SchemaBoard: Supporting Correct Assembly of Schematic Circuits using Dynamic In-Situ Visualization

Yoonji Kim[1], Hyein Lee[1], Ramkrishna Prasad[1], Seungwoo Je[1], Youngkyung Choi[1], Daniel Ashbrook[2], Ian Oakley[3], Andrea Bianchi[1]
Industrial Design KAIST[1],
Human Centred Computing University of Copenhagen[2], Human Factors Engineering UNIST[3]
{yoonji.kim; hyein.l; rprasad; seungwoo_je; youngkyung.choi; andrea}@kaist.ac.kr[1],
dan@di.ku.dk[2], ian.r.oakley@gmail.com[3]

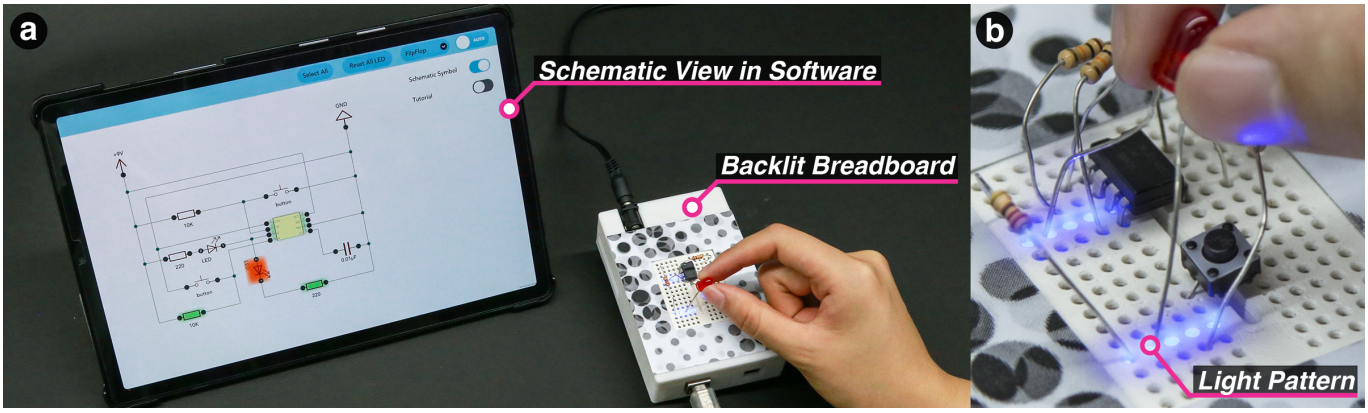

Figure 1. SchemaBoard is an LED-backlit solderless breadboard that can programmatically illuminate its rows to reflect the components or nets that are interactively selected in a schematic shown in a companion software application on a tablet PC. Users can be cued to place or check components based on the rows lit up in response to touching schematic symbols or pins on the circuit diagram shown on the screen of the tablet.

ABSTRACT

Assembling circuits on breadboards using reference designs is a common activity among makers. While tools like Fritzing offer a simplified visualization of how components and wires are connected, such pictorial depictions of circuits are rare in formal educational materials and the vast bulk of online technical documentation. Electronic schematics are more common but are perceived as challenging and confusing by novice makers. To improve access to schematics, we propose SchemaBoard, a system for assisting makers in assembling and inspecting circuits on breadboards from schematic source materials. SchemaBoard uses an LED matrix integrated underneath a working breadboard to visualize via light patterns where and how components should be placed, or to highlight elements of circuit topology such as electrical nets and connected pins. This paper presents a formative study with 16 makers, the SchemaBoard system, and a summative evaluation with an additional 16 users. Results indicate that SchemaBoard is effective in reducing both the time and the number of errors associated with building a circuit based on a reference schematic, and for inspecting the circuit for correctness after its assembly.

UIST'20, October 20–23, 2020, Minneapolis, MN, USA

ISBN 978-1-4503-6708-0/20/04…$15.00

DOI: https://doi.org/10.1145/3313831.XXXXXXX

Author Keywords

Physical computing; circuits; breadboard visualization; system.

CCS Concepts

•Human-centered computing → Human computer interaction (HCI);

INTRODUCTION

The ready availability of cheap sensors, components, and platforms such as Arduino[1] has encouraged many creators and makers without formal electrical engineering training to tackle circuit design and assembly tasks. Open-source tools, such as the Fritzing project [16] and Tinkercad by Autodesk[2], have been crucial in enabling non-technical users in the process

[1] https://arduino.cc
[2] https://www.tinkercad.com

of making, sharing, and producing electronic circuits. A key enabling feature in both Fritzing and Tinkercad is a simplified visualization of the circuit topology. Traditional circuit designs are most frequently depicted in *schematic* form, an abstract representation of the electrical connections between components [24]. In contrast to this, Fritzing and Tinkercad, present a *pictorial*, or literal, view of the circuit, with images of the components as they physically appear, often in conjunction with images of a physical breadboard and connecting wires. The simplicity of this depiction has led to it becoming a defacto standard in maker communities—it is often the preferred circuit representation in guidebooks [5, 10, 20], tutorials[3] and maker project documentation[4]; it is also widely deployed in the research community among groups seeking to develop new tools or tool kits for circuit development [1, 9, 15, 18, 26].

Despite their ubiquity in the maker community, pictorial representations of circuits suffer from two main problems. Firstly, pictorial diagrams are limited in the complexity and scale of the circuits they can describe. Pictorial diagrams rapidly become messy and difficult to read [8], and although this problem can be mitigated by careful adherence to clear circuit style practices [8], the adoption of such standards takes time and requires training. Secondly, although pictorial circuits are common in online tutorials for makers, they are rare in formal educational material (e.g., textbooks) and the vast bulk of online technical documentation. Electronic schematics are therefore often considered as a "necessary evil' [17], because a maker seeking to build a complex circuit or independently study electronics cannot avoid them. However, the abstract nature of electronic schematics presents a substantial problem that hinders their adoption among makers: the transition between schematic and physical circuits is not as obvious as with pictorial representations, making the process of physically assembling a circuit on a breadboard or inspecting its correctness difficult. This task is well studied [4, 8], and researchers report it to be difficult, especially as the number of nets in a circuit increases [28].

In this paper, we address this problem by trying to improve the link between schematic and breadboard representations. We present *SchemaBoard*, an enhanced physical breadboard that incorporates software controlled visual highlighting of specific pin rows. It supports circuit assembly tasks such as accurate component placement and inspection of electrical nets directly on the breadboard. By using in-situ visual cues to highlight parts of a breadboarded circuit, SchemaBoard directly depicts the mapping between the semantic representations shown in schematic diagrams and the arrangements of physical components, wires, pins, and nets that form actual physical circuits. We argue that this more explicit mapping simplifies and speeds up the assembly and inspection of circuits on breadboards. We presents a formative study with 16 users that motivated SchemaBoard's design, the technical details of the system, and a summative evaluation with 16 additional participants that demonstrates the benefits of our approach.

[3] https://learn.adafruit.com
[4] https://www.hackster.io

RELATED WORK

Tools for circuit construction

Building a circuit is a challenging task for makers [4] that requires them to combine knowledge about the specific components that are needed, the appropriate values of those components (e.g., resistance or capacitance) and the way they should be connected. After an initial circuit is assembled, it typically needs to be inspected by checking connections and wiring until it is functional. Due to the complexity and interleaved nature of these tasks, makers find it difficult to determine the precise source of errors [4]. Mellis et al. [21] also found that debugging components and their connections on a breadboard are challenging for amateurs and presented opportunities for tools to help them, such as automatic assembly. Indeed, the severity of these problems (and the high level of knowledge required to build circuits in general) has led to numerous educational tools that seek to lower the bar to entry in this area. A common approach is to create modular tools and systems that allow circuits to be constructed manually by snapping together [2], plugging in [23] or virtually linking [6] component elements. These approaches, differently from SchemaBoard, are limited in that they require predefined (non-standard) components. CircuitStack [26] supports the integration of inkjet-printed circuits into breadboards to entirely avoid the need for wiring. Circuits are printed with conductive ink and clamped to the base of a special breadboard in order to implement a functional circuit. However, this system still requires users to manually assemble circuits by placing components on a breadboard, with the additional challenge of having to match unmarked physical locations on the breadboard (i.e., without visible wires) against those shown in a reference diagram.

Other tools for circuit construction focus on learning. For example, ElectroTutor [27] and HeyTeddy [14] allow novices to use standard hardware components with the Arduino UNO and provide structured unit-testing frameworks that check the correctness of their circuit, their code, and their understanding. CircuitStyle [8] aims to encourage and support good circuit construction practices through a web-based tool that provides authoring functionalities and interactive live tutorials. Finally, Belluci et al. [3] reported initial findings of an Augmented Reality (AR) prototype that guides the placement of components on a breadboard, given a pictorial representation of the circuit. This system does not work with schematic diagrams, and does not support placement of components on crowded breadboards, when the users' hands or other components occlude the tracking camera. In contrast with these examples, SchemaBoard does not focus on learning, but rather on improving the assembly and inspection of circuits on a breadboard, given an arbitrarily complex reference schematic diagram.

Circuit inspection and augmented breadboards

Incorrect assembly of circuits is common [4], and the resultant errors can usually only be identified by a slow and tedious manual inspection of the breadboard connections. Several research projects have sought to improve this process by preventing (or minimizing) circuit design errors. For example, AutoFritz [18] interactively provides wiring suggestions to minimize user

errors while Scanalog [25] and Trigger-Action-Circuits [1] suggest and visualize circuit designs based on high-level descriptions of behavior specified with graphical programming languages. In these latter systems, user input relates mainly to desired circuit function rather than low-level implementation details. While effective, we note that these projects do not directly support correct circuit assembly, but rather focus on providing clear pictorial representations of circuits that can serve as more accessible alternatives to schematics.

Another common approach to circuit inspection is via augmenting breadboards. One theme is to support live, automatic measurement of electrical properties, such as voltage [9,22] or current [29], at different points on a breadboard. Bifröst goes one step further by capturing and visualizing both hardware and software behavior in a single tool [19]. Visualizations of measured data are typically presented in-situ on the breadboard via LED lights [22], interactively via an on-screen pictorial schematic [29] or both [9]. In contrast to SchemaBoard, these projects are intended to support debugging of operational circuits, and are thus reliant on a user's ability to correctly assemble the circuits under test. Furthermore, these projects do not offer any explicit support for the basic and manual task of circuit assembly—situations in which circuits are still under construction and therefore not operational.

FORMATIVE STUDY

While prior work for assisting makers in creating electronic circuits is diverse, we argue there remains a lack of support for the fundamental task of physically placing components and wires on a breadboard, and inspecting the correctness of final assembled circuits. Indeed, the precise, fussy and confusing task of correctly placing wires and components in sockets to realize a circuit depicted in a schematic or pictorial circuit diagram remains much the same today as it was when breadboards were originally introduced 40 years ago [4]. To better understand the problems and issues users face during circuit breadboarding, and inform the design of potential solutions, we recruited 16 self-identified makers from the local student body to participate in a formative study. The study explored and contrasted performance in the tasks of both *assembling* and *inspecting* a circuit on a breadboard when users were provided with one of two distinct visual representations of a circuit: a standard electrical *schematic* or a *pictorial* diagram (as in Fritzing).

Thirteen participants were male and three female and they were aged between 20 and 32 (M=25, SD=3.83). Ten were graduate students and six undergraduates. They had various backgrounds including engineering, design, and Human-Computer Interaction. All had some experience in assembling circuits, yet none of them had received formal training in electrical engineering. Using a 7-point Likert scale, they self-rated their ability of reading and building circuit diagrams 5.6 (SD=.96).

To provide recent experiences for participants to reflect on in the study, we prepared breadboard circuit assembly and inspection tasks. To ensure these tasks were representative of those undertaken in the maker community, we examined the full set of example projects available on the Fritzing website[5]. We retrieved 5083 projects, removed duplicates (700) and corrupted (67) designs to yield a set of 4316 non-corrupted and unique projects. These projects featured a median of seven components connected by a median of 11 wires. We randomly selected four circuit diagrams from this corpus, each with 6 or 7 components connected by between 14 and 17 wires. We manually created equivalent schematic representations for each diagram.

There were two binary independent variables in the study: task type (assembly/inspection) and instruction format (schematic/pictorial). These were arranged in a repeated measures factorial design: all participants spent a maximum of ten minutes working on a circuit in each condition, ultimately experiencing all four circuits. Participants could indicate when they finished a circuit at any time with the ten minute time limit. Circuits were randomly assigned to specific conditions, while presentation order was balanced using a Latin square design. Instructional materials (circuit schematics or diagrams) for assembly tasks were unmodified from the originally selected examples, whereas for inspection tasks deliberate errors were introduced. Specifically, following the error classification in prior work [18], we introduced ten errors in total, two of each of the following five types: miss-wired components and wires as well as missing connections for components and wires, and misplaced components.

In the study, we recorded a video from both top-down (for a clear view of the breadboard) and in front of participants, conducted a 20-minute post-task semi-structured interview, and measured workload using the NASA TLX questionnaire [12]. Participants were compensated with ten USD in local currency. Additionally, as a performance incentive, participants received 5 USD for each pair of circuits that were error-free at the end of the study.

Results

Ten (15.62%) of the study tasks were not self-assessed as complete with the ten minute limit, with incomplete trials evenly split among both study variables (four with schematics, six with Fritzing and four during build and six during debug tasks). We recorded completion times for tasks using a stopwatch or extracted them from the video recordings in ambiguous cases. We measured errors by counting the number of misplaced components and wires for circuits in the assembly task, or the number of unidentified or misidentified mistakes in the inspection task. Two researchers independently counted errors, and, in case of numerical disagreement, discussed each case to reach an agreement. We analyzed all study data using two-way ANOVAs with two independent variables—reference diagram (*pictorial* vs. *schematic*) and task (*assembly* vs. *inspection*). Figure 2 summarizes the numerical results from the study. It shows nearly identical performance in terms of errors and time, while *schematics* resulted in modestly reduced overall workload. No significant main effects or interactions were found. This data suggests that, at least for the tasks we

[5] http://fritzing.org/media/fritzing-repo/projects (accessed March 2019)

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	Avg (SD)
Components translated	5	6	6	6	0	6	1	7	4	4	7	7	6	7	5	5	5.13 (2.06)
Components rotated	0	0	0	0	0	0	0	0	2	0	2	1	0	3	1	0	0.56 (0.96)
Mean Translations per Component	3.3	9.0	4.8	13.5	0	4.0	0.3	5.0	6.2	3.0	8.7	8.0	8.0	5.7	3.6	2.3	5.33 (3.5)
Wires added/removed	-2	0	1	-5	2	-1	0	1	-8	-6	-2	0	-1	-8	-3	-4	-2.25 (3.15)

Table 1. Summary of measured inconsistencies between the pictorial reference and actual circuit on a breadboard. Translations are measured as the offset of rows. For wires added/removed, a positive number indicates use of additional wires, while a negative number indicates use of fewer wires.

selected, neither the diagramming format used, nor the task undertaken, led to clear consistent variations in performance.

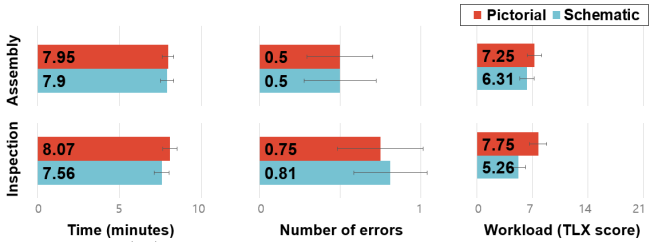

Figure 2. Time, errors and cognitive load for schematic and pictorial diagrams in assembly and inspection tasks in the formative study.

In terms of qualitative data, we collected 4 hours and 23 minutes of interviews. These were transcribed and analyzed by two researchers using open and axial coding methods. We examined this material to achieve a more nuanced understanding of participants' behavior and performance. Specifically, we explored how users experience working with both *schematic* and *pictorial* representations in order to learn about the challenges specific to each format and identify potential opportunities for design.

Makers who self-rated a low confidence with schematics (P2, P5, P6, P7, P11, P15, P16) reported difficulties in both interpreting the schematics and in translating the abstract circuit topologies into physical wiring on a breadboard. Some participants were concerned about how to place components with many pins or with polarity, since schematic symbols do not visually resemble component appearance (P11, P15, P16). Others were confused by the abstract representations of nets and debated on how to translate them physically onto the breadboard: "there were three junction dots where the lines are connected in the schematic diagram, which should be connected to the same row of the breadboard. However, [breadboard and schematic representations] do not correspond to each other, so it's hard to figure out how to handle it (P15)." Participants also reported challenges in creating appropriate circuit layouts. Specifically, as schematics provide no guidance on part placement, participants often found their initial choices caused knock on problems down the line. This ultimately required them to iterate on layouts, a process which typically involved laborious dis- and re-assembly of parts of their circuits mid-task: "I realized that it would have been better to place the button below the chip after I finished the wiring [...] it was annoying to move the components and rewire them (P14)."

Pictorial representations, on the other hand, were highly appreciated for their simplicity and directness, which allowed participants to "lay out components without thinking (P2)". This process, however, was not without hitches—ten participants, for example, noted their circuits "gradually diverged (P13)" from the pictorial reference. There were numerous reasons, from accidental mistranscriptions—"I tried to modify my breadboard circuit as closely as possible to the pictorial diagram, but I went astray in the middle (P12)"—through to intentional improvements—"the reference circuit used two ground rails which seemed unnecessary, so I removed a ground rail and changed wire placements (P15)." Regardless, as their own circuit diverged from the pictorial diagram, users reported the need "to convert [the] pictorial representation into a schematic and map it onto the breadboard again for reviewing (P3)." This process was considered arduous. Similar problems impacted the inspection task, where users had to look for miss-wiring or incorrect connections. P11 summed it up: "The breadboard circuit was not identical to the reference, so I needed to check the connections [...] I had to [mentally] convert the pictorial representation to a schematic diagram, and this was difficult [...] it was easier to use the schematic rather than pictorial representation to find the wrong connections."

These findings, highlighting the pros and cons of both schematic and pictorial diagram formats, shed light on the lack of significant differences in the quantitative data. They suggest that while makers found the simpler and more direct format of the pictorial diagrams appealing in terms of the clarity with which it physically depicted breadboard layouts, these properties also made understanding the underlying logical structure of the circuit more challenging that with schematics. This led to problems whenever there were layout discrepancies between the diagrams and breadboarded circuits. This was a relatively common occurrence—see Table 1 for a summary of how the final circuits differed from the pictorial references provided in the assembly task. Just one user created a circuit in which all of the components were in the specified breadboard locations, ten (62.5%) used fewer wires and three (18.75%) used more. These differences appear driven by a desire to either simplify what was viewed as excess wiring on the breadboard (e.g., by consolidating ground rails) or to space out components over the breadboard to simplify viewing. This latter process tended to increase the number of wires that were required.

In summary, our findings confirm prior work suggesting that makers find allocating positions and resolving problems with the components and wiring on a physical breadboard circuit challenging [4, 8] using either schematic and pictorial source materials. While both approaches present unique sets of problems, we identify increasing access to and facilitating use of schematics for makers as a key opportunity for design. Specifically, while participants valued the logical, abstract representation of circuit schematics for problem solving, they felt a need

for improved support for basic layout tasks such as arranging or orientating components on a breadboard. In addition, our observations about pictorial circuit diagrams highlight some of key qualities to be emphasized in any new prototype in this space. Specifically, while direct pictorial "put-that-there" instructions were appreciated, they also did not scale to the incremental changes that creep into a design. Solutions in this space need maintain accurate, high quality guidance as users intentionally tweak and adjust designs and also reduce the chance for inadvertent component placements that lead to accidental divergences.

The formative study results suggest a need for tools that can combine the advantageous qualities of both abstract and pictorial representations of circuits. We see one way of achieving this as by designing systems that emphasize the links between abstract schematic views and concrete layouts on a breadboard. Within this space, this paper proposes a solution based on interactive in-situ visual cues that augment the topological information in the schematics with direct unambiguous guidance about where components should be located on a physical breadboard.

SCHEMABOARD SYSTEM OVERVIEW

SchemaBoard (Figure 1) aims to support users in the transition between an on-screen schematic representation of a circuit and its corresponding physical instantiation on a breadboard. The main objective is to help users correctly place components on a breadboard (*assembly* task), and supporting quick inspections of whether the layout is congruent with the schematic diagram it is based upon (*inspection* task). SchemaBoard achieves this by maintaining a *dual representation* of circuit diagrams (a computer based schematic view linked to a physical breadboard layout), and by externalizing this mapping via in-situ visualization using an LED matrix underneath the breadboard.

SchemaBoard works as follows. First, it internally computes the location of the components in any schematic diagram, and automatically maps them to physical locations (i.e., rows) on a breadboard. SchemaBoard offers an interactive highlighting feature to visualize each computed component location directly on its coupled physical breadboard. When the user touches an on-screen element of the schematics (i.e., a component or a pin), SchemaBoard lights up the corresponding rows of the breadboard (Figure 3). SchemaBoard supports highlighting component locations, including blinking to indicate the first pin (the standard reference point) for ICs and the negative pin for polarized components such as capacitors, diodes, or LEDs. SchemaBoard also uses highlighting to show all the rows connected to selected electrical nets, giving an overview of connectivity even on crowded breadboards. With this feature, users can quickly identify the location and orientation of components and wires before placing them, or readily verify the correctness of component placements when inspecting assembled circuits.

SchemaBoard operates in two modes—a *manual mode* that allows users to explicitly select components and nets, and a *guide mode* that provides step-by-step circuit assembly for all the elements of a circuit. Users can switch between the two modes at any time using a toggle button. In the *manual mode*, users are free to inspect the location of components and nets, using single or multiple selections (Figure 4). Users can also manually change the mapping between the schematic and the component location on the breadboard, by dragging the component's graphical representation to a new location on a virtual breadboard displayed in software—a feature that mimics the component placement technique used in prior work [15].

In the *guide mode*, SchemaBoard provides step-by-step circuit assembly instructions (Figure 5). Here, rather than the user determining the best order for component placement, they can simply follow the system directions. SchemaBoard indicates on-screen which component to place by name and by highlighting it in the schematics, and illuminates the rows on the breadboard where the user should insert the component or wire. Each jumper wire position is indicated by illuminating two rows of SchemaBoard simultaneously. As with interactive highlighting, it flashes a breadboard row to indicate polarity or pin 1 for ICs. Guides are structured such that all components are placed first, followed by connecting all wires. An optional final stage checks whether all components and wires are correctly connected. In this stage, the software highlights all the *electrical nets*—all the rows in the circuit that are connected to the same circuit node—allowing users to quickly verify missing or misplaced circuit elements.

In summary, given a schematic, SchemaBoard computes valid locations of components on its breadboard and visualizes them in-situ using an underlying LED matrix. It also maintains an internal record of the topology of the circuit, so that the user

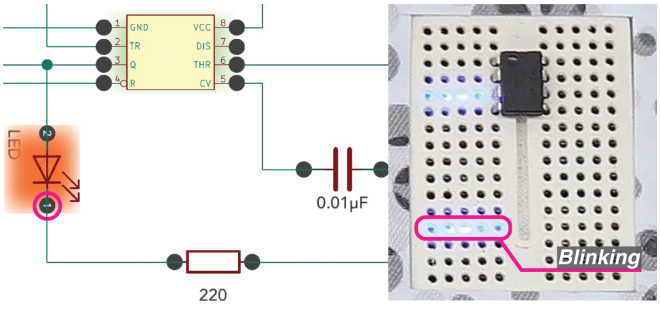

Figure 3. Schemaboard circuit diagram showing a selected component highlighted in orange in the schematic view (left). SchemaBoard breadboard highlighting the rows where the component should be placed. A blinking row indicates where to position pin 1 (right).

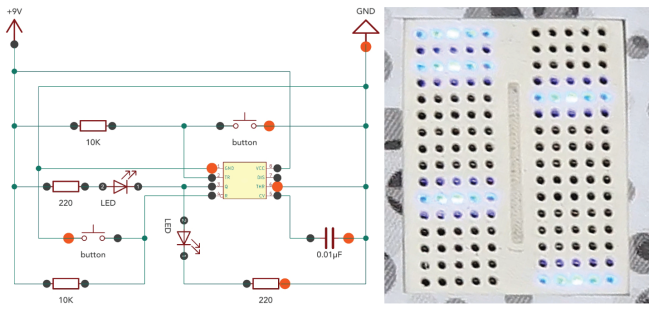

Figure 4. Schemaboard software showing highlighting of the ground net via orange colored pins in the schematic view (left). Multiple corresponding rows are visually highlighted on the breadboard (right).

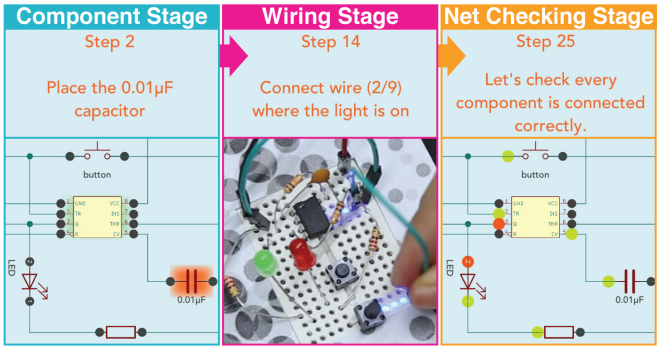

Figure 5. SchemaBoard Guide Mode showing instructions for placement of components (left), wires (center), and optional check of all the nets (right). Orange highlights show the current component or wire while green highlights show those for which instructions have already been provided.

can freely modify it, or can highlight specific nets to visualize planned pin connections. We note that SchemaBoard does not sense components' locations [30] and therefore cannot prevent users from incorrect placements or detect erroneous circuits [9, 29]. In contrast, it is instead designed to support users in accurately mapping schematic diagrams onto physical circuits by providing an initial valid reference layout, flexibly accommodating users' adjustments to that design and providing in-situ feedback on the breadboard to help ensure that the circuits that are created correctly instantiate the design in the schematic and are free of errors.

IMPLEMENTATION

Hardware

The SchemaBoard hardware (Figure 6) takes the form of a standard 45x35 mm mini breadboard with two symmetrical sets of 16-terminal strips (32 in total). Each strip has five interconnected 2.54 mm-spaced sockets and sits over a uniquely

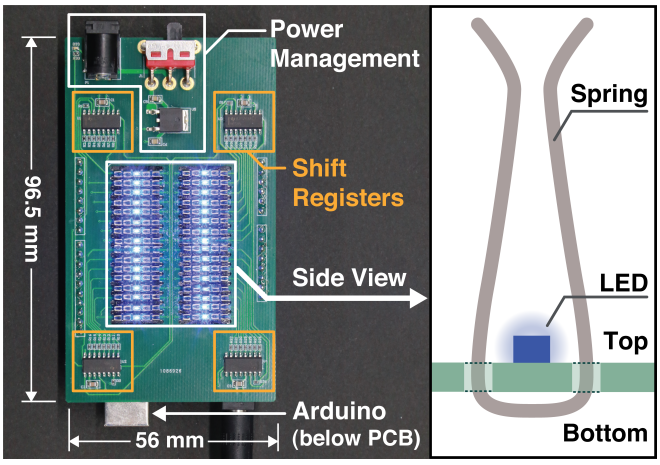

Figure 6. SchemaBoard hardware. The part of the PCB designed to fit underneath a breadboard is structured as two columns of bars. Each bar features a single LED mounted at its center and is bordered top and bottom by a small gap. Each bar fits within a row of five standard breadboard spring clips without disrupting wire placement (see right of figure). The LEDs are powered and controlled via connections at the left and right of the bars.

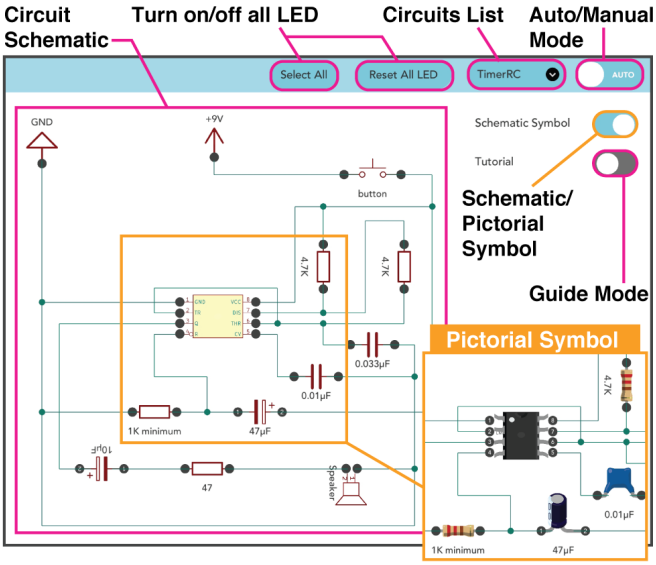

Figure 7. SchemaBoard software UI.

addressable blue LED —one LED per breadboard row. The LEDs and breadboard are part of a custom PCB in the form of an Arduino UNO shield. The connected Arduino can instruct one or more of the LEDs to switch on, switch off, or blink periodically. This is achieved through four cascading 8-bit shift registers (SN74HC595DR), each responsible for controlling eight LEDs (for a total of 32 LEDs, as shown in Figure 6). In order to make the LED light visible to users while maintaining breadboard functionality, we used surface-mount LEDs soldered on the top layer of a perforated circuit board. Then the strips, connecting the rows, are mounted so as to enclose the LEDs (see Figure 6-right). The light is scattered by the strip resulting in the entire row being lit. The system is powered by a 12V external DC power adapter that is regulated to 5V on the board for compatibility with the Arduino. The Arduino is controlled via a serial connection to the SchemaBoard software running on a PC.

Software

The SchemaBoard software consists of three separate modules that: control the breadboard LEDs; create and maintain an internal representation of the circuit diagram and its topology and; display the SchemaBoard UI. The breadboard control module runs on SchemaBoard's Arduino and simply controls the LEDs in response to commands issued from a connected PC over a serial link. The PC runs software, in the form of a NodeJS[6] server, that issues commands to the hardware and can load a common format for schematic diagrams (KiCad[7] *.sch* files) and process these with a solver that produces valid physical breadboard layouts. This solver operates on the schematic's netlist, which is comprised of lists of interconnected component pins. First, it assigns the two top-left rows of the breadboard to ground and the two top-right to power. Next, it places components across the breadboard (e.g., dual in-line packages such as ICs). The solver then determines

[6] https://nodejs.org
[7] http://kicad-pcb.org/

the number of rows needed for each net; if more than four components need to be connected together, it uses one pin to bridge between two adjacent rows (because SchemaBoard uses a standard five column breadboard format). Finally, the script lays out the nets uniformly, trying to keep components close to each other and minimizing the number of jumper wires for connections. This whole optimization process needs to run only once per schematic, and its results are stored as files in JSON format. Finally, the PC software can accept wireless connections from the SchemaBoard UI software in order to transmit data abut the loaded schematic file and calculated breadboard layout as well as to propagate commands to control the breadboard highlighting. Currently, SchemaBoard supports 20 common schematic symbols including three abstract or generic components (8 pin chip, 16 pin chip, 2 pin component) that can be used to describe the connections of a wide range of different devices. The set of supported symbols/components was selected from an analysis of a database of circuits disseminated in the maker community. It featured the following symbols: battery; vcc; ground; 8 pin chip; 16 pin chip; 2 pin component; unpolarized capacitor; polarized capacitor; resistor; photo-resistor; inductor; diode; Zener diode; transistor; 6 pin relay; op-amp; 555 timer; LED; speaker and; switch.

The SchemaBoard UI software runs on a tablet (Samsung Galaxy Tab S5e) and was written using Unity 3D in C#. It is responsible for the overall system logic, calculating the states of the LEDs, handling user input, and graphically visualizing schematics— Figure 7 shows this interface. The tablet communicates via a HTTP wireless connection to the NodeJS[8] server on the PC. At startup, users load a schematic. Once selected, this is sent to the solver on the PC (via http), and the resulting JSON file of nets and connections is returned. Using toggle buttons the user can select the mode of operation (manual or guide) and how the components should be visualized (using traditional schematic symbols or pictorial icons). This dual visualization was motivated by our study findings, as some users remarked they were unable to clearly identify components using their schematic symbols. In the UI, users can tap any component or connection. This is then highlighted on the screen and, via commands propagated to the Arduino by the PC server, also on the physical breadboard, where all connected rows light up. On-screen color overlays also reveal which components are currently selected, and, in guide mode, the components for which instructions have already been provided.

SchemaBoard's hardware and software are open-source and full details, including research related materials, can be found online at https://github.com/makinteractlab/SchemaBoard.

EVALUATION STUDY

We conducted a two by two mixed-design study with makers to compare the tasks of *assembly* and *inspection* of circuits (within subjects) using either *SchemaBoard* or a *baseline* involving traditional use of schematic diagrams (between groups). We recruited sixteen makers, thirteen male and three female, aged 19-30 (M=24.4, SD=3.3). Ten were graduate students and the remaining six undergraduates. They had various education backgrounds including engineering (biology, computer science, material science, mechanical and electronic engineering), physics, and industrial design. None was previously involved in the formative study. Participants were randomly assigned to either SchemaBoard or baseline conditions. To check for population biases between the two groups, we collected an initial self-efficacy score [7] that represents individual confidence in assembling circuits using schematic diagrams. Participants in the SchemaBoard group rated themselves 71.8/100 (SD=11.7) and in the baseline group 75/100 (SD=9.2). An independent-samples t-test revealed no differences between these groups, suggesting that skill levels among the two groups were relatively homogeneous.

The experimental procedure closely mirrored the formative study. Each participant completed two tasks: *assembly* of a circuit from scratch, and *inspection* of a provided circuit with standard errors introduced as bugs. In both the baseline and SchemaBoard conditions, we provided schematics as a reference. Before starting the experiment, we introduced the SchemaBoard group to the system via a five-minute tutorial session with a simple example circuit. When using SchemaBoard in the study tasks, participants could use any of its features including both the guide (step-by-step) and manual modes (manual selections). The system started up in guide mode. For the tasks, we selected two representative circuits from the Encyclopedia of Electronic Components [13]. Both circuits had 10 components (e.g., IC, LED, switches, capacitors, resistors, speaker) and required 4 to 7 wires. For the *inspection* tasks, we introduced ten standardized errors in each circuit using the same convention adopted in the formative study. We fully balanced assignment of circuits to tasks.

After a brief introduction, each participant performed the *assembly* and *inspection* tasks for up to 20 minutes followed by a NASA TLX [12] questionnaire to measure workload and again a self-efficacy questionnaire to assess participants' confidence in their performance. The longer duration of the tasks, compared to that in the formative study, is due to the use of more complex circuits (42% more components). Task order was fully balanced to counter practice or fatigue effects. After both tasks, we performed a 20-minute semi-structured interview. The experiment took approximately one hour and was recorded on video. Participants were compensated with 10 USD in local currency and additionally received 5 USD if both circuits were completed correctly. This procedure resulted in the following measures: task completion rate (within the allotted 20 minute time limit); task completion time (for completed circuits); number of errors in final circuit (for completed circuits) and; TLX workload and self-efficacy scores. We note that task completion rate was self determined—participants finished a trial when they believed they had fully and correctly completed the given study task.

Quantitative Results

We first examined completion rates using two Fisher's Exact Tests (FET) [11], one on each of the study variables. FET was selected due to its suitability for contingency testing on small samples. This procedure revealed that completion rates

[8] https://nodejs.org

were significantly higher in both the assembly task and for the SchemaBoard group (both $p = 0.0068$). Examination of the raw data revealed these differences were due to the fact that seven study tasks (21.8%) were not completed and that these failures all occurred in the baseline inspection condition. This strongly suggests that the inspection task was extremely difficult for makers using standard schematics—by their own self-assessment, most participants (87.5%) could not complete study tasks within 20 minutes. The lack of similar patterns of failure in the SchemaBoard group provides clear evidence for the benefits it can provide—rather than just offer improvements to efficiency or accuracy, SchemaBoard seems to have enabled makers to perform a technical task that was otherwise entirely beyond their abilities. This represents a very strong endorsement of the system.

The fact that only a single participant completed the inspection task in the baseline condition led to a lack of equivalent time and error data for statistical comparison against the SchemaBoard condition. Instead we simply report these data for baseline and SchemaBoard and restrict formal analysis of these metrics to data from the assembly task alone. Figure 8 shows completion time and error data from all tasks that were self-assessed as complete in the study. For the seven participants who did not indicate they had completed the inspection task within the time allocated (all in the baseline condition), we note the mean number of errors was very high in comparison to all other conditions in the study (M: 5, SD: 3.16)—this suggests a failure to complete the task is simply a knock on effect of a failure to isolate and resolve errors in the circuit. This data provides further evidence that makers find inspecting and debugging circuits with standard schematics extremely challenging.

In terms of the assembly task alone, Shapiro-Wilk tests showed time data was normally distributed and a independent samples t-test revealed that SchemaBoard led to significantly faster performance than baseline ($p < 0.001$). On the other hand, error data was not normally distributed and a Wilcoxon signed-rank test indicated there were no differences in the error counts between conditions ($p = 0.11$). Similarly, a FET on the proportion of participants whose circuits were genuinely correct or not (SchemaBoard: 6 from 8, baseline: 3 from 8), was not significant (p=0.31). This indicates that, for the assembly task, SchemaBoard substantially sped up performance (by approximately 40%, or nearly 6 minutes), but did not lead to significantly more accurate final circuits. Given the small number of participants in the study, we suggest that the lack of significance in the error data may be a type II error. An alternative explanation may be that the relatively low number of errors during assembly tasks —an overall median of zero errors (mean: 0.81, SD: 1.51) per circuit— simply indicates correctness was not a major challenge for most participants during the assembly tasks.

Finally, we analyzed TLX overall workload and self-efficacy scores. The data are presented in Figure 8. Shapiro-Wilk tests indicated all data was normally distributed so we used mixed method two by two ANOVAs. All main effects as well as the interaction in the TLX data were significant. This shows that SchemaBoard led to significantly reduced overall workload ($F(1,14) = 20.34$, $p < .001$, $\hat{\eta}^2_G = .542$) and higher ratings of efficacy ($F(1,14) = 5.27$, $p = .038$, $\hat{\eta}^2_G = .251$). In addition, the results highlight the increased challenge of the inspection task—compared to assembly, it led to greater workload ($F(1,14) = 29.24$, $p < .001$, $\hat{\eta}^2_G = .280$) and reduced efficacy ($F(1,14) = 9.28$, $p = .009$, $\hat{\eta}^2_G = .068$). We note that the significant interaction effect in workload ($F(1,14) = 23.82$, $p < .001$, $\hat{\eta}^2_G = .241$) is due to the fact that SchemaBoard shows only modest increases (5.9%) in the inspection task, while baseline shows very substantial increases (44.2%). This reinforces the data from the task completion rate. It suggests that one of the key benefits of SchemaBoard is that it greatly simplifies the challenging task of circuit inspection and debug—facilitating completion by lowering the difficulty of these complex and important activities.

Qualitative Results from Interviews

The 3h 25m of interviews were conducted in the local language and transcribed and analyzed by two researchers using open and axial coding methods.

In general, all makers in the SchemaBoard group understood how to use the system and commented positively on its overall functionality. P2 stated: "I could easily use [...] SchemaBoard for assembl[ing] circuits intuitively." P6 reported that the dual representation of nets on the screen and through the LEDs on the breadboard was helpful in both understanding the schematic and checking the circuit. Furthermore, all makers appreciated the guide mode because they felt it enabled them to assemble the circuit and remain confident it was without missing parts (P8). P7 summed it up as they "trusted the auto-routed circuit from SchemaBoard".

Supporting different levels of expertise. Multiple participants commented that SchemaBoard could support a range of makers, from "elementary school students" (P2) to a more general set of "those who are new to circuits, to professionals who make complex circuits" (P4). P8 noted SchemaBoard could aid both novices and experts, saying: "Using this, even beginners will be able to create and test circuits, and experts can reduce mistakes and debugging time." Reflecting on their previous experience, a novice (P10, self-efficacy of 6.8/10 in the pre-study assessment) and expert (P12, self-efficacy of 9.7/10) users describe what they mostly appreciate of SchemaBoard.

P10 (novice): *"I often plugged components in the wrong place because of the dense pinholes in the breadboard, however, this would be preventable because the SchemaBoard lit where to place components."*

P12 (expert): *"It's great to be able to use new parts or chips that you haven't experienced without looking at the datasheets. The system already has the information of how to connect the pins."*

Increasing users' confidence. Overall, all participants appreciated how SchemaBoard could fit the working style of makers with different expertise levels. A possible explanation for this universal support is that all participants in the SchemaBoard group reported a quantifiably increased level of confidence

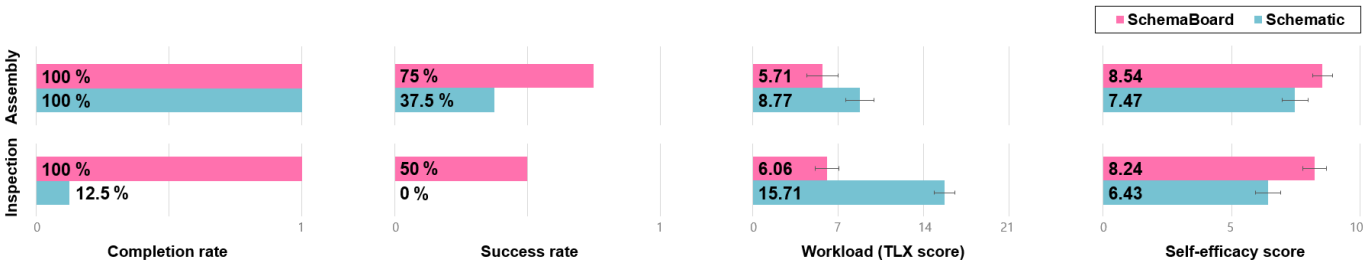

Figure 8. SchemaBoard evaluation study results: self-assessed completion rate, independently assessed success rate, workload and self-efficacy. Data are from all trials.

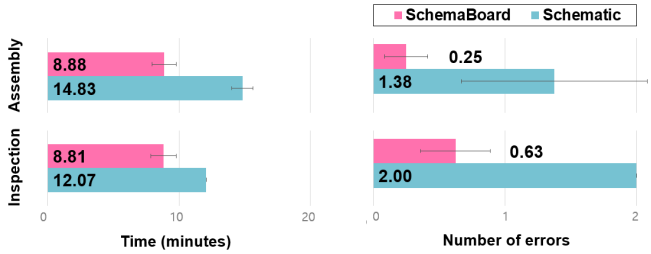

Figure 9. SchemaBoard evaluation study time and error results. Data are derived only from trials that were self-assessed as completed.

in building circuits and in trusting the correctness of the outcomes. Participants reported that SchemaBoard "reduced the concerns [...] about mistakes like polarity and miss-wiring" (P14) and gave them confidence that the circuit was complete and inspected: "In previous experience with standard schematics, I [...] often forgot what component had been checked and what should be next. However, using SchemaBoard, I was convinced that there was no missing part to check because I completed all the steps (P8)."

DISCUSSION AND LIMITATIONS

The difficulties that makers experience during assembly of circuits is a well-studied problem [4, 8]. Past work has sought to tackle this by preventing design errors in the first place [1, 8, 18, 25] or by assisting makers in debugging incorrect circuits [9, 19, 29]. To complement this work, we examine the fussy, tedious, error-prone and ubiquitous task of physically constructing circuits on a breadboard by plugging in components and wires. Our formative study documents key problems with both the schematic and pictorial circuit diagrams used as source material in this task. Specifically, makers find schematics hard to accurately translate into valid circuit layouts, while pictorial circuit diagrams are inflexible and makers report frequently, and confusingly, diverging from the depicted arrangements.

Based on these outcomes, we designed SchemaBoard, a system that facilitates rapid and accurate construction and examination of breadboard layouts while maintaining support for intentional user customization. SchemaBoard achieves this by generating and maintaining editable circuit layouts from schematics and using lighting patterns shown directly on a custom LED back-lit breadboard to visualize correct component placements (or specific nets) in response to direct input by users. This makes the mapping between the abstract schematic source material and the physical breadboard layout precise, clear and explicit. Furthermore, it is compatible with both user- and system-driven (guide mode) placement of components. Our user evaluation provides evidence that SchemaBoard achieves its goals. Both novice and expert users appreciated SchemaBoard's simplicity, and valued completing significantly more circuits in less time and while experiencing reduced workload and increased self confidence.

Despite these positives, some participants were concerned that, due to the high level of support Schemaboard provides, they might successfully use schematics to construct circuits without ever understanding them. While similar comments could be levied at almost any tool, these worries do highlight the multi-faceted nature of the aid provided by Schemaboard: it both lays out a circuit and then provides detailed instructions for how to assemble it accurately. While this is doubtless highly efficient, a factor strongly appreciated by participants, it is also highly automated and leaves little scope for the types of deep engagement in a task that might best support learning. To make Schemaboard more suitable for learning environments, we suggest that its different features could be selectively enabled—for example, novices seeking to improve their skills in reading a schematic and laying out a circuit could manually perform this task and use Schemaboard to provide feedback only during assembly. This would help ensure the constructed circuit accurately follows their design and reduce cases where layout design errors (the learner's current focus) are confusingly compounded with assembly errors. Equally, Schemaboard's layout engine could be used without assembly feedback to help a learner improve their skills in schematic circuit design and bespoke breadboard construction without worrying that the specific layout they are building contains errors. In the future, we see value in studying the different features of Schemaboard separately in order to clearly determine both their individual benefits (e.g., in comparison to other techniques for marking breadboard rows such as simple print labels) and also how they can be used to best scaffold learning. While using the full set of features, as in the current study, likely maximizes efficiency (for all) and accessibility (for novices), Schemaboard may also be suitable for a wider range of tasks and users if appropriately customized.

Furthermore, despite its utility, the current prototype also has clear opportunities for technical improvement. Future iterations of the system will use full-size breadboards and RGB LEDs with multiple blinking patterns to convey more information about specific nets or components. It would be also

possible to place an LED at each hole of the breadboard to indicate specific positions within the rows for more detailed instructions. In terms of software, we aim to support schematics with multiple pages and from different software suites beside Kicad. Future work will also increase the size of supporting electronic components. Finally, we also seek opportunities to improve the automatic layout of components on breadboard, and development of interaction techniques that can enhance support for the user's manual placement of physical components in arbitrary locations. These enhancements will expand the scope of the current SchemaBoard system and provide support for makers working on the broadest possible set of projects.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2020R1A2C1012233).

REFERENCES

[1] 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. DOI: http://dx.doi.org/10.1145/3126594.3126637

[2] Ayah Bdeir. 2009. Electronics As Material: LittleBits. In *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction (TEI '09)*. ACM, New York, NY, USA, 397–400. DOI: http://dx.doi.org/10.1145/1517664.1517743

[3] Andrea Bellucci, Alberto Ruiz, Paloma Díaz, and Igancio Aedo. 2018. Investigating Augmented Reality Support for Novice Users in Circuit Prototyping. In *Proceedings of the 2018 International Conference on Advanced Visual Interfaces (AVI '18)*. Association for Computing Machinery, New York, NY, USA, Article Article 35, 5 pages. DOI: http://dx.doi.org/10.1145/3206505.3206508

[4] 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*. ACM, New York, NY, USA, 3485–3497.

[5] Thomas Brühlmann. 2019. *Arduino: Praxiseinstieg*. MITP-Verlags GmbH & Co. KG.

[6] Alvaro Cassinelli and Daniel Saakes. 2017. Data Flow, Spatial Physical Computing. In *Proceedings of the Eleventh International Conference on Tangible, Embedded, and Embodied Interaction*. ACM, New York, NY, USA, 253–259.

[7] Deborah R Compeau and Christopher A Higgins. 1995. Computer self-efficacy: Development of a measure and initial test. *MIS quarterly* (1995), 189–211.

[8] Josh Urban Davis, Jun Gong, Yunxin Sun, Parmit Chilana, and Xing-Dong Yang. 2019. CircuitStyle: A System for Peripherally Reinforcing Best Practices in Hardware Computing. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST '19)*. Association for Computing Machinery, New York, NY, USA, 109–120. DOI: http://dx.doi.org/10.1145/3332165.3347920

[9] 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. DOI: http://dx.doi.org/10.1145/2984511.2984566

[10] Robert Faludi. 2010. *Building wireless sensor networks: with ZigBee, XBee, arduino, and processing*. " O'Reilly Media, Inc.".

[11] R. A. Fisher. 1922. On the Interpretation of 2 from Contingency Tables, and the Calculation of P. *Journal of the Royal Statistical Society* 85, 1 (1922), 87–94. http://www.jstor.org/stable/2340521

[12] Sandra G Hart. 1986. NASA Task load Index (TLX). Volume 1.0; Paper and pencil package. (1986).

[13] Fredrik Jansson and Charles Platt. 2014. *Encyclopedia of Electronic Components; Volume 2: LEDs, LCDs, Audio, Thyristors, Digital Logic, and Amplification.* Maker Media.

[14] Yoonji Kim, Youngkyung Choi, Daye Kang, Minkyeong Lee, Tek-Jin Nam, and Andrea Bianchi. 2019a. HeyTeddy: Conversational Test-Driven Development for Physical Computing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 4, Article Article 139 (Dec. 2019), 21 pages. DOI: http://dx.doi.org/10.1145/3369838

[15] Yoonji Kim, Youngkyung Choi, Hyein Lee, Geehyuk Lee, and Andrea Bianchi. 2019b. VirtualComponent: A Mixed-Reality Tool for Designing and Tuning Breadboarded Circuits. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 177, 13 pages. DOI:http://dx.doi.org/10.1145/3290605.3300407

[16] 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. DOI: http://dx.doi.org/10.1145/1517664.1517735

[17] 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*. ACM, New York, NY, USA, 283.

[18] Jo-Yu Lo, Da-Yuan Huang, Tzu-Sheng Kuo, Chen-Kuo Sun, Jun Gong, Teddy Seyed, Xing-Dong Yang, and Bing-Yu Chen. 2019. AutoFritz: Autocomplete for Prototyping Virtual Breadboard Circuits. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, USA, 403.

[19] 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*. ACM, New York, NY, USA, 299–310.

[20] Michael McRoberts. 2013. *Beginning Arduino*. Apress.

[21] 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*. ACM, New York, NY, USA, 1270–1281.

[22] Yoichi Ochiai. 2010. The Visible Electricity Device: Visible Breadboard. In *ACM SIGGRAPH 2010 Posters (SIGGRAPH '10)*. ACM, New York, NY, USA, Article 98, 1 pages. DOI: http://dx.doi.org/10.1145/1836845.1836950

[23] Joel Sadler, Kevin Durfee, Lauren Shluzas, and Paulo Blikstein. 2015. Bloctopus: A novice modular sensor system for playful prototyping. In *Proceedings of the ninth international conference on tangible, embedded, and embodied interaction*. ACM, New York, NY, USA, 347–354.

[24] Canadian Standard. 1975. Graphic Symbols for Electrical and Electronics Diagrams. (1975).

[25] Evan Strasnick, Maneesh Agrawala, and Sean Follmer. 2017. Scanalog: Interactive design and debugging of analog circuits with programmable hardware. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM, New York, NY, USA, 321–330.

[26] 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. DOI: http://dx.doi.org/10.1145/2984511.2984527

[27] Jeremy Warner, Ben Lafreniere, George Fitzmaurice, and Tovi Grossman. 2018. ElectroTutor: Test-Driven Physical Computing Tutorials. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology (UIST '18)*. ACM, New York, NY, USA, 435–446. DOI: http://dx.doi.org/10.1145/3242587.3242591

[28] Thomas G Wilson. 1988. Life after the schematic: The impact of circuit operation on the physical realization of electronic power supplies. *Proc. IEEE* 76, 4 (1988), 325–334.

[29] 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. 2017a. 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. DOI: http://dx.doi.org/10.1145/3126594.3126646

[30] 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. 2017b. 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. DOI: http://dx.doi.org/10.1145/3126594.3126634