement, and the specific interconnections through traces and vias, which reifies the schematic. Both of these phases rely extensively on GUIs and manipulating components with a mouse. Making modifications to the schematic after layout has already been done can produce errors which need to be manually corrected. There is a lack of bidirectionality between the phases, which results in friction in the process.

We argue that parametric PCB design tools could address some of these issues. We specifically argue that using a programming language for PCB design will allow users to parametrically define and automate board design. Furthermore, programming PCB design enables the use of programming language techniques for verifying program correctness. This is an approach taken by Hardware Description Languages (HDLs) [Lin et al. 2021; Mashtizadeh 2007; Shahdad 1986], but these thusfar have had a high threshold to adoption. Rather than create a new domain specific language we use an existing general purpose language, JavaScript, along with a small library for describing PCBs which serves as the HDL. We present SVG-PCB, both a JavaScript HDL and a browser-based editor for designing small-scale PCBs. The language provides users the capability to programmatically and parametrically describe relationships and placements of board features. The editor supports textual and graphical representations which are linked through bidirectional mechanisms. We specifically target the design of small-scale boards for interactive devices, an example of which is shown in Figure 1, focusing on hardware's long tail [Hodges and Chen 2019].

With SVG-PCB, we demonstrate an accessible approach to creating PCBs which can be easily shared and modified. Our board descriptions are flexible enough to support structured algorithmic representations but retain the user-friendliness of direct-manipulation interfaces, thanks to bidirectional editing mechanisms.

The remainder of this paper is structured as follows: section 2 presents related work and contrasts our approach, section 3 gives an overview of the features and design choices of our tools, then their implementation in JavaScript is discussed in section 4. We provide an evaluation of our tool through demonstration applications in section 5, followed by future work in section 6 and finally a conclusion in section 7.

2 RELATED WORK

In this section, we provide an overview of related work and contextualize our contribution in this space. We focus on two main areas of related work: systems research on PCB design tools and hardware description languages for PCB design. For our system, we build upon prior work in geometry processing and user interface development; this prior work is described in Section 4 System Implementation.

2.1 Toolkits Supporting Electronics Design and Fabrication

Creating tools for PCB design, fabrication, and debugging is an ongoing research topic. Prior research includes methods for making circuit design more iterative and accessible [Lo and Paulos 2014; Qi and Buechley 2014; Qi et al. 2015], software tools for designing unconventionally shaped circuits [He et al. 2022; Hong et al. 2021; Umetani and Schmidt 2017; Zhu et al. 2020], integrating circuits into existing objects [Ramakers et al. 2016; Tseng and Kawahara 2021], and hardware tools for cross-referencing PCB components to schematics and datasheets [Goyal et al. 2013; Strasnick et al. 2019]. This prior work demonstrates extensive interest in creating tools that support electronics design and fabrication.

This recent interest in electronics design tools can be partially attributed to the Maker Movement, which has significantly broadened interest in interactive technology development [Kuznetsov and Paulos 2010; Lindtner et al. 2014; Tanenbaum et al. 2013]. As new groups of people take up electronics design and fabrication, researchers are exploring how their interests may inform new systems and methods, for example creating biodegradable electronics to reduce environmental impact [Arroyos et al. 2022; Song et al. 2022] and friendly electronics design tools [Knörig et al. 2009; Save et al. 2013; Vidal-Silva et al. 2019; Willis 2015].

While circuit board design tools targeted at the Maker Movement have seen an increase recently, we note that none of these are focused on parametric design, or leveraging the benefits of integrating programmatic representations within the tool. This results in tools that are easy to learn initially, but limited in their capabilities in the long run. It also results in designs which are difficult to share and modify among users.

2.2 Hardware Description Languages (HDLs) for PCB Design

Historically, HDLs were first proposed for describing logic-level functionality in complex Integrated Circuits (IC) such as FPGA and ASIC [Flake et al. 2020]. Simulation tools interface naturally with HDL descriptions, and the board layout can be automatically generated. Nelson *et al.* [Nelson et al. 2012] proposed a new form of HDL that focuses on PCB design rather than IC. The main advantage is an object-based description of the schematic, rather than manual user inputs, while maintaining full flexibility when generating the PCB layout.

Although most HDL is described in its own language, Mashtizadeh [Mashtizadeh 2007] showed how a modern programming language like Python is an efficient environment for describing hardware at a high level. More recently, Lin *et al.* [Lin et al. 2021] have demonstrated bidirectional editing between Python code and a graphical representation. However, these works do not consider the physical layout of the generated board. The user has therefore no control over the practical aspects of the shape, impacting its ease of fabrication in the context of the Maker community.

Finally, while there are examples of web-based HDLs [Dasygenis 2014], there is to our knowledge no tool that takes advantage of the browser's JavaScript interpreter and is fully client-side. To summarize, our primary improvement over existing HDLs is representing
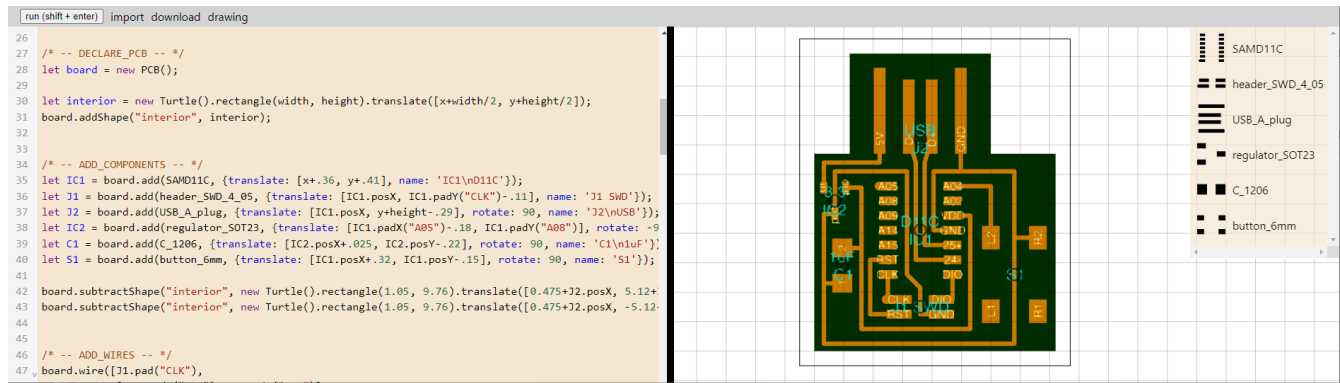
Figure 2: View of SVG-PCB, showing the text editor (left) and the graphical interface (right).

both board connectivity and layout, while leveraging an existing general-purpose language.

3 USING SVG-PCB

SVG-PCB was created to help beginner board designers produce simple designs and re-mix existing board designs quickly and easily. To develop a board users can start by visiting the SVG-PCB website [1]. Because it is entirely client-side users need not download or configure the software to run locally. Users can define the footprints of their board design using plain serializable object representations, or by importing those representations from existing libraries or external URLs. Users then instantiate a board object which can be used to place components on the board and add wires with arbitrary curves. Users can also create arbitrary geometry to define irregular board shapes. Finally users can render the board by mapping board layers to colors and exporting the board as an SVG or Gerber file. The board can be easily shared through its JavaScript description.

To illustrate the usage of SVG-PCB throughout section 3 we will consider the hypothetical user, Diana. Diana is a small business owner who makes custom LED lamps. Diana wishes to produce a custom PCB that fits precisely into a new enclosure her company designs. When first opening SVG-PCB she is presented with the interface shown in Figure 2.

3.1 Footprint Description

Our illustrative user, Diana, copies the majority of a previous PCB description for her LED lamp. But she wants to modify the LEDs that are used and the way they are positioned on the PCB. Starting from the new LED's datasheet, Diana defines a footprint for the new LED component.

A footprint specifies the landing pattern a component will later be soldered onto. Our footprint definition follows a straightforward philosophy:

- Each pad must have a unique name
- Their shape is a general SVG path
- The pad can affect one or more layers

Footprints of this type can be represented/serialized as purely generic objects, making it suitable for interoperability with other EDA tools. The data model is as follows:

```
1  {
2      padName: {
3          pos: [x, y],
4          shape: "as SVG path data string",
5          layers: ["F.Cu"]
6      }
7  }
```

Here, `F.Cu` refers to the front copper layer. This is probably the default layer one would use most of the time. The `pos` value positions the pad relative to the center of the component. The shape is then centered around `pos`. A pad can be on multiple layers.

3.2 Adding Components

Diana then wants to add the component she just defined to the board she is designing.

Components can be added to the board by specifying a footprint object, a position and angle. The result is a component object which can be used to access pad positions. Consequently users can directly reference a pad of a given component when placing another component or tracing wires, making parametric boards with little added complexity for the user. Figure 3 shows an example component placement.

3.3 Wire Description

Diana then defines the interconnections between the components by specifying wires. Wires can be described with a list of points, chamfers, fillets, and Bezier curves of arbitrary degree, as illustrated in Figure 4.

The different types of points in a wire path are interpreted as follows:

- **Default:** a regular point, simply containing a `x` and `y` coordinate.
- **Chamfer:** specified by a `chamfer` command followed by the radius of the chamfer and the location of the corner being chamfered.
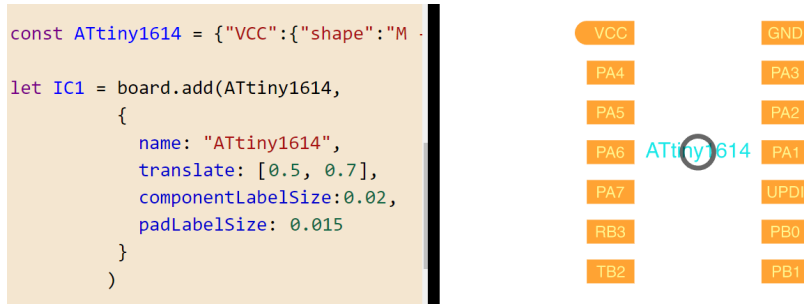- **Fillet:** same logic as chamfers but have a rounded corner.

Figure 3: Component objects can be instantiated from a footprint.
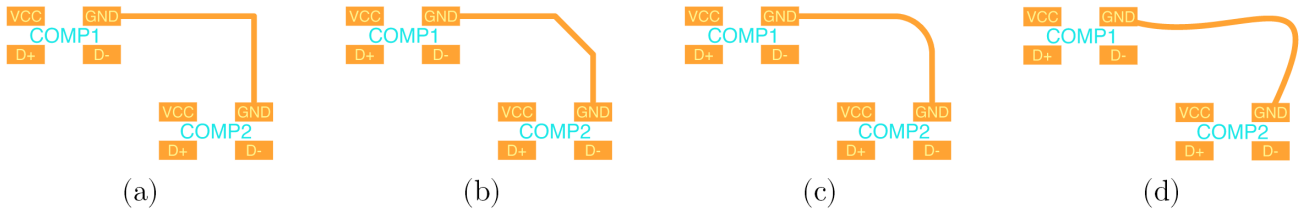
Figure 4: Different type of wire points. Left to right: regular (a), chamfer (b), fillet (c) and Bezier splines (d).

- **Bezier spline:** specified by a `bezier` command followed by points for the location of each desired control handle.

An example wire path can be seen below:

```
1  board.wire([
2    [x1, y1],
3    ["chamfer", r1, [x2, y2]],
4    ["fillet",  r2, [x3, y3]],
5    [x4, y4],
6    ["bezier", [x5, y5], [x6, y6]],
7    [x7, y7]
8  ], thickness)
```

Chamfers and fillets can not be the first or last elements of a wire because they must be applied at corner points and need a direction when computing their geometry. Beziers must be preceded and proceded by plain (non-chamfered and non-filleted) points, in order to avoid over-constrained geometry. Wire thickness is passed as an additional parameter when creating a wire.

3.4 Arbitrary Geometry

Diana's lamp design requires specific irregular geometry. Arbitrary geometry can be defined from SVG path data which allows Diana to shape her board to her curvy lamp. This geometry can be added to any layer of the board.

3.5 Rendering a Board

Finally, Diana renders the PCB and exports it for fabrication. Boards can be rendered by specifying the target PCB, the target layers with certain colors, the x and y limits which will be generated upon export, and the real life scale. An example render is seen below:

```
1  renderPCB({
2    pcb: board,
3    layerColors: {
```

```
4      // "layer": "hex",
5      "interior": "#002d00ff",
6      "F.Cu": "#ff8c00cc",
7      "padLabels": "#ffff99e5",
8      "componentLabels": "#00e5e5e5",
9    },
10   limits: {
11     x: [minX, maxX],
12     y: [minY, maxY]
13   },
14   mm_per_unit: 25.4
15  });
```

4 SYSTEM IMPLEMENTATION

SVG-PCB embeds a JavaScript code editor which performs incremental parsing, with a geometry engine that generates board visualizations and export formats. The entire system is designed to support real-time interaction performance rates (repeated operations perform in <10ms) in order to enable the bidirectional manipulation features described below. The PCB data structure associates footprints, shapes, and wire descriptions with named layers of the board. These layers can be retrieved quickly as unflattened paths or flattened into the outlines of real geometric features.

The design goals of our system are:

- 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.

Figure 5: Rendering results with multiples layers (a), and with only traces in black (b).

- Allow users to leverage the flexibility and power of programmatic representations with the ease of direct manipulation interfaces through bidirectional editing.

In the remainder of section 4 we will explain the bidirectional editing feature set, algorithms and techniques used to achieve interactive performance rates, how constraints can be described, and the output formats of the tool.

4.1 Bidirectional Editing

One of the primary advantages of our tool is providing the power and flexibility of formal representations through textual interfaces with the ease of manipulating a graphical interface. The challenge in providing the user with both representations is synchronizing manipulations of the design that occur in either. Bidirectional editing allows users to manipulate either representation, textual or graphical, while keeping both in sync. Available graphical manipulations are determined by the text source file and graphical user interactions map to intuitive alterations of the text. There are three mechanisms that enable intuitive user interaction:

- Any numerical constant in the code editor can be dragged with the mouse cursor. For instance, a constant defining the position of a component or wire can be modified in this way, showing real-time updates in the render.
- A number of visual handles in the graphical interface let users move positions. This affects the code by adding or modifying an additive constant to an item's position. Components and wires can be manipulated in this way.
- New components can be added from the graphical interface by selecting a footprint from a component toolbox and dragging it into the design. The adequate lines of code are automatically added for instantiating the component at the target location.

4.2 Real-Time Interaction

Real-time interaction performance rates are necessary to support bidirectional manipulation. The computations associated with these manipulations have to be highly efficient to keep interactions responsive. When the syntax tree is altered by direct manipulations the entire board program is evaluated. We utilized a variety of techniques in order to prevent this evaluation from lagging user

interactions. These techniques include selectively generating complete board geometries based upon user actions, relying on the code editor's incremental parsing of the concrete syntax tree for analysis of the source file structure, and selectively updating the application user interface upon state changes with tagged template literal rendering. Complete board layer geometries are calculated only during export.

For visualization, board layers are rendered as SVGs and non-expanded wires are rendered as paths with the appropriate stroke thickness. Upon export to real geometry we apply Angus Johnson's expanded implementation of Bala R. Vatti's clipping algorithm [Vatti 1992] to expand wire geometries and generate collapsed outlines of all layer geometry.

Abstract syntax tree manipulation is accelerated by leveraging CodeMirror's [2] incremental parsing system, Lezer. Incremental parsing is a technique to avoid re-parsing entire concrete syntax trees for source files by parsing a source file once and efficiently updating it upon edits [Wagner and Graham 1998]. These concrete syntax trees represent the source texts exactly, whereas an abstract syntax tree only represents the content-related details. Incremental parsing is notably used in the Tree Sitter project [3], and commonly applied to develop syntax highlighting systems. When editing portions of the syntax tree through direct manipulation, such as when dragging a component, the corresponding textual representation is identified by walking the entirety of the programs syntax tree. The sub-tree representing the translation is then modified and converted back to textual form which is then re-inserted into the source file. The modified portion is then re-parsed and the entirety of the syntax tree reanalyzed upon further action. Analyzing the entire syntax tree to identify relevant snippets which must be rewritten allows the system to be robust, while preserving its performance by only re-parsing modified snippets.

The application is efficiently rendered by only preforming costly Document Object Model (DOM) updates to portions of the view modified by state changes. This is enabled by diffing changes in the view through tagged template literals before updating the DOM itself, we utilize Andrea Giammarchi's uhtml library for this [4].

---

[2] https://github.com/codemirror
[3] https://github.com/tree-sitter/tree-sitter
[4] https://github.com/WebReflection/uhtml
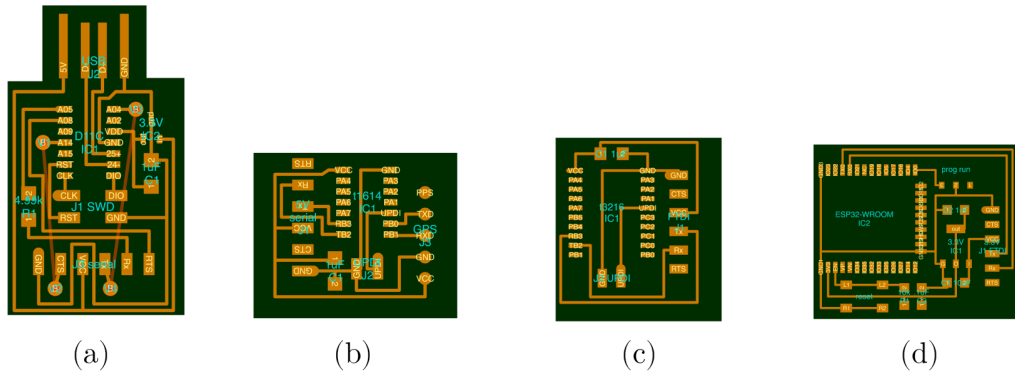
(a)   (b)   (c)   (d)

Figure 6: Example boards used for teaching digital fabrication, including ATSAMD11C14 USB-to-serial bridge (a), ATtiny1614 with GPS module (b), ATtiny3216 serial echo (c) and ESP32 development board (d).
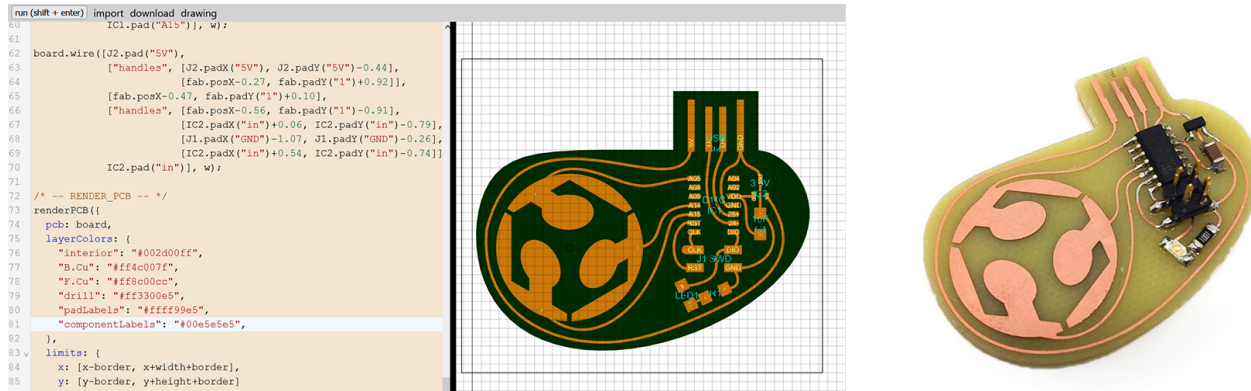

Figure 7: Left: example board design using SVG insertion and Bezier splines wires. Right: resulting board milled on a desktop CNC milling machine.

4.3 Referential Constraints

Referential constraints refer to the ability to reference pad and component positions as variables. This allows users to build up dependent board designs that maintain internal structure upon changes. Constraints can be encoded in the programmatic description of the board. Component pads can be referenced when placing other components or wires. Consequently users can place components and board features with relative dependencies to other board features. An example of such a placement would be positioning a component 10mm to the right of a ground pin of another component. If the ground pin were to be moved the component to the right would remain 10mm to the pin's right.

4.4 Output

Exporting files for PCB fabrication is a critical component of EDA. SVG-PCB supports export in multiple formats, accommodating both industrial boardhouse expectations and digital fabrication techniques such as milling. The main output format is a multi-layered SVG, as this is the native format used during editing. The user selects the board layers to be rendered as well as their color

and transparency. We also support Gerber file export for common layer types.

The output layers and colors are defined in a function call in the textual representation, this leaves full freedom to the user for generating different outputs based on conditional statements.

5 APPLICATIONS

Here, we present a variety of applications that emerged over the course of a year refining and sharing our tool with an active community of makers, programmers, and researchers. Some of these use cases demonstrate the basic features that we planned for, while others display the wanted side effects of including a fully-fledged programming language within the editor.

5.1 Basic Boards for Education

SVG-PCB was successfully applied in an educational environment, namely as part of a global class on digital fabrication [5]. Over 40 simple boards were distributed through URLs, each demonstrating a basic electronics principle or device. 6 depicts some of these designs. These boards are primarily surface mount and designed to
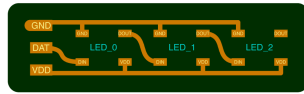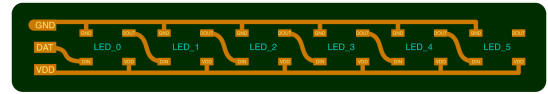
---

[5]https://fabacademy.org/

(a)   (b)   (c)

Figure 8: Code generating a procedural array of components (a), with arbitrary length and automatic wiring. Examples are shown for $n = 3$ (b) and $n = 6$ (c).
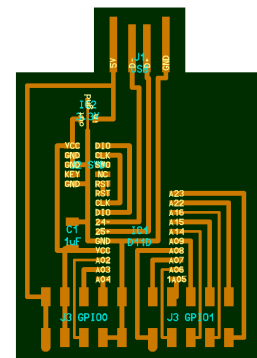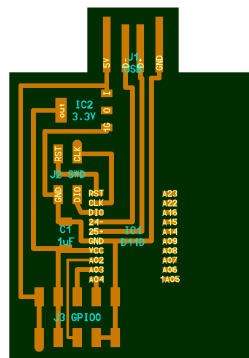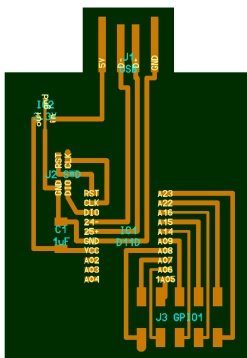

Figure 9: Reconfigurable board, where the user can select connectors and some components (here, the voltage regulator) from a list.

be fabricated by a desktop milling machine and soldered by hand. To support this workflow, SVG-PCB easily integrates with web-based software for developing re-configurable digital fabrication pipelines, mods [Peek and Gershenfeld 2018]. The result of milling one of these boards can be seen in Figure 1.

To summarize, this example shows the ease of sharing designs while enabling quick customization by the end user, without installing any additional software or library.

5.2 Bezier Spline Wires

To demonstrate direct insertion of SVG content, as well as full Bezier capabilities when creating wires, we designed a minimal board with USB capabilities based on a Microchip ATSAMD11C14, shown in Figure 7. The Fab Lab logo in SVG format was converted to a custom footprint by defining each path as a pad. For the wires, the position of each Bezier spline control point is defined relative to a nearby element on the board. This results in a fully parametric set of curvy traces that can distort in a consistent way when dragging one of the components.

For further showcasing capabilities of the ATSAMD11C14, the Fab Lab logo acts as a capacitive touch sensor with 3 independent switches. This microcontroller supports a full HID keyboard USB class, allowing the pads to trigger keyboard presses.

5.3 Array

Having a full programming language at the user's disposal lead to interesting procedural electronics boards, designed in a research environment. In one specific case, a flexible PCB strip containing an array of WS2812b LEDs was generated from a simple `for` loop directive. Chained wires were easily added by referencing pads of the previously added component object. Figure 8 shows a code snippet and the resulting boards for different numbers of loop iterations. While generating an array is a basic functionality of HDL tools, having tight control on the geometry and wiring is unique to our approach.

5.4 Conditional Design

Another emerging use of the code editor is integrating conditional statements as part of the design. In this example, a generic USB-capable board was made based on a Microchip ATSAMD11D14, but the type of connectors and voltage regulator can be selected from a list. This accommodates for changes in the inventory available when producing the board, ultimately increasing the potential user base for a given design with minimal user input. A few examples are shown in Figure 9.

6 DISCUSSION AND FUTURE WORK

We argue that a JavaScript based hardware description language for PCB design will allow users to more easily share and remix boards by letting them create parametric and automated board designs. Programming PCB descriptions enables the use of algorithmic techniques for creating designs which are iterative, conditional, and hierarchical. Existing HDLs focus on describing board connectivity but not geometry and give up convenient graphical editing features. We overcome this by including board layout in our programmatic representation, and by creating an web-based editor which allows users to visualize and manipulate board designs graphically. The editor links the textual and graphical representations through bidirectional editing features. These features allow users to create and modify designs without having to manually manipulate syntax.

One limitation of our current system is a lack of design rule checking to verify circuit properties. Some checks, such as determining shorts, would require knowing the topological connectivity of boards which could be specified through netlists. These netlists were not initially included because our system does not rely on the two part design process of traditional EDA tools where a user first creates a schematic (which defines a netlist) and then specifies a layout. Optional netlists in our tool could also be used for autorouting. Design rule checking also verifies physical properties of circuits which relate to board layout, such as separation distances of wires. These sorts of checks are not precluded by our system and can be added in the future.

We plan to implement a number of improvements to the tool. The bidirectional editing currently allows users to drag wires and components, but creating new wires would require a finer-grain handling of every component's pad. Another aspect to improve is the constraint system, which currently requires users to manually type references to other components in the code editor. These kinds of constraints can be inferred from the graphical interface, similar to graphical constraint tools in a modern CAD software.

7 CONCLUSION

In this work, we have presented an approach to describing PCBs with a JavaScript HDL and an editor which allows users to dynamically create and manipulate these textual representations with a synchronized graphical interface. Our tool is a hybrid between HDLs and visual editor EDAs, allowing bidirectional editing from either context window. Our approach reduces barriers to adoption of the HDL by leveraging the capabilities of a modern programming language. This allows users to define board connectivity and geometry parametrically, whereas most HDLs only allow users to specify connectivity and most visual editors are non-parametric. Single file representations also enable users to easily distribute modifiable boards. We presented the use of our board in distributed educational and research environments. Our hope is that further development and adoption of tools such as SVG-PCB will allow more people to discover PCB design through tools which are easy to get started with, yet flexible and powerful, and with designs which are easy to modify, remix, and share.

REFERENCES

Vicente Arroyos, Maria L K Viitaniemi, Nicholas Keehn, Vaidehi Oruganti, Winston Saunders, Karin Strauss, Vikram Iyer, and Bichlien H Nguyen. 2022. A Tale of Two Mice: Sustainable Electronics Design and Prototyping. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, Article 263, 10 pages. https://doi.org/10.1145/3491101.3519823

Minas Dasygenis. 2014. A web EDA tool for the automatic generation of synthesizable VHDL architectures for a rapid design space exploration. In *2014 9th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 1–2.

Peter Flake, Phil Moorby, Steve Golson, Arturo Salz, and Simon J Davidmann. 2020. Verilog HDL and its ancestors and descendants. *Proc. ACM Program. Lang.* 4, HOPL (2020), 87–1.

Pragun Goyal, Harshit Agrawal, Joseph A. Paradiso, and Pattie Maes. 2013. BoardLab: PCB as an Interface to EDA Software. In *Proceedings of the Adjunct Publication of the 26th Annual ACM Symposium on User Interface Software and Technology* (St. Andrews, Scotland, United Kingdom) *(UIST '13 Adjunct)*. Association for Computing Machinery, New York, NY, USA, 19–20. https://doi.org/10.1145/2508468.2514936

Liang He, Jarrid A. Wittkopf, Ji Won Jun, Kris Erickson, and Rafael Tico Ballagas. 2022. ModElec: A Design Tool for Prototyping Physical Computing Devices Using Conductive 3D Printing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 4, Article 159 (dec 2022), 20 pages. https://doi.org/10.1145/3495000

Steve Hodges and Nicholas Chen. 2019. Long tail hardware: Turning device concepts into viable low volume products. *IEEE Pervasive Computing* 18, 4 (2019), 51–59.

Freddie Hong, Connor Myant, and David E Boyle. 2021. Thermoformed Circuit Boards: Fabrication of Highly Conductive Freeform 3D Printed Circuit Boards with Heat Bending. 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 669, 10 pages. https://doi.org/10.1145/3411764.3445469

Rushil Khurana and Steve Hodges. 2020. *Beyond the Prototype: Understanding the Challenge of Scaling Hardware Device Production.* Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3313831.3376761

EDA KiCad. 2017. A cross platform and open source electronics design automation suite. *Disponible en: http://kicad-pcb. org* (2017).

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)*. Association for Computing Machinery, New York, NY, USA, 351–358. https://doi.org/10.1145/1517664.1517735

Stacey Kuznetsov and Eric Paulos. 2010. Rise of the Expert Amateur: DIY Projects, Communities, and Cultures. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries* (Reykjavik, Iceland) *(NordiCHI '10)*. Association for Computing Machinery, New York, NY, USA, 295–304. https://doi.org/10.1145/1868914.1868950

Richard Lin, Rohit Ramesh, Antonio Iannopollo, Alberto Sangiovanni Vincentelli, Prabal Dutta, Elad Alon, and Björn Hartmann. 2019. Beyond Schematic Capture: Meaningful Abstractions for Better Electronics Design Tools. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300513

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*. 1039–1049.

Silvia Lindtner, Garnet D. Hertz, and Paul Dourish. 2014. Emerging Sites of HCI Innovation: Hackerspaces, Hardware Startups & Incubators. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 439–448. https://doi.org/10.1145/2556288.2557132

Joanne Lo and Eric Paulos. 2014. ShrinkyCircuits: Sketching, Shrinking, and Formgiving for Electronic Circuits. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (Honolulu, Hawaii, USA) *(UIST '14)*. Association for Computing Machinery, New York, NY, USA, 291–299. https://doi.org/10.1145/2642918.2647421

Ali Mashtizadeh. 2007. *PHDL: A Python hardware design framework.* Ph. D. Dissertation. Massachusetts Institute of Technology.

David A. Mellis, Sam Jacoby, Leah Buechley, Hannah Perner-Wilson, and Jie Qi. 2013. Microcontrollers as Material: Crafting Circuits with Paper, Conductive Ink, Electronic Components, and an "Untoolkit". In *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction* (Barcelona, Spain) *(TEI '13)*. Association for Computing Machinery, New York, NY, USA, 83–90. https://doi.org/10.1145/2460625.2460638

Brent Nelson, Brad Riching, and Richard Black. 2012. *Using a Custom-Built HDL for Printed Circuit Board Design Capture.* Technical Report. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

Nadya Peek and Neil Gershenfeld. 2018. Mods: Browser-based rapid prototyping workflow composition. *ACADIA 2018* (2018).

Jie Qi and Leah Buechley. 2014. Sketching in Circuits: Designing and Building Electronics on Paper. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 1713–1722. https://doi.org/10.1145/2556288.2557391

Jie Qi, Andrew "bunnie" Huang, and Joseph Paradiso. 2015. Crafting Technology with Circuit Stickers. In *Proceedings of the 14th International Conference on Interaction Design and Children* (Boston, Massachusetts) *(IDC '15)*. Association for Computing Machinery, New York, NY, USA, 438–441. https://doi.org/10.1145/2771839.2771873

Raf Ramakers, Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2016. Retro-Fab: A Design Tool for Retrofitting Physical Interfaces Using Actuators, Sensors and 3D Printing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 409–419. https://doi.org/10.1145/2858036.2858485

Yogesh Dilip Save, Rajalekshmi Rakhi, ND Shambhulingayya, Ambikeshwar Srivastava, Manas Ranjan Das, Saket Choudhary, and Kannan M Moudgalya. 2013. Oscad: An open source EDA tool for circuit design, simulation, analysis and PCB design. In *2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS)*. IEEE, 851–854.

M. Shahdad. 1986. An Overview of VHDL Language and Technology. In *23rd ACM/IEEE Design Automation Conference*. 320–326. https://doi.org/10.1109/DAC.1986.1586107

Katherine W Song, Aditi Maheshwari, Eric M Gallo, Andreea Danielescu, and Eric Paulos. 2022. Towards Decomposable Interactive Systems: Design of a Backyard-Degradable Wireless Heating Interface. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 100, 12 pages. https://doi.org/10.1145/3491102.3502007

Jürgen Steimle. 2015. Printed Electronics for Human-Computer Interaction. *Interactions* 22, 3 (apr 2015), 72–75. https://doi.org/10.1145/2754304

Evan Strasnick, Sean Follmer, and Maneesh Agrawala. 2019. Pinpoint: A PCB Debugging Pipeline Using Interruptible Routing and Instrumentation. In *Proceedings of the*

2019 CHI Conference on Human Factors in Computing Systems (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3290605.3300278

Theresa Jean Tanenbaum, Amanda M. Williams, Audrey Desjardins, and Karen Tanenbaum. 2013. Democratizing Technology: Pleasure, Utility and Expressiveness in DIY and Maker Practice. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. Association for Computing Machinery, New York, NY, USA, 2603–2612. https://doi.org/10.1145/2470654.2481360

Tiffany Tseng and Yoshihiro Kawahara. 2021. Circuit Assemblies: Electronic Modules for Interactive 3D-Prints. In *Designing Interactive Systems Conference 2021* (Virtual Event, USA) *(DIS '21)*. Association for Computing Machinery, New York, NY, USA, 1115–1128. https://doi.org/10.1145/3461778.3462024

Nobuyuki Umetani and Ryan Schmidt. 2017. SurfCuit: Surface-Mounted Circuits on 3D Prints. *IEEE Computer Graphics and Applications* 37, 3 (2017), 52–60. https://doi.org/10.1109/MCG.2017.40

Bala R Vatti. 1992. A generic solution to polygon. *Commun. ACM* 35, 7 (1992), 56–63.

Cristian Vidal-Silva, Jorge Serrano-Malebran, and Felipe Pereira. 2019. Scratch and Arduino for Effectively Developing Programming and Computing-Electronic Competences in Primary School Children. In *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*. 1–7. https://doi.org/10.1109/SCCC49216.2019.8966401

Tim A Wagner and Susan L Graham. 1998. Efficient and flexible incremental parsing. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 20, 5 (1998), 980–1013.

Karl D. D. Willis. 2015. Project Wire. https://www.karlddwillis.com/project-wire/, accessed 2022.

Junyi Zhu, Yunyi Zhu, Jiaming Cui, Leon Cheng, Jackson Snowden, Mark Chounlakone, Michael Wessely, and Stefanie Mueller. 2020. *MorphSensor: A 3D Electronic Design Tool for Reforming Sensor Modules*. Association for Computing Machinery, New York, NY, USA, 541–553. https://doi.org/10.1145/3379337.3415898