Beyond Schematic Capture

Meaningful Abstractions for Better Electronics Design Tools

Richard Lin, Rohit Ramesh, Antonio Iannopollo,
Alberto Sangiovanni Vincentelli, Prabal Dutta, Elad Alon, Björn Hartmann
University of California, Berkeley
{richard.lin,rkr,antonio,alberto,prabal,elad,bjoern}@berkeley.edu

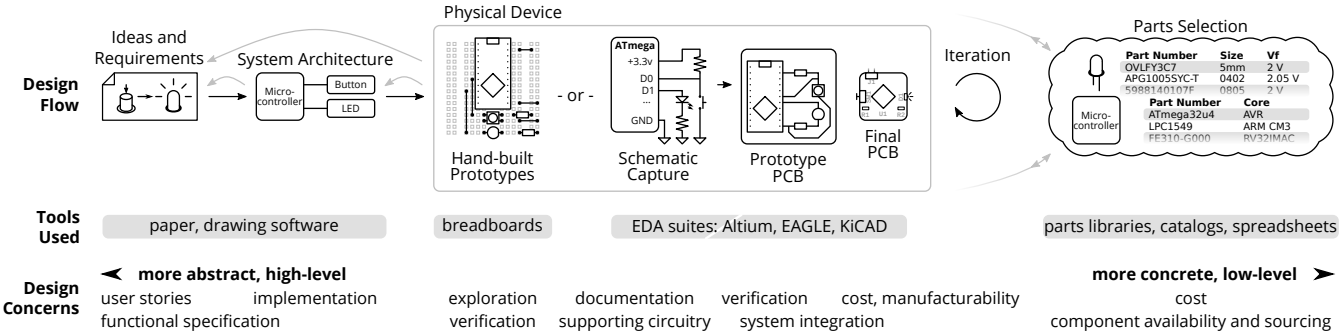

Figure 1: The electronics design flow, as described by our participants. Users start with an idea, refine that into a system architecture, and then iterate physical prototypes. Parts selection happens throughout the process. While certain steps require linear progression, iteration and revision of earlier stages also happen. Overall, EDA tools only support a small part of this process, and moving between steps was a major source of friction.

ABSTRACT

Printed Circuit Board (PCB) design tools are critical in helping users build non-trivial electronics devices. While recent work recognizes deficiencies with current tools and explores novel methods, little has been done to understand modern designers and their needs. To gain better insight into their practices, we interview fifteen electronics designers of a variety of backgrounds. Our open-ended, semi-structured interviews examine both overarching design flows and details of individual steps. One major finding was that most creative engineering work happens during system architecture, yet current tools operate at lower abstraction levels and create significant tedious work for designers. From that insight, we conceptualize abstractions and primitives for higher-level tools and elicit feedback from our participants on clickthrough mockups of design flows through an example project. We close with our observation on opportunities for improving board design tools and discuss generalizability of our findings beyond the electronics domain.

CCS CONCEPTS

• **Human-centered computing → Human computer interaction (HCI)**; • **Applied computing → Computer-aided design**.

KEYWORDS

printed circuit board (PCB) design; electronics design automation (EDA) tools; schematic capture; PCB layout.

ACM Reference Format:
Richard Lin, Rohit Ramesh, Antonio Iannopollo,, Alberto Sangiovanni Vincentelli, Prabal Dutta, Elad Alon, Björn Hartmann. 2019. Beyond Schematic Capture: Meaningful Abstractions for Better Electronics Design Tools. In *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019), May 4–9, 2019, Glasgow, Scotland UK*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3290605.3300513

CHI 2019, May 4–9, 2019, Glasgow, Scotland UK

ACM ISBN 978-1-4503-5970-2/19/05.
https://doi.org/10.1145/3290605.3300513

1 INTRODUCTION

Interactive products are everywhere in modern life — they give us ways to interact with processes and technologies

that are impossible to directly control. A microwave lets you control a powerful source of electromagnetic radiation to warm up your soup. A thermostat can measure the warmth of a room and provide you, not only a way to see that measurement, but also ways to change the temperature to your liking. While interactive devices span many application domains, often at their core are electronics, built on printed circuit boards (PCBs) and designed using electronic design automation (EDA) tools. The ubiquity of PCBs means that improvements to their design processes in general can be transformative.

A variety of board-level EDA tools exist for different user groups: EAGLE [4] is largely geared towards hobbyists; KiCad [19] is open source; while Altium [1], Cadence Allegro [9], and Mentor Xpedition [26] are geared towards professionals working on complex projects. Overall, the still-dominant UI paradigm for these tools was established in the 1980s: users start by designing the circuit in *graphical schematic capture* tools, interactively adding and connecting abstract representations of individual electronic components. They then proceed to *board layout*, where they place physical representations of circuit components and route conductive traces between pins. Most tools can also perform automated analyses such as schematic sanity and layout manufacturability checks, while advanced tools may offer signal integrity and electromagnetic compatibility verification.

However, tools have not kept pace with the shifting community of designers or the new opportunities afforded by more powerful computers and algorithms. Online resources, beginner-friendly platforms like Arduino [3], and falling costs of both parts and fabrication means that the barrier to entry for electronics has never been lower. While breadboarding is both common and accessible for beginners, the resulting devices are fragile and bulky. Moving beyond prototyping usually means using EDA tools to build PCBs. Yet, these tools have stayed fundamentally the same over the years, without providing the design assistance that would serve these new communities of designers.

Altogether, these changes have prompted recent exploration of novel approaches towards PCB EDA tools, which exploit modern techniques and the rapid growth of available computational power [5, 30]. We argue that in order to be successful, such work must be grounded with empirical studies about user needs. Yet, the bulk of the published HCI work on usability in EDA suites dates back to the early 1980s, as systems moved away from text-based schematic entry and towards the currently-dominant graphical schematic capture paradigm [23, 32]. It is time for us to start searching for the next paradigm, and ensuring that it will support designers' needs for years to come.

Research Questions

Our goal is to inform the development of novel EDA tools, and we primarily seek to discover **what approaches to better design tools are fruitful, and why**.

As we believe that effective tools must fit the needs of users, it is crucial that we understand current design practices. This includes **understanding the design flows from idea to physical device**. Such a broad investigation allows us to take a holistic look at the process, including steps that are underserved by current tools.

For each step in the flow, we also delve into **which tools (if any) are used, whether they work well, and where the pain points are.** This deeper look reveals details of designers' thought processes, intermediate goals within their workflows, and their interactions and frustrations with existing tools.

We then use what we learn to envision plausible, alternate tools that can better support these design flows. By examining participant feedback with mockups of our prospective designs, we look more closely at **what designers find valuable, and where future tool builders should focus**.

Contributions

Our overall contributions consist of a formative interview study with 15 participants to assess current practices and problems (in Sections 4 and 5), followed by the design of a tool concept that addresses major issues (in Section 6), and ending with user feedback on a mockup of the tool concept (in Sections 7 and 8). This overall methodology follows the example of prior papers that combine formative studies with concept designs for better tools in other domains [17, 27].

2 RELATED WORK

Foundations

Work exploring human factors in PCB design tools appeared as early as the 1970s. One major issue at the time was the awkwardness of punch card or text based schematic entry.

Matthews [23] criticized tools of the time as "operator-aided computers", and describes an interactive graphical schematic entry system similar to today's mainstream tools. Despite the requirement for more user time on then-expensive computing resources, this system both reduced the overall design time and eliminated an error-prone punch card step.

Shiraishi regarded interactive schematic drawing as time-consuming, and his ICAD/PCB system [32] instead digitizes hand-drawn schematics using pattern recognition techniques. However, it focused on logic circuits and required a standard schematic style.

Another common theme of EDA systems from this era was integration between different steps, commonly schematic

entry, layout, and simulation. Reasons included the time, cost, and potential for errors from manual translation [6, 21, 23].

Systems also explored methods for component placement and board routing. Varying degrees of automatic placement were part of many tools, while other systems provided interactive feedback on manual placements [23, 32]. Autorouting was also a research theme [6, 21–23], typically in conjunction with interactive manual routing.

Rager and Weiner [29] did an in-depth study on dense board layouts, recommending an interactive system where a human guides automated processes, but is provided with powerful assistive tools when lower-level manual intervention is needed.

Recent Work

With a dominant paradigm established, later research literature focused on deepening technical aspects such as better autorouting techniques [14], while new user-facing features were predominantly introduced by a growing industry. In-house studies of tool usability for commercial software generally are not available to the academic community.

The rise of the maker movement has spurred some recent work on examining the use EDA tools by different communities. Mellis [25] observed novices over the course of a six week workshop as they were taught embedded design and built boards. While it is possibly the most comprehensive examination involving modern EDA tools and practices so far, the focus on novices sets a low complexity ceiling and does not address needs of more advanced users.

Beyond board design tools, some recent work focuses on electronics prototyping. Fritzing [20] aims to ease the transition from breadboards to PCBs, CircuitStack [37] adds a printed connectivity layer to breadboards, and Crossed Wires [8] examines issues in breadboard circuit construction.

A key research direction has been to develop augmented breadboards, as many students start building circuits on breadboards. These include measuring and visualizing voltages [12] and current flows [39], and detection of inserted components through active probing [40]. Bifröst [24] further combines code instrumentation and logic analyzer circuit tracing for examining the hardware-software boundary.

There has also been work on instrumentation outside the structure of breadboards, such as using augmented reality to overlay simulation data onto a physical device [11] or linking a probed point to a schematic pin [15].

Such a large body of work hints at the many problems with modern electronics practices, but many of these novel systems only seek to improve a specific part of the process instead of examining design holistically.

Novel Approaches to PCB Design

Recently, there has also been work on radically different paradigms for board design.

One end of the spectrum takes inspiration from hardware description languages like Verilog and hardware construction languages like Chisel [18], both of which improve design by raising the level of abstraction. PHDL [28] is a Verilog-like language that provides for module-level re-use with a limited degree of parameterization. JITPCB [5] takes the concept one step further by embedding hardware construction functionality in a general-purpose programming language, allowing more complex, user-defined circuit generators.

On the other end are highly automated tools, which generate circuits from high-level specifications. EDASolver [13] takes in a tree describing the basic structure of an embedded device with requested peripherals, and produces a circuit meeting those requirements. EDG [30] further advances the concept, inferring both a compute block and interface elements from a list of requested peripherals and external connections. Both systems ensure correctness by reasoning over electronic quantities and limitations including voltages, currents, and absolute maximum ratings.

Trigger-Action-Circuits [2] takes design inputs at an even higher, behavioral level, and allows users to explore tradeoffs between a variety of generated candidate designs. Of these recent approaches, it is the only one with a user study.

For the others, it is unknown if their specification formats and abstractions are suitable for current designers and address pain points. Examples for all these systems also tend to be quite simple, falling within the capabilities of an intermediate hobbyist designer. Applicability towards more complex, professional designs and requirements is uncertain.

Creativity Support Tools

EDA tools are part of the larger class of creativity support tools, which has received attention in the HCI literature with topics ranging from theoretical foundation, observations, and suggestions [33–35]. In particular, Resnick et al. [31] describe a set of twelve design principles and recommendations for these tools. However, these high-level principles must be grounded with domain knowledge and user feedback to formulate concrete, actionable improvements. Our paper seeks to build this bridge to the specific domain of EDA tools.

3 PARTICIPANTS

We conducted an interview study with 15 participants (14 male), of which 10 returned for the follow-up mockup study. While small, this group covers a wide variety of skill levels, design types, and EDA tools used. Critically, both professional and hobby users are included. A summary of participants' backgrounds is shown in Table 1.

Participant	*Age*	*Motivations*	*EDA tool*	*Design discussed*	*Mockup study*
P01	late 20s	school, hobby	EAGLE	LED board	Yes
P02	early 20s	school, hobby	EAGLE	analog feedback-controlled heater	Yes
P03	early 20s	school, job (startup)	EAGLE	Arduino motor controller	No
P04	early 20s	school, research, hobby	EAGLE	music recording system	Yes
P05	late 20s	research, side jobs, hobby	EAGLE	robotics	Yes
P06	early 30s	research, hobby	EAGLE	electrical muscle stimulation	No
P07	early 20s	school, hobby, job (industry engineer)	KiCad	IO controller	Yes
P08	early 30s	job (industry engineer), hobby	KiCad	educational kits	Yes
P09	mid 30s	research, school	EAGLE	educational blocks kit	Yes
P10	mid 30s	job (industry engineer), hobby	EAGLE	breakout board	Yes
P11	late 20s	job (research engineer), hobby	Altium	motherboards for chip tapeouts	Yes
P12	early 20s	job (industry engineer), hobby	Altium	power converter	Yes
P13	late 20s	research	EAGLE	embedded development board	No
P14	late 20s	job (industry engineer), hobby	DipTrace	debug adapter	No
P15	late 30s	job (industry engineer), hobby	Altium	general consumer electronics	No

Table 1: Summary of study participants.

All participants are familiar with the design process from idea to PCB, and all but one have completed at least one full project consisting of all those steps.

We recruited participants using two methods: personal referrals (7 participants), and relevant email lists (8 participants) such as those of a local makerspace, university design courses, and student groups. While the only criteria was some experience building PCBs in EDA tools, we did not recruit those working on highly complex designs to avoid a long tail of specialized issues. Participants were compensated with a $20 gift card for each of the interview study and mockup study.

4 INTERVIEW STUDY: METHODOLOGY

Interviews were semi-structured and start with background information, including motivations, designs, and views of flow from idea to final device. Based on those responses, we then go into depth on each step in the flow, examining tools used, pain points, references used, and general suggestions or comments. Interviews averaged 90 minutes with a standard deviation of 29 minutes, and were conducted either in-person at the participant's workplace or through videoconference.

Utilizing the principles of contextual inquiry [7], we asked for an example design to ground discussions when possible. A majority of participants were able to do so, but some could not because of confidentiality and lost files. Instead, we asked them to either visualize their designs or bring up stock schematic and board layout images.

Interviews were conducted by one interviewer and audiotaped with the participant's consent. One researcher, experienced with board design and familiar with most of the tools discussed, then conducted an open coding phase over the transcriptions, and further grouped codes into related topics [38]. From these, we looked for themes that both had design implications for EDA tools and either had support among multiple participants or were notable outliers.

5 INTERVIEW STUDY: FINDINGS

Participants provided rich data on their design flows, and how tools both did and did not support steps in those flows.

Design Flows

As shown in Figure 1, we broadly divide the design flow into these steps, in order: specification finding, system architecture development, and physical device iterations on a variety of media (including breadboards, milled PCBs, and commercially produced PCBs). Overall, each step incrementally refines the design to be more concrete, until finally a PCB can be produced. While there is a strict chain of dependencies between steps, designers regularly iterated and backtracked, especially in response to new information from testing and design.

Specification Finding. Determining the requirements and specifications for a device is a varied process that differed from user to user and from project to project. Specifications could capture a whole host of design goals including technical and functional requirements, user interactions, and aesthetic goals. These could be captured as drawings on a whiteboard, lists on documents and slide decks, or even a chip design that the system is built around. In many cases, these were living documents, with requirements and project scoping being a back-and-forth process where each edit forces many other changes down the line.

System Architecture Development. Specifications were then refined into a system architecture, represented as a block diagram. This serves as an intermediate step, translating from requirements into an implementation strategy.

The key distinguishing feature of this step is support for varying and mixed levels of abstraction.

> Each engineer will have what feels right for them. (P15)

Blocks in participants' architecture diagrams ranged from the generic ("accelerometer", "trigger circuit", or even just "sensors") to the specific (part numbers and subcircuit schematics). Three participants had examples that mixed abstractions on the same document, with some blocks being generic and others having part numbers. Some diagrams also indicated types of information flow between design elements such as communication buses or protocol information.

Drawings were overwhelmingly the most common representation: ten participants used either paper, whiteboards, or graphics software like PowerPoint and Visio. Schematic capture tools could also be used to produce nonfunctional diagrams, and two mentioned occasionally using EDA tools for this step. While digital tools gave designers powerful advantages including hyperlinking, cloud sharing, and backup, the unconstrained nature of drawings was most important:

> I feel very free to sketch in whatever language I want and whatever higher level I want. (P06)

Overall, participants generally enjoyed this step:

> I kind of like it. [...] It's a very creative area where somebody gives you requirements and you have the freedom to meet them however you see fit. [...] There's the creative freedom that you don't have once you get to the schematic and the layout. (P14)

Prototyping. Ten participants talked about a prototyping phase, which could be done with solderless breadboards, soldered protoboards, milled PCBs, or development boards and kits. Agility was a goal, which rapid prototyping machines could help with:

> I'm fortunate enough to have an LPKF [PCB mill] to mill the boards with. And that's been great. Usually the board goes through three or four revisions after soldering, so it's not just that, oh, I made one board and then it's done. (P03)

More generally, others also iterated on PCBs for their projects, with earlier boards acting as prototypes of the final design.

Prototypes were generally intended to validate some aspect of the design, though one participant also noted their value for exploring concepts and implementations. Validation was not limited to electrical functionality: mechanical characteristics, user feedback, and firmware development were also goals.

Schematic Capture. Schematic entry is where PCB design suites typically enter the design process.

Concerns here tended to be much lower-level, to the point where issues of schematic layout and readability were as common as those of circuit design and functionality. Participants noted the value of the schematic as a reference for later debugging or a document that should be shared with others. Aesthetics aside, messy designs could also conceal schematic errors or lead to bugs.

Mentions of manual transcription as part of the process were common – from either physical prototypes, or combining block diagrams with vendor-supplied reference schematics. While circuit designs in the abstract saw re-use, the inability to import data resulted in a time-consuming, tedious process. Yet, this was not completely devoid of designer input: reference designs may need to be adapted for the specific application through parts selection and component sizing. Quality and trust were also barriers to direct re-use: for example, worries about the quality of random Internet parts libraries or quirks in unofficial organization-wide reference designs.

Overall, attitudes about schematic entry were less positive:

> It's more of a necessarily evil. I wouldn't say it's a bad thing or a good thing, it's just like, I need to do this because otherwise I can't get my board. (P03)

Board Layout. Participant concerns during this phase were also low level and often related to the physical design and the final product: mechanical integration, signal integrity, manufacturability, and cost.

Despite both schematic capture and layout being part of the same EDA suite and schematic import being a common feature in layout tools, moving between schematic and layout was a notable source of friction. Five participants complained about the initial placement of components in layout:

> Altium kind of just barfs it out in a, not stacked on top of each other, but there's really not a lot of rhyme or reason. [...] It all seems pretty random. (P11)

Updating a layout after a schematic modification was also noted as problematic.

Participants also frequently consulted datasheets, placement rules, and routing guidelines during this step. While parts libraries and design guidelines could be shared between projects, layout re-use was rare. This was a result of limited tool support and projects needing customized layouts.

Parts Selection. Parts selection happened throughout the other stages of the design process. For example, critical parts

may be specified on the block diagram, while common parts like resistors may not be picked until just before ordering.

Concerns varied widely. Eight participants mentioned optimizing for cost, while anther worked in a price-insensitive industry. Three also preferred parts that were immediately available in their makerspace or research lab. Otherwise, there was a long tail of other concerns, including hand-solderability, stocking, RoHS compliance, or avoiding vendors in organization-wide blacklists.

Overall, this phase could require significant manual work and was deceptively difficult:

> It's something that I find to be challenging and I think that people underestimate, [...] everyone's like, "eh, whatever, you're just buying stuff" and then they realize like "oh, actually, just buying stuff is not super easy". (P05)

Iteration. As alluded to throughout, many concerns do not fit purely within one design phase. For example, participants mentioned going back and forth between layout and schematic to optimize pin assignments for routing, or re-designing the schematic to work around unavailable parts.

In general, while later steps are dependent on the results of earlier steps, those results are not always locked down. As an extreme example, one participant recalls being told:

> Hey, you made this great device to guarantee these specs, but we really need this new part and it kind of breaks the spec that we gave you before. Deal with it. (P12)

One strategy participants used to deal with this was defensive design. This included defending against mistakes and errors, such as by inserting optional jumpers between sub-circuits to allow modification or removal of connections, and defending against specification changes, such as by picking a microcontroller with a wide peripheral set for flexibility.

Use of Automation

Participants talked about their experiences using automation features provided by their tools. These features aim to minimize errors and ease tedious tasks, and fell into the broad categories of design verification and routing assistance.

Design Verification. EDA suites generally include electrical rules check (ERC), which checks schematics for common issues, and design rules check (DRC), which checks layouts for manufacturability.

ERC is commonly implemented by assigning pin classes (for example: input, output, or bidirectional) and defining a matrix of legal connections. Opinions were varied: six mentioned using this feature (all with caveats), while five specifically mentioned not using it. While electrical rules checking has utility in catching some simple mistakes like unconnected wires, the limitations were significant:

> A lot of false negatives. And false positives. Very few true positives. (P07)

On the other hand, no one mentioned skipping layout-versus-schematic or DRC, both of which are generally very accurate. Complaints were limited to bugs, like not catching split ground planes.

Routing Assistance. Only two participants reported using autorouting, all limited to simple designs. The general view was that the benefits were not worth the time costs of setting up the job properly or fixing poor results.

However, mixed-initiative, assistive routing features were well-received. These include online or interactive DRC, which does manufacturability checks on traces as they are placed, and smart routing tools like push-and-shove, which allow the trace being placed to intelligently displace existing traces.

> The auto routers are [terrible], the auto placements are [terrible]. It's a highly manual process. I like push and shove routing, those are great. (P12)

Tool Selection

We also asked participants about why they chose their particular EDA suite. Community effects dominated: their choices were influenced by the tools used by their friends or teams, the tools taught in class, and the existence of an ecosystem of tutorials and libraries. For those in industry, widespread usage was also important for compatibility with contractors and ease of hiring.

Those using or switching to KiCad noted the benefits of open-source software, such as lack of vendor lock-in, ease of sharing, and perpetual access to designs.

Summary

Overall, our main takeaway is that much of the interesting and creative work happens through a combination of system architecture and parts selection. Past that, schematic capture tends towards elaborating the system architecture by mashing in reference circuits, but the lack of design re-use results in a tedious and time-consuming process.

Links across steps are also major sources of friction. While converting a paper system architecture into a digital schematic is burdensome but unavoidable, moving between schematic, layout, and parts selection was just frustrating.

6 CONCEPT DESIGN

Our core insight from the interviews, then, is for designers to work at the system architecture level. This higher level of abstraction captures the essential design intent without being mired in implementation details.

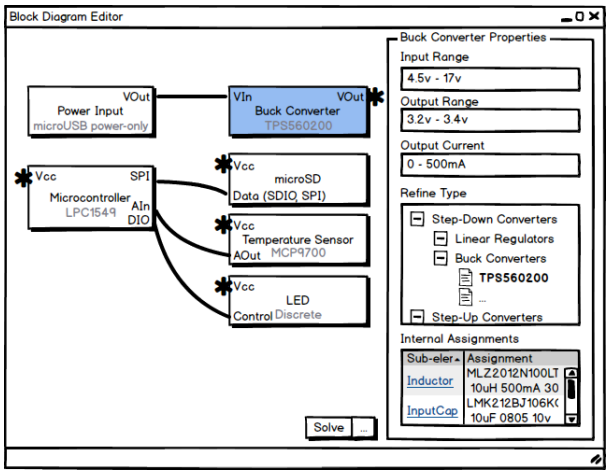

Figure 2: Mockup of the block diagram interface, showing the system architecture of the datalogger example design. The details pane on the right shows information on the selected buck converter: modeled operating parameters, selected and alternate implementations, and parts selections internal to the block. Showing operating parameters demonstrates how the system ensures design correctness.

We note that this strategy has support in the creativity support tools literature [31], satisfying the principles of *supporting exploration* by reducing the design effort and *designing for designers* by being grounded in actual workflows.

Interfaces

Such a tool must allow users to both design their system architecture and build the libraries of block implementations needed to transform high-level designs into a complete schematic.

Ambiguity in Block Diagrams. As schematics are fundamentally block diagrams, the interactions and interfaces from today's schematic capture tools provide a solid and familiar starting point. In our use case, these block diagrams would also need to scale between multiple levels of abstraction. At the lowest level, blocks would still represent individual components, but at higher levels, blocks would be sub-circuits. While many tools already support this with the notion of *hierarchy blocks*, additional features are necessary to support the system architecture level of design.

Primarily, we need support for ambiguity. While current schematics must be fully defined down to the last part, system architecture diagrams in our interviews tended to encode minimalist design intent, leaving many decisions open. An example would be labeling a block generically as "accelerometer" instead of with a specific part number.

This ambiguity further provides opportunities for tools to automate the currently-manual and sometimes-tedious parts selection process. Recent work in synthesizing schematic from high-level specifications, including EDG [30] and Trigger-Action-Circuits [2] demonstrate the technological feasibility of this approach. As participants generally optimized for some criteria (commonly, but not always, cost) during their parts selection process, tools should also optimize for an user-defined objective function. Alternatively, the system could generate and display a shortlist of alternatives as in Trigger-Action-Circuits, though they reported mixed results with their novice participants.

An underlying constraint-based data model, as described in EDG, works well here. Types of components, like "accelerometer", would be just one aspect that could be constrained. More powerfully, such a system allows users to directly enter functional specifications, such as the minimum required bandwidth of said accelerometer. This also gracefully handles nonuniform ambiguity, which we observed from diagrams containing a mix of generic blocks as well as specific part numbers.

Supporting Libraries. An unambiguous high-level design still must be combined with implementations of used blocks to form a layout-ready schematic. However, our interviews show this to be a major issue: there usually aren't libraries of block implementations, and designers generally have to transcribe from datasheet reference circuits. Practical solutions must also incentivize the creation and sharing of re-usable libraries, either by making the process easier or by providing additional value for designers.

In any tool responsible for parts selection, libraries would need to model parts to a sufficient degree to check correctness of the entire system. As with EDG, electrical characteristics like absolute maximum ratings could be encoded in a block's constraints. This would automate some of the currently-manual checks mentioned by our participants, such as voltage and current compatibility. Furthermore, these checks could address one of the primary drawbacks of ERC, being more accurate than current pin-type based schemes.

One roadblock is that reference circuits often need to be customized for each application. In these cases, static hierarchy blocks would preclude any meaningful re-use. However, a generator methodology may be the solution: encoding the rules for generating a block implementation instead of fixed, static instances. As a simple example, a generator for a LED circuit would contain the logic to size the resistor given the LED current and voltages.

Generators built using hardware construction languages (HCLs) have been used for both chips [18] and PCBs [5]. Despite the limited exploration of their usability, HCL based generators show significant promise as an abstraction. We

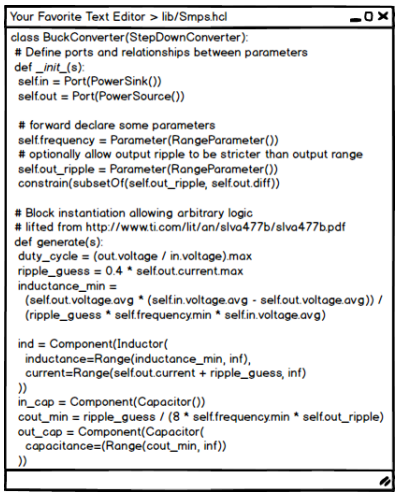

Figure 3: Mockup of the hardware construction language interface, showing the implementation of a buck converter generator. The first block of code, in `__init__`, defines the block interface: ports and constraints between parameters on those ports. The second block of code, in `generate`, contains the logic for instantiating sub-components once the block interface has been fully resolved. Here, this consists of equations transcribed from a buck converter datasheet.

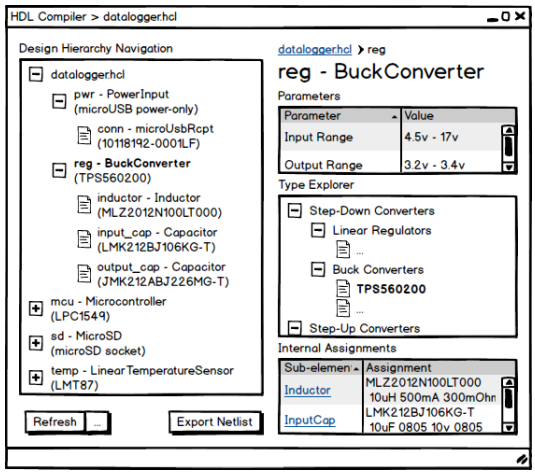

Figure 4: Mockup of the hardware construction language compiler interface. This provides information similar to the detail pane in the block diagram interface, but using a tree view for navigation in absence of block positioning data.

further note that, as is common in the chip industry, these textual representations can be applied to top-level designs as well. This may be highly advantageous in some cases: for example, instantiating large arrays of LEDs becomes trivial.

Allowing recursive ambiguity, where block implementations can contain further ambiguous blocks, can also be helpful. Reference schematics may be ambiguous: for example, even given a specific accelerometer, its reference circuit may include capacitors that do not have part numbers. This also aligns well with some observed design practices, where common passives are not chosen until ordering.

Interface Mockups

An example of the augmented block diagram interface is shown in Figure 2. This shows a potential system architecture for a datalogger that records temperature data to a microSD card. Designers would be able to specify blocks that can range from the generic, as with the temperature sensor, to the specific, such as the microSD card socket. When a specific part number is needed for a generic block, the user could either allow the tool to automatically choose, or refine individual blocks from a list of compatible parts.

An example of the HCL approach is shown in Figure 3. Our example design for a buck converter generator illustrates how current barriers to reuse in schematic tools are addressed. The parameters in the block interface specify what the subcircuit needs to do, while the constraints ensure design consistency by defining limits and how parameters propagate. The arbitrary code in the generator can then build customized subcircuits applicable in many different designs, for example by encoding the component sizing equations taken from the datasheet.

7 MOCKUP STUDY: METHODOLOGY

As building the proposed design tool is a nontrivial engineering task, we believe it is important to validate and refine our design first. In particular, we want to understand whether users would find this abstraction useful, and more importantly, their reasoning and any limitations.

To do so, we built clickthrough mockups of design flows through an example project spanning the two interfaces described above. These mockups allowed us to talk concretely with a visual aid that conveys similarities to conventional EDA tools, but without requiring the full system that any meaningful interactivity would require. We choose a datalogger as our example project because they have real-world applications, and balance easy participant comprehension with being complex enough for better tooling to be meaningful. The example system architecture, including the choice of blocks, are modeled off of observed diagrams.

After some preliminaries, we presented our participants with an empty canvas in the block diagram interface. From there walked participants through instantiating the high-level architecture from parts libraries before asking the tool to fully solve the design as shown Figure 2. The finalized

design is equivalent to a full schematic, which we explain as being directly exportable to a layout tool like KiCad.

We then move into the HCL mockups, first showing a one-to-one transcription of the datalogger high-level design in code as a conceptual bridge. Further examples demonstrate the power of HCLs, first showing array instantiation of temperature sensors using a for loop, then showing the buck converter generator in Figure 3. We provide inspection into the solved design through the compiler interface in Figure 4. A tree view replaces the block diagram view as the HCL does not encode block placement and layout information, and the block properties pane becomes read-only.

We asked open-ended questions about advantages, disadvantages, and applicability, particularly compared against each other or conventional tools. We also asked about acceptable solver runtimes. For the HCL, we further asked about the utility of a hypothetical schematic visualization view and what kind of additional verification users would perform. The latter gets at notions of trust in the tool and libraries.

We purposefully used a sketch-like art style to key participants to focus more on design abstractions instead of UI specifics [16]. Additionally, by having participants compare-and-contrast between two interfaces, and asking for the rationale behind answers, we hope to reduce the effects of acquiescence bias [36]. This is especially relevant for those recruited through personal referrals.

Otherwise, the interview and analysis procedure were the same as the initial interview study. Ten of the original interview participants, as described in Table 1, returned for this follow-up study. Interviews averaged 46 minutes with a standard deviation of 14 minutes.

8 MOCKUP STUDY: FINDINGS

Participants were generally enthusiastic about the system architecture level of abstraction for its ability to reduce manual work, but noted concerns about increased design automation.

Advantages

Automated design verification, essentially a more powerful ERC, was the most common advantage, mentioned by 5 participants. These automated checks reduce the chance of an uncaught error making it to fabrication while the encapsulation of datasheet parameters allowed replacement of the tedious manual verification process.

A related advantage, mentioned by three participants, was the integration of parts data into the main design flow:

> It does all of the parameter searching, and comes up with an appropriate part, which is what I do anyway just on Digi-Key, which doesn't have a very friendly user interface that is not tied closely into the design. (P11)

Designing at the system architecture level also provided advantages. Three participants noted the similarity to their existing processes, that this was part of their existing flow:

> I'm already generating some visual representation that's generated in software. If that can connect me to my other things, then I would really value that. (P05)

Not only would automated linkages from system architecture to schematic to layout save time, but it could eliminate mistakes during manual transcription. Errors where something is forgotten entirely during transcription were especially insidious, compared to design correctness issues which were more likely to be caught during inspection.

Finally, some participants brought up additional benefits with the HCL interface. Two talked about automated consistency throughout the design even as other parts change, and another noted that the methodology used for manual calculations was often not kept and must be rediscovered if needed later.

Limitations

The most common concern, mentioned by six participants, was a requirement for or dependence on quality libraries. Missing parts could either be invisible, especially for users solely relying on the system, or difficult to build.

All participants were inclined to share their libraries, but some noted limitations like concerns about competitiveness (especially if sharing uncommon parts), employer policy, and quality bars. Reasoning for this general attitude ranged from open-source philosophy to the practical benefits of community contributions. However, one participant expressed doubt about whether part manufacturers would contribute to a system that interoperates with competitors' parts.

Correctness was also a commonly cited criteria, especially since the tool introduces generative features:

> You're automating design here. That is, it's hard to do and it requires a lot of trust. (P07)

Discussions of trust in the overall tool were generally implicit: all participants mentioned doing some kind of verification on the generated output, from connectivity-based spot checks to comparing against datasheet specs. Sometimes, these statements would be qualified: one mentioned being thorough the first time, while two others suggested building trust by having the system show its work by generating report including the data sources and rules behind checks.

Trust in the libraries themselves was also a key part of trusting the tool. Of the five who talked about this, four mentioned trusting libraries from the part manufacturer or reputable organizations like Digi-Key and Adafruit. Trust in community libraries was mixed and based on a variety of heuristics, such as attention to detail and spot checks

against datasheets. Community feedback was another aspect, including rating systems and indications of successful builds.

Finally, even the higher level of abstraction still requires nontrivial engineering knowledge:

> Beginners don't understand the difference between buck and boost and current and max and minimum footprint space versus component cost. (P10)

Blocks vs. HCL

While all participants were able to follow and understand the HCL examples, they also talked about trade-offs with the block diagram interface.

When asked about use cases, there were (predictably) mentions of parameterization and repetitive designs for the HCL. However, there were also mentions of its unsuitability for designs where its capabilities are not required, such as connectivity-driven or straightforward designs. One participant made the observation that:

> [The HCL] feels less kind of exploratory. It feels more like something I'd do if I already have sketched out something on paper, and then I need to figure out the components. [...] [The block diagram interface] feels almost like, to be a little bit abstract, it feels less serious, right? Because you're kind of working with these graphical representations, whereas this is code. (P04)

Participants were more critical of the HCL, with five mentioning the learning curve as a disadvantage. Four also mentioned the code representation as more difficult to work with, instead preferring a visual schematic. One described the HCL as completely unusable, though could still see value for large repetitive operations.

Participants had mixed feelings about textural interfaces. Two believed it would be faster, though one thought that even writing a for loop would be slower than operations in the block diagram interface. Another noted benefits of compatibility with version control tools and text editors.

Finally, one participant recognized that it is not an either-or situation, correctly noticing the possibility of using a GUI to define constraints. As both the block diagram interface and the HCL are built on top of the same data model, both could support constraints with the appropriate interface elements.

Running Time

Thoughts about acceptable solver running times largely fell into two broad groups: interactive, generally on the order of seconds, and batch, which spanned minutes to hours. An equal number of participants were in each group.

Those who wanted interactive runtimes pointed to the responsiveness of existing board design tools and modern websites as justification. They also suggested a modified version of the mockup flow to achieve these speeds, such as solving a subset of the design, or incrementally solving for design changes.

Those participants who were comfortable with batch processes cited both the time savings of automation as well as avoiding manual tedious work. Three mentioned benchmarking against manual processes, such as parts search. Another talked about the idea of active time and background time: while manual verification of a schematic requires active attention to the problem, one could attend to other tasks while waiting for the solver to complete in the background.

Summary

First, our results suggest that designers have two primary concerns when evaluating new tools: correctness and design effort. However, both must be evaluated holistically, across the entire design flow.

While the integration of parts data from datasheets provides a correctness advantage from a more powerful ERC, designing at the system architecture level also eliminates an error-prone manual transcription step from paper. Both also provide an important speed advantage: the higher level of abstraction also frees users from worrying about details that a computer could solve, and block re-use reduces time spent reinventing the wheel. However, trust in both the system and libraries was a major concern, but could be earned through visibility into automated processes and community feedback mechanisms.

Second, reliable partial automation seems to be preferable to unreliable full automation. The initial interviews hint at this, with participants preferring assistive push-and-shove routing to fully automated routing. We see a similar trend in the mockup study, where participants were happy with the incrementally higher level of abstraction instead of pushing for, say, full synthesis from system requirements.

It may be useful to view the balance of user effort and system effort as a multi-dimensional trade-off, in terms of factors such as user time required, tediousness of tasks, expressiveness of abstractions, and feasibility of automation. For example, asking the user to further constrain a design to reduce the search space for runtime reasons may be a reasonable strategy.

Finally, based on the feedback from the mockup user study, we believe that our concept design is a good starting point for the designers of future tools.

9 FUTURE WORK

While we address what we think are the highest-impact and lowest-hanging fruit in electronics design tools with

our concept, both our interviews and principles for creativity support tools [31] suggest that these considerations are worth further investigation:

Iteration. Designers tended to iterate, both within the EDA suite, such as optimizing between schematic and layout, and through physical prototypes. Our concept only tangentially addresses iteration through refining constraints of the example design.

Open Interchange. We observed two design flows involving significant use of external tools (Inkscape and chip design suites), and there are likely to be more highly custom workflows. However, supporting these may be more of an implementation detail, by documenting file formats or exposing programming interfaces.

Community and Collaboration. Community effects were a large factor throughout both the initial and mockup interviews. Library quality and availability were emphasized in the mockup responses, but both may ultimately depend on the existence of a vibrant community. How to encourage the formation of, and sharing within, such a community may be as important as the tool design itself.

Enabling Library Creation. Tooling may also encourage creating libraries by partially automating turning datasheets into machine-readable data. For example, uConfig [10] is able to extract pinout data from PDFs for datasheets from certain vendors. Furthermore, tools might also parse the highly-regular electrical characteristics tables, and populate block model parameters.

Beyond the Schematic. While our concept primarily addresses schematic capture, the interviews also suggest improvements to other stages like layout. Additionally, there may be value in persisting ambiguity past schematic capture, such as to optimize for layout area.

Beyond Electronics

While this study was conducted in the context of PCB design tools, the findings and recommendations may be applicable to other domains. The ideas of incrementally raising the level of abstraction, specification and utilization of ambiguity in design, and eliminating tedious transcription work through better integration can generalize to any design domain. Knowing the limitations and requirements of these approaches, such as the need for trustworthy automation, will be important to building practical systems.

Our exploration comparing and combining visual interfaces and programming languages can also inform other design domains. One application may be in mechanical CAD, where parameterized parts could be defined in a powerful generator language, akin to OpenSCAD, and instantiated in a visual assembly-level interface.

10 LIMITATIONS

Ultimately, electronics design is a very broad field with many specialties. While we chose to address PCB design in general because of its ubiquity, we also realize that tools tailored for a particular subdomain may be more powerful.

Our goal of looking at entire PCB design flows also trades depth for breadth. An interview study meant a fairly high-level investigation and would be subject to participant recall limitations. However, our findings could form the basis for more targeted observational studies of particular steps.

Participants also tended to be younger (early 20s to late 30s), likely as a result of a majority of the mailing lists being university-affiliated. However, given the observation of similar design flows and repeated themes, we believe that we have at least found some interesting directions for future work. The data here could also form the basis for wider studies that use more scalable methods such as surveys.

Finally, while the feedback on our tool concept was largely positive, it is far from proof that it is useful, usable, or even feasible. Though we try to maintain an internal consistency in our mockups and construct a representative design flow, they are only our best effort at imagining what such a tool would look like without actually building it. However, now knowing that we are not horribly off the mark, we can build the system and test its effectiveness with user studies encompassing a broad range of applications.

11 CONCLUSION

In order to discover what novel approaches to PCB design tools are fruitful, we survey a diverse group of PCB designers over two rounds of interviews.

We start by examining participants' design flows as they moved from idea to physical device, and the tools that supported their work. We found that much of the creative design process happened during system architecture design and parts selection, without the support of the existing tool ecosystem. Defining features of this level of design include mixed levels of abstraction and design ambiguity.

With these lessons, we envision an augmented block diagram and a hardware construction language interfaces that better support this level of design. These are inspired by successes from other domains with feasibility grounded in prior research.

Feedback from mockups of design flows through those concept tools indicate that integrating previously separate data sources, like paper diagrams and datasheets, could enable large gains in design time and correctness. Furthermore, as tools automate more design, trust becomes a central issue.

We believe that these findings and concept designs will lay the foundation for a new generation of tools that enables designers to work more efficiently and effectively.

12 ACKNOWLEDGMENTS

This work was supported in part by NSF awards CNS 1505728 and IIS 1149799; Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award Number DE-AR0000849; ADEPT Lab industrial sponsor Intel; ADEPT Lab affiliates Google, Siemens, and SK Hynix; and the Camozzi Group via the iCyPhy consortium. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

REFERENCES

[1] Altium. 2018. Altium Designer. https://www.altium.com/altium-designer/

[2] Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2017. Trigger-Action-Circuits: Leveraging Generative Design to Enable Novices to Design and Build Circuitry. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 331–342. https://doi.org/10.1145/3126594.3126637

[3] Arduino. 2018. Arduino - Home. https://www.arduino.cc

[4] Autodesk. 2018. EAGLE | PCB Design Software. https://www.autodesk.com/products/eagle/overview

[5] Jonathan Bachrach, David Biancolin, Austin Buchan, Duncan W Haldane, and Richard Lin. 2016. JITPCB. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2230–2236. https://doi.org/10.1109/IROS.2016.7759349

[6] Hedayat Markus Bayegan and Einar Aas. 1978. An Integrated System for Interactive Editing of Schematics, Logic Simulation and PCB Layout Design. In *Proceedings of the 15th Design Automation Conference (DAC '78)*. IEEE Press, Piscataway, NJ, USA, 1–8. http://dl.acm.org/citation.cfm?id=800095.803058

[7] H. Beyer and K. Holtzblatt. 1998. *Contextual Design: Defining Customer-centered Systems*. Morgan Kaufmann. https://books.google.com/books?id=T8pcH4QjATkC

[8] Tracey Booth, Simone Stumpf, Jon Bird, and Sara Jones. 2016. Crossed Wires: Investigating the Problems of End-User Developers in a Physical Computing Task. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 3485–3497. https://doi.org/10.1145/2858036.2858533

[9] Cadence. 2018. Allegro PCB Designer. https://www.cadence.com/content/cadence-www/global/en_US/home/tools/pcb-design-and-analysis/pcb-layout/allegro-pcb-designer.html

[10] Sébastien Caux. 2018. uConfig. https://github.com/Robotips/uConfig.

[11] Joshua Chan, Tarun Pondicherry, and Paulo Blikstein. 2013. LightUp: An Augmented, Learning Platform for Electronics. In *Proceedings of the 12th International Conference on Interaction Design and Children (IDC '13)*. ACM, New York, NY, USA, 491–494. https://doi.org/10.1145/2485760.2485812

[12] 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 (UIST '16)*. ACM, New York, NY, USA, 677–686. https://doi.org/10.1145/2984511.2984566

[13] EDASolver. 2016. EDASolver: Welcome to Functional EDA. https://edasolver.com

[14] A. C. Finch, K. J. Mackenzie, G. J. Balsdon, and G. Symonds. 1985. A Method for Gridless Routing of Printed Circuit Boards. In *Proceedings of the 22Nd ACM/IEEE Design Automation Conference (DAC '85)*. IEEE Press, Piscataway, NJ, USA, 509–515. http://dl.acm.org/citation.cfm?id=317825.317937

[15] 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 (UIST '13 Adjunct)*. ACM, New York, NY, USA, 19–20. https://doi.org/10.1145/2508468.2514936

[16] Saul Greenberg and Bill Buxton. 2008. Usability Evaluation Considered Harmful (Some of the Time). In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 111–120. https://doi.org/10.1145/1357054.1357074

[17] Kotaro Hara, Christine Chan, and Jon E. Froehlich. 2016. The Design of Assistive Location-based Technologies for People with Ambulatory Disabilities: A Formative Study. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 1757–1768. https://doi.org/10.1145/2858036.2858315

[18] A. Izraelevitz, J. Koenig, P. Li, R. Lin, A. Wang, A. Magyar, D. Kim, C. Schmidt, C. Markley, J. Lawson, and J. Bachrach. 2017. Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 209–216. https://doi.org/10.1109/ICCAD.2017.8203780

[19] KiCad. 2018. KiCad EDA. http://kicad-pcb.org/

[20] 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 (TEI '09)*. ACM, New York, NY, USA, 351–358. https://doi.org/10.1145/1517664.1517735

[21] F. B. Lavering. 1964. AUTO CARD Automated Printed Circuit Board Design. In *Proceedings of the SHARE Design Automation Workshop (DAC '64)*. ACM, New York, NY, USA, 9.1–9.29. https://doi.org/10.1145/800265.810745

[22] Ola A. Marvik. 1979. An Interactive Routing Program with On-line Clean-up of Sketched Routes. In *Proceedings of the 16th Design Automation Conference (DAC '79)*. IEEE Press, Piscataway, NJ, USA, 500–505. http://dl.acm.org/citation.cfm?id=800292.811760

[23] Andrew J. Matthews. 1977. A Human Engineered PCB Design System. In *Proceedings of the 14th Design Automation Conference (DAC '77)*. IEEE Press, Piscataway, NJ, USA, 182–186. http://dl.acm.org/citation.cfm?id=800262.809124

[24] Will McGrath, Daniel Drew, Jeremy Warner, Majeed Kazemitabaar, Mitchell Karchemsky, David Mellis, and Björn Hartmann. 2017. Bifröst: Visualizing and Checking Behavior of Embedded Systems Across Hardware and Software. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 299–310. https://doi.org/10.1145/3126594.3126658

[25] David A. Mellis, Leah Buechley, Mitchel Resnick, and Björn Hartmann. 2016. Engaging Amateurs in the Design, Fabrication, and Assembly of Electronic Devices. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems (DIS '16)*. ACM, New York, NY, USA, 1270–1281. https://doi.org/10.1145/2901790.2901833

[26] Mentor. 2018. Xpedition Enterprise. https://www.mentor.com/pcb/xpedition/

[27] Martez E. Mott, Jane E., Cynthia L. Bennett, Edward Cutrell, and Meredith Ringel Morris. 2018. Understanding the Accessibility of Smartphone Photography for People with Motor Impairments. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing*

Systems (CHI '18). ACM, New York, NY, USA, Article 520, 12 pages. https://doi.org/10.1145/3173574.3174094

[28] Brant Nelson, Brad Riching, and Josh Mangelson. 2012. Using a Custom-Built HDL for Printed Circuit Board Design Capture. PCB West 2012 Presentation.

[29] David Rager and Herb Weiner. 1978. The Design of a Dense PCB Using an Interactive DA System. *SIGDA Newsl.* 8, 2 (June 1978), 50–58. https://doi.org/10.1145/1061458.1061464

[30] 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 (SCF '17)*. ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3083157.3083159

[31] 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).

[32] Hiroshi Shiraishi, Mitsuo Ishii, Shoichi Kurita, and Masaaki Nagamine. 1982. ICAD/PCB: Integrated Computer Aided Design System for Printed Circuit Boards. In *Proceedings of the 19th Design Automation Conference (DAC '82)*. IEEE Press, Piscataway, NJ, USA, 727–732. http://dl.acm.org/citation.cfm?id=800263.809282

[33] Ben Shneiderman. 2002. Creativity Support Tools. *Commun. ACM* 45, 10 (Oct. 2002), 116–120. https://doi.org/10.1145/570907.570945

[34] Ben Shneiderman. 2007. Creativity Support Tools: Accelerating Discovery and Innovation. *Commun. ACM* 50, 12 (Dec. 2007), 20–32. https://doi.org/10.1145/1323688.1323689

[35] Ben Shneiderman. 2009. Creativity support tools: A grand challenge for HCI researchers. *Engineering the User Interface* (2009), 1–9.

[36] Maryam Tohidi, William Buxton, Ronald Baecker, and Abigail Sellen. 2006. Getting the Right Design and the Design Right. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. ACM, New York, NY, USA, 1243–1252. https://doi.org/10.1145/1124772.1124960

[37] 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 (UIST '16)*. ACM, New York, NY, USA, 687–695. https://doi.org/10.1145/2984511.2984527

[38] R.S. Weiss. 1995. *Learning From Strangers: The Art and Method of Qualitative Interview Studies*. Free Press. https://books.google.com/books?id=i2RzQbiEiD4C

[39] Te-Yen Wu, Hao-Ping Shen, Yu-Chian Wu, Yu-An Chen, Pin-Sung Ku, Ming-Wei Hsu, Jun-You Liu, Yu-Chih Lin, and Mike Y. Chen. 2017. CurrentViz: Sensing and Visualizing Electric Current Flows of Breadboarded Circuits. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 343–349. https://doi.org/10.1145/3126594.3126646

[40] Te-Yen Wu, Bryan Wang, Jiun-Yu Lee, Hao-Ping Shen, Yu-Chian Wu, Yu-An Chen, Pin-Sung Ku, Ming-Wei Hsu, Yu-Chih Lin, and Mike Y. Chen. 2017. CircuitSense: Automatic Sensing of Physical Circuits and Generation of Virtual Circuits to Support Software Tools.. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 311–319. https://doi.org/10.1145/3126594.3126634