VirtualWire: Supporting Rapid Prototyping with Instant Reconfigurations of Wires in Breadboarded Circuits

Woojin Lee
Industrial Design, KAIST
Daejeon, Republic of Korea
aerolane0302@gmail.com

Ramkrishna Prasad
Mechanical Engineering, KAIST
Daejeon, Republic of Korea
rprasad@kaist.ac.kr

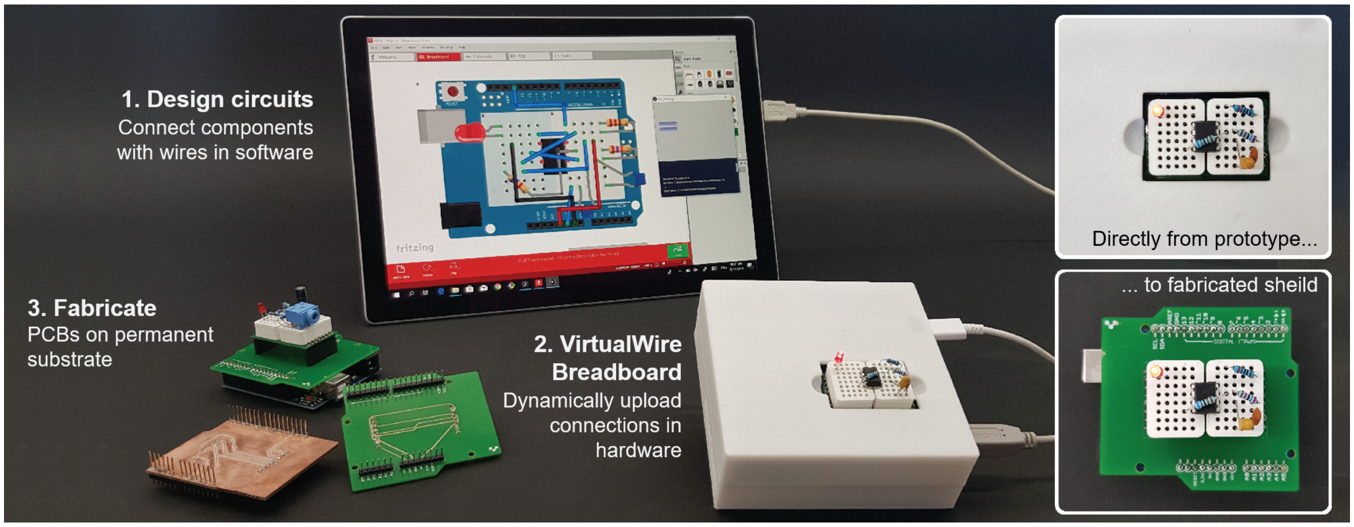
Seungwoo Je
Industrial Design, KAIST
Daejeon, Republic of Korea
yanga_seungwoo@kaist.ac.kr

Yoonji Kim
Industrial Design, KAIST
Daejeon, Republic of Korea
yoonji@kaist.ac.kr

Ian Oakley
Human Factors Engineering, UNIST
Ulsan, Republic of Korea
ian.r.oakley@gmail.com

Daniel Ashbrook
Human Centred Computing, University of Copenhagen
Copenhagen, Denmark
dan@di.ku.dk

Andrea Bianchi
Industrial Design, KAIST
Daejeon, Korea
andrea@kaist.ac.kr



1. Design circuits
Connect components with wires in software
2. VirtualWire Breadboard
Dynamically upload connections in hardware
3. Fabricate
PCBs on permanent substrate
Directly from prototype...
... to fabricated sheild
fritzing

Figure 1: VirtualWire system overview. Users create circuits in software tools such as Fritzing (1), place real components on a breadboard with the circuit connections automatically and dynamically realized from the software (2) and, fabricate permanent versions of their circuit (e.g., PCBs) when completed (3). On the right, the breadboard can be directly transplanted to the PCB for immediate usage.

ABSTRACT

Assembling circuits is a challenging and time consuming activity for novice makers, frequently resulting in incorrect placements of wires and components into breadboards. This results in errors that are difficult to identify and debug, and delays that hinder creating, exploring or reconfiguring circuit layouts. This paper presents VirtualWire, a tool that allows users to rapidly design and modify circuits in software and have these changes instantiated in real-time as electrical connections on a physical breadboard. To achieve this, VirtualWire dynamically translates circuit design files into physical connections inside a hardware switching matrix, which handles wiring across breadboard rows and to/from an embedded Arduino. The user can interactively test, tune, and share different circuit layouts for an Arduino shield, and once satisfied, can fabricate the circuit on a permanent substrate. Quantitative and qualitative user studies demonstrate that VirtualWire significantly reduces the time taken for (by 37%), and the number of errors made during (by 53%) circuit assembly, while also supporting users in creating readable, space-efficient and flexible layouts.

TEI '21, February 14–17, 2021, Salzburg, Austria
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8213-7/21/02…$15.00
https://doi.org/10.1145/3430524.3440623

CCS CONCEPTS

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

KEYWORDS

Physical computing; circuits; virtual wires; system.

ACM Reference Format:

Woojin Lee, Ramkrishna Prasad, Seungwoo Je, Yoonji Kim, Ian Oakley, Daniel Ashbrook, and Andrea Bianchi. 2021. VirtualWire: Supporting Rapid Prototyping with Instant Reconfigurations of Wires in Breadboarded Circuits. In *Fifthteenth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '21), February 14–17, 2021, Salzburg, Austria.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3430524.3440623

1 INTRODUCTION

Circuit design and prototyping activities are being practiced by increasingly diverse non-expert communities interested in building tangible interfaces and artifacts [17, 18]. Enabled by the low barriers to entry and encouraged by the popularity of electronic prototyping platforms such as Arduino[1], circuits are now commonly created and built by a broad spectrum of non-technical makers and of academic researchers, such those in the TEI community. For these users, assembling circuits, most typically by placing wires and components into breadboards to instantiate designs depicted in learning materials, remains a challenging and time-consuming activity [33, 34]. The key problems of *breadboarding* are both simple and fundamental: components are misplaced and wires are omitted, connected incorrectly, or inadvertently disconnected. Both novice and intermediate users are reported to find these errors hard to diagnose and debug [5, 34]. These issues impede the ability of novices to explore and tinker with breadboarded circuits and contribute to a sense that the technology is intimidating and confusing [2].

To minimize issues associated with tangled wires and fragile connections, researchers have proposed techniques to simplify breadboarding, emphasizing qualities such as the speed and ease with which circuits can be built [6, 19], supporting more complex designs (e.g., multi-layered) [33], and the use of novel substrates and materials such as paper [25], stickers [10], and printable ink [12]. While these systems typically result in reliable circuit layouts, they lack the exploratory and iterative qualities of breadboarding: once circuits are designed in software, they are rendered onto a fixed substrate and components are soldered [6, 31] or glued [10, 19, 25] to their surface. Modifications to the design require redoing the entire process. CircuitStack [33] is a notable exception: it places the circuit topology on swappable sheets printed in conductive-ink clamped under a breadboard holding circuit components. While this approach combines reliable circuit layouts with swappable components, making changes to the circuit topology remains a multi-step and laborious process. It requires redesign on a PC, reprinting new circuit sheets and, dis- and re-assembly of the breadboard and clamped circuits.

To better support novices in the breadboarding process, we present *VirtualWire* (Figure 1), a technique that borrows from prior research on virtually simulating electronic components with hardware [14, 28, 35], but instead of focusing on the behavior of specific components in the circuit, it enables to reconfigure the hardware connections (e.g., wires) between arbitrary physical components using software. Although circuit hardware reconfigurations on chip are possible with technologies that enable gate-level programmability, such as system-on-chip (e.g., Cypress[2]) and field-programmable gate arrays (FPGAs), these systems require understanding of digital combinatorial logic, and the usage of highly specialized tools and hardware description languages (e.g, Verilog or VHDL) that are very complex and unsuitable for novices. On the contrary, with VirtualWire, users simply create and visually modify circuit layouts in an electronic design software by directly drawing wires on a virtual breadboard or an electronic schematic. VirtualWire then physically instantiates the electrical connections using a programmable analog switching matrix that supports arbitrary configurations across rows of the breadboard and to/from an embedded Arduino. Differently from closely related work [14, 30], the user can therefore connect components on a breadbaord with unlimited number of *"virtual wires"* and in any desired arrangement. Finally, once the user is satisfied with the circuit, the system can also generate a PCB layout for fabricating the circuit on a permanent substrate.

This paper presents the design of VirtualWire and demonstrates its utility via a series of example applications in the form of Arduino shields. We then validate VirtualWire's usefulness with quantitative and qualitative user studies that show it is significantly faster than traditional breadboarding while also reducing the number of mistakes users make and supporting circuit layouts that are perceived to be simple to parse and read, spatially efficient, and easy to update and modify.

2 RELATED WORK

2.1 Circuit prototyping and fabrication

Solderless breadboards have been the standard method of prototyping circuits since their introduction in the 1970s [24]. While their simplicity allows users to create and reconfigure a circuit by simply plugging components and jumper wires into breadboard sockets, they struggle to scale. With complex circuits containing numerous jumper wires, the benefits of breadboards can be overpowered by their limitations. Quickly finding connection points for wires becomes more difficult and re-positioning a component or re-configuring a circuit can require pulling out most of wires and components and starting over. Prior research has reported that beginners perceive circuits on breadboards to be delicate [33], and that miswiring issues are difficult to identify and debug [5].

Complexity and accuracy in circuit construction can be achieved using Printed Circuit Boards (PCBs), which mechanically support and electrically connect electronic components using conductive tracks on one or more layers. PCBs are very reliable, as they form permanent point-to-point connections between components without wires. However, they are most appropriate for completed designs, as they cannot be easily modified, reconfigured, or debugged after production. Some research seeks to increase their flexibility—Pinpoint [29], for example, simplifies PCB debugging via an automatic pipeline that supports actions such as probing signals, disconnecting components and isolation testing. While this prior work

[1]https://www.arduino.cc

[2]https://www.cypress.com

mainly deals with post-design activities that take place when a circuit layout has been fabricated and is going through processes of testing and revision, the current paper focuses instead on the early stages of prototyping.

Many researchers have also sought to close the gap between circuit prototyping and fabrication, proposing techniques that combine aspects of the flexibility of breadboards with the reliability of PCBs. One approach is to fabricate circuits on paper or plastic with conductive ink, enabling users to hand-draw [19] or print [6, 12] circuit traces, and apply them to 3D objects [38]. Both PaperPulse [25] and PEP [23] further exploit printed circuits on paper with multiple layers, integrated editors, and custom assembly instructions. Circuit Eraser [21] and CircuitAR [20], focus on techniques to fix or debug a paper-based circuit, supporting iterative prototyping and reworking. While these approaches can be used to revise circuit topology, they still rely on re-printing (or re-drawing) the circuit layout, require permanent (e.g., soldering or z-tape) methods for attaching components, and do not scale to sophisticated circuits with many crossed connections. Modularized approaches such as stickers [10, 11, 16] solve the soldering problem but limit users to only pre-made components.

CircuitStack [33] combines the flexibility of solderless breadboards with the accuracy and speed of paper circuits. It achieves this by vertically stacking multiple paper layers containing the circuit's wiring (rendered in conductive ink) underneath a breadboard. Components can be placed on the breadboard without jumper wires, as connections are realized on the layered paper circuits. The authors demonstrate a working prototype with multiple conductive layers, and show that the system is easily configurable, accurate and supports rapid assembly of circuit components. However, as with other paper circuit techniques, making changes to the wiring is laborious. It involves redesigning and reprinting the paper stack and dis- and re-assembling the breadboard. We argue this prolonged process makes exploration and iteration of different layouts time consuming, and removes much of the beneficial flexibility of standard breadboards.

In contrast to previous work, VirtualWire enables the *programmatic* configuration of connections between components placed on a breadboard. This approach allows the topology of a circuit to be immediately and dynamically reconfigured, without the need for jumper wires, fabrication, or dis- and re-assembly.

2.2 Programmable hardware for prototyping

Programmable hardware is a large and established research area in electronics, covering topics such as diverse as gate-level hardware configuration, hardware simulations, and combinational and sequential logic. Due to the complexity and sophistication of these topics, research in these areas usually addresses an experienced audience of electronic engineers. Since the primary goal of this paper is a system that supports *novice* makers, we omit this literature and instead refer interested readers to Alphonsus and Abdullah's comprehensive survey [1]. Here, we will address the more relevant body of literature on programmable hardware systems for prototyping.

Previous work focused on easing the physical construction of circuits. VirtualComponent [14] allows users to place and tune "virtual" components—physically located on an external module—onto a physical breadboard to quickly experiment with different circuit designs by tuning the components' values. Similarly, VISIR [30] allows the construction of physical circuits from a remote workbench, where fixed components placed on an external bank can be routed to different locations on a remote breadboard. However, neither system supports the possibility of drawing arbitrary number of wires to connect components. Scanalog [28] supports the construction of analog circuits using high-level modules built into the hardware (using a field-programmable analog array - FPAA) that can be reconfigured and controlled in software, but there are major limits of the type and number of components that can be instantiated inside the FPAA chip (four configurable analog blocks). Proxino [35] does the exact opposite, by enabling user to create a *virtual prototype* for a circuit by blending the input or output value from a physical proxy (e.g., sensor or actuator) into a simulated virtual circuit. Specifically, with Proxino the input sensors or the actuators are physical proxies plugged into the hardware, while the circuit topology (i.e., the connections between components) only exists in software and behaves following a SPICE simulation. VirtualWire is different from all these approaches by allowing a reconfiguration of physical connections through software. Finally, Visible Breadboard [22] allows users to place components on a 12x-scale breadboard, and to construct a circuit without wires by linking any point to cardinally adjacent points by drawing the connections directly on the board surface with their fingers. While highly interactive and flexible, Visible Breadboard mandates manual configuration via touch; its large size makes using standard components challenging; and it does not support arbitrary connections, as each socket can only be connected to those cardinally adjacent to it. These factors make it difficult to produce and reconfigure circuit designs using the system.

Breadboards are not the only way to prototype circuits. LittleBits [4], DataFlow [7], Bloctopus [26] and NET Gadgeteers [32] all support circuit construction using modular building blocks. Despite many individual differences, these systems share the common objective of lowering the threshold for early-stage prototyping through pre-assembled modules with built-in capabilities that can be easily connected together to yield functional circuits. While these systems enable blocks of complex functionality such as cameras and displays, they also limit users to a strict set of pre-manufactured modules. Other research aims to simplify the circuit design process. For example, Trigger-Action-Circuits [2] automatically produces circuits and assembly instructions based on high-level descriptions of desired behavior, and AutoFritz [15] proposes auto-complete functionality to assist in the design of breadboarded circuits, based on a database of previously created common circuits. Finally, CircuitStyle [8] is a teaching tool that peripherally enforces best-practices and guidelines for implementing circuit prototypes. While these systems help with the design of circuit layouts, they still require users assemble circuits manually using the traditional process of plugging wires into a standard breadboard.

3 VIRTUALWIRE

VirtualWire is a novel tool for prototyping circuits that allows users to insert components on a physical breadboard and to connect them

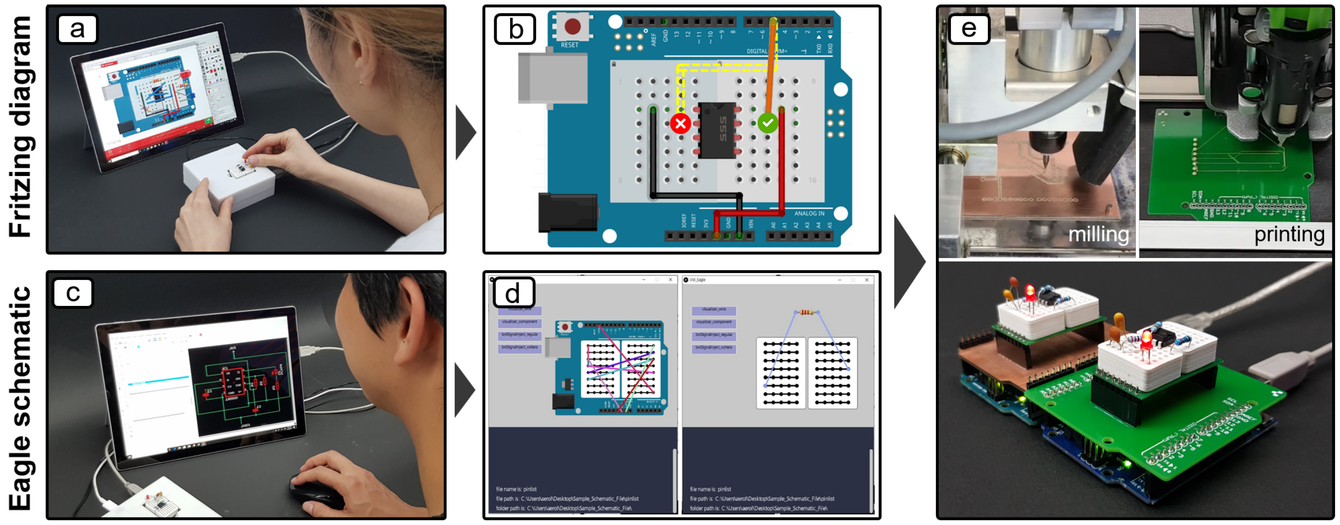


Figure 2: Step-by-step walkthrough from prototyping in Fritzing (ⓐ, ⓑ) or Eagle (ⓒ, ⓓ) to fabrication of PCBs (ⓔ). More details in the text.

without physically placing wires. Instead users can design the circuit layout using computer software (e.g., Fritzing[3] or Autodesk Eagle[4]) to specify *virtual wires* that describe how components should be connected. As connections are placed or updated, the virtual circuit layout is automatically updated on the physical breadboard in real time, producing real electric connections matching the virtual ones, creating a fully functional circuit. The final circuit design can be saved, shared, or automatically prepared for fabrication as a PCB.

VirtualWire's reconfigurability is enabled by a switching matrix that dynamically handles connections between any two points of the breadboard and between the breadboard and the pins of a user-programmable Arduino UNO embedded within the hardware. This switching matrix, in contrast to similar technology used in prior work [14, 22, 30], allows users to create unlimited and arbitrary connections among the components on the breadboard and/or the Arduino pins, without limiting the topology of the circuit. Furthermore, all modifications to the circuit design are immediately recognized and reflected in hardware, without requiring physical re-wiring or physical dis- and re-assembly tasks [33].

3.1 Design rationale

To establish constraints around an appropriate breadboard size, we extracted metrics relating to the size and complexity of circuits of typical Arduino projects in two common maker resources: the Arduino Project Book [3] and the online Fritzing projects repository[5]. These resources have been used as source of typical circuit designs in prior work [5, 15]. We manually examined the projects in the Arduino book, and obtained circuits from the Fritzing repository by scraping the website and removing duplicate entries, corrupted files, and files not containing any Arduino. The dataset is available online[6].

The Arduino book contains 15 projects with an average of 7.6 (SD: 3.7) unique electrical nets, 9.3 (SD: 4.7) wires, and 6.5 (SD: 3.6) components (Arduino included) with an average number of 3.35 connected pins (SD: 1.13) per component. Our Fritzing dataset has 4316 projects containing an average of 12.1 (SD: 12, median: 9) electrical nets, 16.9 (SD: 21.5, median: 11) wires, and 11.7 (SD: 16.4, median: 7) components. Each component has on average 3.4 (SD: 3.2, median 2.8) connected pins. Projects in both sets included both analog and digital components and operated with the typical voltage levels supported by the Arduino platform (3.3V and 5V).

Based on this data, we derived design requirements for VirtualWire. To match the maker communities it is intended for, it supports both Fritzing-style pictorial diagrams and electrical schematics, and both analog and digital circuits capable of interfacing with an Arduino—hence 3.3V or 5V logic levels, and a maximum operating current of about 50mA (Arduino digital pins can handle up to 40mA). We opted to embed an Arduino as a representative platform commonly used in maker communities—it can be programmed with standard tools and any pin can be connected to any of the breadboard nets. Finally, reflecting the small size of the circuits in the repositories ($nets \leq 12$, $wires \leq 17$), and in line with prior work [36, 37], we selected a breadboard with 16 rows (16 unique electrical nets) that can be linked by an unlimited number of interconnections (wires).

3.2 Workflow and system capabilities

In this section, we describe VirtualWire's capabilities with two linked walk-through examples. In the first, a user is building an oscillator circuit using a 555 chip in astable mode. She uses two

[3]https://fritzing.org
[4]https://www.autodesk.com/products/eagle/overview
[5]https://fritzing.org/projects
[6]https://github.com/makinteractlab/Dataset_CircuitsFritzing

resistors and one capacitor to calibrate the frequency and duty-cycle of the generated PWM signal to 10 Hz with a 50% duty-cycle. An LED and a current-limiting resistor are used to visualize the output blinking patterns. The user places the physical components on VirtualWire's physical breadboard, then uses a Fritzing file to place matching components on the virtual breadboard—a custom component containing a 16-row breadboard over an Arduino UNO as in Figure 2ⓐ. After checking that the virtual components' locations in Fritzing mirror those on the physical breadboard, she connects the components with wires in software. The 5V and GND pins of the embedded Arduino are used for power supply.

Once ready to test the circuit, the user starts up the VirtualWire software and specifies the input design file. This is then parsed and transmitted to the VirtualWire hardware, which renders all the connections automatically. On inspecting the design in Fritzing, the user realizes she made a transcription error and rectifies it by moving a wire to an adjacent pin (Figure 2ⓑ); after saving her design, VirtualWire automatically updates the hardware's state. She then launches the Arduino IDE, writes some test code and explores the behavior of her prototype.

In the second example, a more experienced maker familiar with schematics, is creating an analogous circuit, but working with electrical schematics instead of a Fritzing diagram. Using Autodesk Eagle, he draws a schematic of the oscillator circuit (Figure 2ⓒ). After exporting the schematic file as a pinlist (a ASCII file containing a list with pads and pins, with directions and names of the nets connected to the pins), he loads it into the VirtualWire software which automatically computes the position of all components on the breadboard via an algorithm, and visualizes them graphically on the screen (Figure 2ⓓ). The user is guided to place each component on the breadboard by on-screen instructions designed to ensure it matches the location of its virtual counterpart (Figure 2ⓓ, right). Any subsequent changes in the design file are reflected in how the breadboard rows and Arduino pins are connected.

Finally, both users are satisfied with their prototypes. To finalize the circuit, they use the VirtualWire software to export a PCB layout file that can be opened in Eagle. The layout file is shaped as an Arduino shield, with pre-made plated holes for headers and sockets. It contains the connections between the pins in the final circuit. They then run Eagle's auto-router to create physical tracks, export it for fabrication and ultimately render it for testing using conductive ink on a FR-4 substrate (e.g., using the Voltera machine[7]) or milling it onto a copper substrate for final production (see examples in Figure 2ⓔ). While the PCB requires manual assembly to solder headers and sockets, individual components do not need to be placed or soldered. Instead, the breadboard with the components can be removed from the VirtualWire box and placed as-is on top of the fabricated PCB for immediate use.

4 IMPLEMENTATION

VirtualWire is composed of an open-source[8] physical augmented breadboard and PC software that controls it and displays a user interface.

[7]https://www.voltera.io
[8]https://github.com/makinteractlab/VirtualWire

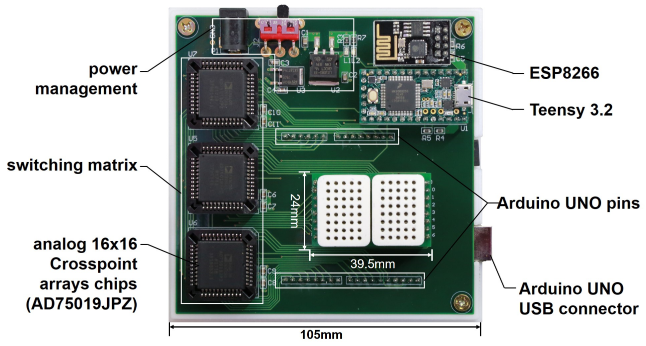


Figure 3: VirtualWire hardware components. The breadboard module connects via standard pluggable headers and is removable.

4.1 Hardware

The hardware (Figure 3) consists of a controller board and a removable (or swappable) mini-breadboard module. As in prior work [37], the breadboard module contains 8 rows and 10 columns (split in two banks) with a standard 100mil pitch (2.54mm) between holes. It is 33 mm by 21 mm and 7 mm high. It can support up to 16 distinct electrical nets that can be connected together arbitrarily. The breadboard module is mounted on top of a custom PCB that connects to the controller board via standard pluggable pins and headers. The controller board houses a micro-controller (Teensy 3.2), a wireless bridge (ESP8266), sub-circuitry to handle both 3.3V and 5V logic levels, a user-programmable Arduino UNO, and a switching matrix. The system is powered by a 12 V DC power supply, and is enclosed in a 10.5 x 10.5 by 4 cm box.

The switching matrix is composed of three analog 16x16 crosspoint arrays chips (AD75019JPZ) in cascade mode controlled with TTL logic by the Teensy microcontroller. Two chips (U1 and U2) are used to connect any of the 16 rows of the breadboard to any of the pins of the embedded Arduino UNO, including its 14 digital pins, 6 analog pins, and pins for power management (3.3V, 5V, GND, VIN, IORED, RESET and AREF). These connections have a typical

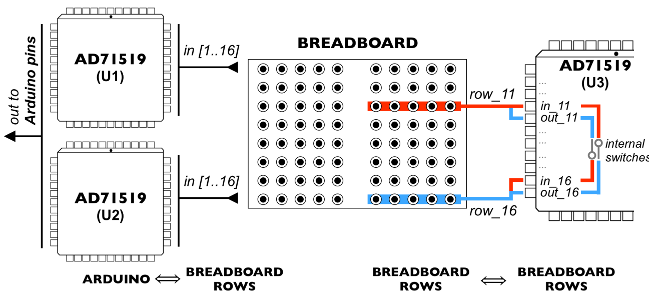


Figure 4: Block diagram of the switching matrix: U1 and U2 provide connections between any row in the breadboard and any pin of the embedded Arduino. U3 provides a double-path from any row of the breadboard to any other row.

on-resistance of 150Ω, hence capable of handling a maximum current of 33mA at 5V (hence about 160mW, where the AD75019JPZ chip is capable of dissipating up to 1W in total). These ratings are compatible with the Arduino's maximum rated current for digital pins (about 40mA). The third chip (U3) is used to handle connections between any two rows in the breadboard. This is achieved by shorting the input and output sockets of the chip, so that each row of the breadboard can simultaneously act both as input and output. For example, the diagram in Figure 4 shows how rows 11 and 16 are connected – internally the chip routes the input of row 11 to the output of row 16, and vice-versa. With this configuration we enable two distinct routes between any two points of the breadboard, lowering the on-resistance to 75Ω and doubling the current threshold. This enables the use of external power supplies on the breadboard to drive components with higher current requirements, such as DC or servo motors.

4.2 Software

The VirtualWire software is a collection of scripts managed by a Java application that monitors the currently loaded input design files – these can be either Fritzing project files or Eagle schematic files. VirtualWire extracts wire connections from circuit diagrams generated with Frtizing by parsing its standard utf-8 encoded XML files. Wires in Fritzing files are sequences of lines connecting specific breadboard locations so we achieve this simply by tracing each line to its start and end point. This results in a list of connections that are formatted into a series of commands in JSON format and sent as packets of up to 1024 bytes to the VirtualWire hardware wirelessly or over serial at 115200 bps. When working with Fritzing, VirtualWire presents no on-screen UI—users just work with Fritzing as normal, but any change they save to their circuit is immediately and automatically pushed to the VirtualWire breadboard.

Eagle schematics succinctly store circuit topology and nets. However, they do not indicate how components should be placed on a breadboard, the information required by the VirtualWire hardware. To create this concrete representation, we developed a solver in Python that automatically assigns each component in a Eagle schematic to a valid position on the breadboard. The solver works for a wide (and extensible) range of components: resistors; capacitors; inductors; potentiometers; diodes; LEDS; buttons; servos; batteries and; ICs. It requires each be labelled following a strict naming convention based on the initial of the component type followed by a unique number (e.g., R1 and R2 should be the component names used in a circuit with just two resistors).

The solver then operates as follows: it loads a pinlist from an Eagle file, internally converts it to a set of electrical nets and determines if the circuit can fit on the VirtualWire breadboard. If not, the user is informed and the process halts. If the circuit fits, the solver then generates a concrete routing. It starts by assigning DIP packages that straddle the breadboard, then goes through all other nets and randomly assigns them to the remaining rows. These simple heuristics ensure placement of components that have adjacent pins (e.g., ICs) are valid and helps distribute the remaining components evenly across the breadboard, avoiding clusters. The random assignment of nets to rows also exploits the small size of the current VirtualWire hardware: we assume components with wire leads (e.g., resistors, capacitors, etc.) can be freely positioned anywhere on the breadboard, regardless of where their other lead(s) are placed. Once all nets are placed on the breadboard, the solver computes the location of each component and wire, and internally stores them as descriptors in JSON format. These descriptors are then sent to the VirtualWire hardware and also visualized on the PC screen as a Fritzing style pictorial circuit diagram.

VirtualWire can also export circuits as Eagle's proprietary *brd* files, containing the layout for an Arduino shield and the necessary holes to house the system's removable mini-breadboard. Our software works by injecting the layout file with the connections for the electrical nets and by assigning them to specific pins. This file can then be used to produce a PCB with any conventional method (e.g., milling, printing).

5 EXAMPLE USAGES

This section contains examples of applications that demonstrate VirtualWire's capabilities and uses. The goal is to provide an illustrative and grounded description of how VirtualWire can support makers in their circuit design, development and construction practices. It is organized into four key activity types: building, exploring/tuning circuits and, finally, integration with existing practices.

5.1 Building circuits from scratch

VirtualWire enhances the basic process of constructing new circuits with both analog and/or digital components. For example, Figure 5 shows a battery powered micro servo motor (SG90) controlled using a 5K potentiometer—moving the wiper results in movements of the servo motor's shaft. This is achieved by connecting the wiper pin of the potentiometer to one of the Arduino's analog pins, and then mapping the read value to the duty-cycle of the PWM (Pulse-Width Modulation) pin connected to the servo. This simple example highlights VirtualWire's ability to deal with external power sources and mixed-domain signals.

A more complex example, adapted from an online repository[9], uses a dual operational amplifier (LM358), six resistors, one capacitor, one LED, and a 3.5mm audio jack (10 components and 10 wires in total) to visualize an audio signal (Figure 6). The input audio signal is acquired through the audio jack, low-pass filtered (in hardware) and fed into an analog pin of the Arduino. In software,

[9]http://fritzing.org/projects/op-amp-low-pass-filter-for-35-audio-jack

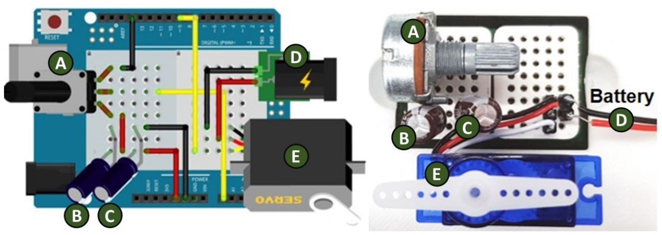


Figure 5: The Fritzing circuit used to power and control a servomotor using a potentiometer (left) and its physical layout in hardware (right). Letters show the components' mapping.

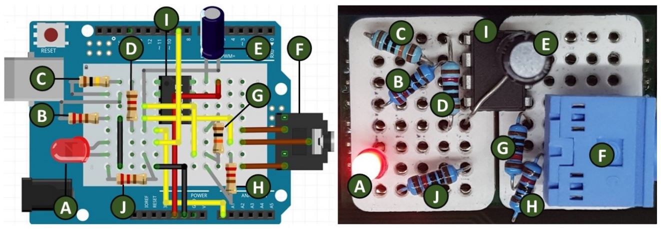


Figure 6: Fritzing pictorial circuit diagram and VirtualWire breadboard realizing a sound level visualizer circuit.

these readings are again mapped to the duty-cycle of a PWM signal that controls the LED brightness. The resultant system varies the brightness of the LED in real time according to the audio volume. We use this more sophisticated example to illustrate that VirtualWire is capable of supporting typical circuits showcased in the maker community's online repositories, involving processes such as reading analog data, hardware filtering, and PWM output.

5.2 Exploring variations

Circuit designers often consider possible alternative hardware configurations. VirtualWire simplifies exploring these different options. For example, a user may wish to learn about the operation of a component through hands-on practice. A quad 2-Input NAND gate chip (SN74LS00N), for instance, offers the opportunity for students to learn about logic gates by practicing different layout configurations. In Figure 7, two SPDT switches are used to control the logic input for the quad NAND gate chip, and the output is fed into an LED that turns on when the logic output is high (e.g., 5V), or off when low (0V). We explored different wiring configurations to build AND, OR or XOR gates. As a result, the LED lights up in different situations: only when both switches are on when configured as an AND gate; when at least one switch is on for an OR gate and; when the two switches have different states for a XOR gate. Note that each configuration can be saved and loaded at will, enabling quick switching between alternatives.

A more practical example involves digital modules that support various complementary features, such as different communication

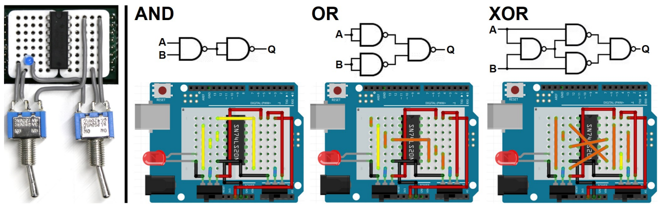


Figure 7: VirtualWire breadboard with two switches and quad 2-Input NAND gate (left). Other images show wiring configuration for AND, OR and XOR gates. Note that the wires on the breadboard (left) are not interconnections; they simply connect the large switch PINs to the breadboard. Wire reconfigurations are highlighted in orange.

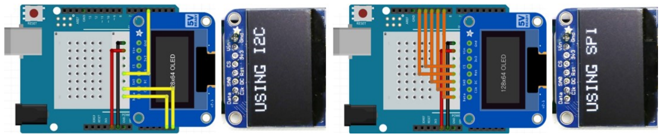


Figure 8: Fritzing pictorial circuit diagrams showing the wiring required to connect an OLED display via SPI or I^{2}C protocols. Change between these protocols is a software-only process with VirtualWire. Wire reconfigurations are highlighted in orange.

buses (e.g., UART, SPI, I^{2}C). Switching between these options typically involves physical re-wiring, in addition to software changes, making testing out the various options laborious. VirtualWire simplifies this process by providing a fully virtual environment for establishing the different connections required for each option—Figure 8 shows how this works for the UG-2864HSWEG01 OLED display, featuring communication through either SPI or I^{2}C.

5.3 Rewiring for tuning

Beyond the key feature of enabling breadboarding without physical wires, VirtualWire also supports rapid and seamless adjustments. In fact, after a design file has been linked to the system, any changes to the virtual electrical nets are automatically detected and pushed to the hardware. This feature can also be used to optimize specialized circuit design sub-tasks, such as picking the value of components from among a limited set of alternatives. For example, Figure 9 illustrates tuning a low-pass filter by adjusting the values of the resistors (R) and capacitors (C) in its RC network. The breadboard has four resistors (330Ω, 15kΩ, 200kΩ, 510kΩ) and three capacitors (1nF, 10nF, and 100nF) arranged to support tuning a filter that takes a 5V 50% duty-cycle PWM signal and produces a steady voltage level. By placing this set of components on the physical breadboard, a user can repeatedly rewire the circuit, exploring 12 possible filters

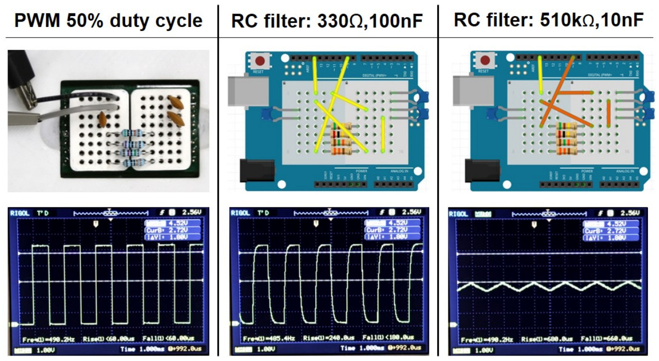


Figure 9: VirtualWire breadboard with various resistors and capacitors to support tuning a filter (left). Center and right show Fritzing pictorial circuit diagrams with the connections for different filters. Bottom row shows original signal (left) and filtered outcomes (center and right). Wire reconfigurations are highlighted in orange.

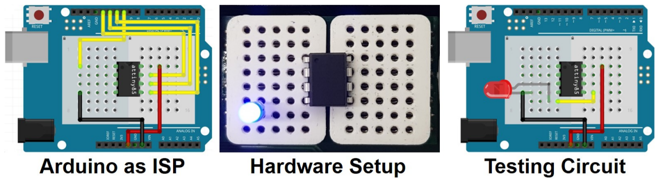


Figure 10: Frizting pictorial circuit diagram for an Arduino to act as ISP for an AVR micro-controller (left); Diode blinking circuit for the AVR micro-controller (right). Center shows VirtualWire breadboard for both circuits; switching between these functions can be achieved simply by loading different Fritzing sketches.

to rapidly identify an optimal solution (in this case, $510k\Omega$ and 10nF) that will output a steady 2.5V signal.

5.4 Integration with traditional workflows

Finally, VirtualWire integrates with design and fabrication tools widely adopted in the maker community. Circuits can be designed using Fritzing or Autodesk Eagle, with VirtualWire software running in the background to observe circuit changes. Once complete, users have the option to print custom PCBs with their tools of choice. For example, Figure 2 shows the oscillator circuit from the walk through rendered into printed and milled Arduino shields. We also note that VirtualWire allows designers to rely on software tools and practices typically used for collaborative work with digital contents. Thanks to Fritzing and Eagle, VirtualWire supports undo and redo operations. Circuits can also be easily shared between makers and iterations can be tracked using versioning software such as git.

VirtualWire can also serve as a chip-programmer tool. For example, Figure 10 (left) shows VirtualWire's internal Arduino set up as an In-System Programmer (ISP) for flashing the popular 8-bit AVR microcontroller ATtiny85[10]. This setup allows a user to write code in the Arduino IDE and flash it directly to the ATtiny chip without any other hardware. Code on the ATtiny can then be tested by simply loading a new circuit design file and observing the results—Figure 10 shows a circuit layout for testing code that blinks an LED. Switching between flashing code to the ATtiny and running code on the ATtiny is rapid and requires no physical re-wiring.

6 TECHNICAL EVALUATION

We characterized the performance of the system by measuring the electrical characteristics of connections comprised of 1 to 15 wires in series (i.e., each wire connects two adjacent breadboard rows). The resulting documentation of stray capacitance and associated resistance highlights the system's performance under various use cases. We considered two representative cases of Arduino use: a static configuration (5V direct current from an external power supply), and a dynamic configuration (5V PWM at 490.20 Hz with 50% duty cycle as default on Arduino UNO). Both sources were applied at row 1 and measured at all the other rows. Table 1 shows the collected values for voltage, current, resistance, power, power loss ($L_p = 10 * log_{10}(P_{out}/P_{in})$), as well as rise- and fall-time for the PWM signal. Static measures were taken using a Fluke 177 multi-meter , while dynamic measures were recorded using the signal generator and oscilloscope tools within the Digilent Analog Discovery 2.

[10]https://www.microchip.com/wwwproducts/en/ATtiny85

Table 1: Electrical characteristics of connections made between 1 and 15 wires in series. Input at 5V, 50mA.

Wires	V_{out}	I	R	ΔR	P	L_p	Rise	Fall
	mV	mA	Ω	Ω	mW	dB	us	us
1	5016	49.6	101		248.8	-0.02	0.56	0.13
2	5016	26.1	193	91	130.9	-2.81	0.94	0.49
3	5016	17.9	281	88	89.8	-4.45	1.2	0.84
4	5016	13.6	369	88	68.2	-5.64	1.4	1.07
5	5016	11	458	89	55.2	-6.56	1.58	1.24
6	5016	9.2	546	88	46.1	-7.34	1.7	1.36
7	5016	7.9	634	88	39.6	-8.00	1.83	1.47
8	5016	7	722	88	35.1	-8.52	1.89	1.52
9	5016	6.2	809	87	31.1	-9.05	1.97	1.63
10	5015	5.6	897	88	28.1	-9.49	2	1.64
11	5015	5.1	985	88	25.6	-9.90	2.05	1.68
12	5015	4.7	1074	89	23.6	-10.26	2.06	1.7
13	5015	4.3	1161	87	21.6	-10.64	2.09	1.77
14	5015	4	1248	87	20.1	-10.96	2.09	1.72
15	5015	3.8	1334	86	19.1	-11.18	2.03	1.71
Avg.	5015.6	11.7	720.7	88.0	58.9	-7.7	1.7	1.3
SD	0.5	12.1	393.5	1.2	60.9	3.2	0.5	0.5

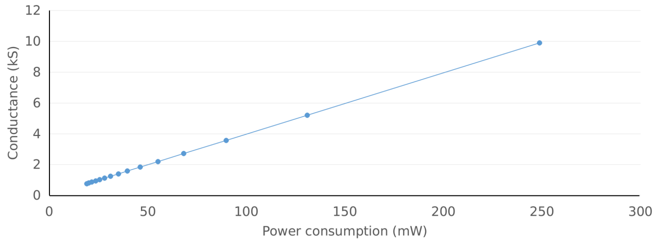


Figure 11: Linearity of power and conductance.

Results show an average resistance per connection of 88Ω, and that the total resistance is linearly proportional to the number of wires. Figure 11 shows that conductance ($G = 1/R$) is linearly proportional to the power consumption, or, in other terms, that resistance is inversely linearly proportional to power. Both rise- and fall-time show that a timed signal is substantially unaffected by routing, thus making VirtualWire particularly suitable for digital circuits which work with TTL logic and have low current requirements. To demonstrate this, we connected a servomotor control-signal wire at row 16 and were able to control the motor from the PWM input supplied at row 1 and traversing all the other rows.

7 USER EVALUATIONS

We conducted two separate studies to assess the efficiency and overall experience of prototyping with VirtualWire. We opted for two separate evaluations with distinct groups of users rather than a combined study because of the lengthy prototyping sessions required.

7.1 Study 1: Quantitative evaluation

The first study compared objective measures of performance between VirtualWire and traditional breadboarding. We conducted a quantitative evaluation with 12 participants aged 21-33 (M: 24.1, SD: 3.37, 4 females, 8 males), all undergraduate and graduate students from various engineering departments and industrial design. All students had previous experience using Arduino and physical computing on traditional breadboards. Six participants self-rated themselves on a scale of 1 to 7 as advanced makers (M: 4.8, SD: 0.9) and six as novices (M: 2.5, SD: 0.8). Each session lasted approximately one hour, and participants were compensated with 10 USD in local currency.

After a brief demonstration of VirtualWire, participants were asked to perform four circuit-assembly tasks using either VirtualWire or traditional jumper wires on a breadboard. The study followed a balanced within-subject repeated-measures design with two consecutive tasks performed in each of the two conditions. Circuits were chosen from the example applications described earlier in the paper in order to present tasks of varying complexity. These were 1) a blinking LED using a 555 timer chip; 2) wiring an OLED display to an Arduino UNO via I^2C and SPI protocols; 3) building AND/OR logic gates using a NAND gate chip; and 4) controlling a servo motor with a 5K potentiometer. For each circuit, participants were provided with a printed image showing the complete Fritzing diagram of the finished circuit.

Participants were given a maximum of ten minutes to complete each circuit, otherwise the task was recorded as a failure. At any time participants could submit their circuit to an experimental moderator, who immediately checked it for correctness. If correct, the task finished. If incorrect, participants were informed there was one or more errors (but not what they were) and asked to continue the task. Participants were able to submit circuits multiple times for each task up until the expiry of the ten minute time limit.

All times at which participants' submitted circuits to be checked were logged. If a submitted circuit was correct, the time it was submitted was considered to be the *task completion time.* The total *number of checks* requested was also recorded as was the overall *success* or failure of each circuit completion task. In order to record mistakes and errors at a finer granularity, all study sessions were recorded with a downwards facing camera positioned directly above the VirtualWire board or breadboard. Additionally, all VirtualWire sessions were recorded with a screen capture system. *Errors* in this material were defined as placing a component or wire in a physical or virtual position that deviates from the provided Fritzing diagram. In order to extract this information, two raters examined all videos according to a pre-agreed classification scheme in which errors were logged as either *self-detected errors* (e.g., corrected independently by the participant) or *at-check errors* (e.g., present in a circuit submitted for check) and involving either *missing* or *misplaced* wires and components or *inverted* components. To alleviate bias and correct for mistakes in rater performance, the lists of errors recorded by the two raters were compared and the differences examined and resolved into a unified final set that both raters endorsed as accurate.

7.1.1 Results. The results revealed a wide range of differences in performance. Firstly, two circuits (8.3%) in the breadboard condition were not completed, while all circuits in the virtual wire condition were successfully completed. A Fisher exact test [27] indicated this difference was not significant (p=1.0). Shapiro-Wilk tests indicated data for task completion time was normally distributed and a subsequent t-test showed that VirtualWire led to significantly more rapid task completion times (p<0.001) than standard breadboarding: a mean of 206 seconds (SD 45.3) vs 328.6 seconds (SD 75.4). The number of checks requested by users was not normally distributed and transforms did not resolve this. Accordingly, we analyzed data with non-parametric Wilcoxon tests. No significant difference was observed between the mean of 1.29 (SD 0.4) checks requested in the VirtualWire condition and the mean of 1.5 (SD 0.48) checks requested in the breadboard condition. Error counts were also not normally distributed, but responded to a power transform ($\lambda = -0.825$). A t-test on the normally distributed transformed data indicated that VirtualWire led to significantly fewer errors than breadboarding (p=0.027). We present a detailed breakdown of errors according to the classification scheme used by the raters in Table 2.

Table 2: Breakout of errors by occurrence and type.

Error-type	Self-corrected errors		Errors at check	
	Baseline	VirtualWire	Baseline	VirtualWire
Missing Wire	0	0	5	0
Missing Component	0	0	0	0
Inverted Component	2	0	2	0
Misplaced wire	8	10	25	11
Misplaced Component	4	4	12	2
Total	14	14	44	13

To provide a more detailed depiction of performance in this complex study task, Figure 12 plots the duration of each trial and highlights the times at which check requests, self-detected errors and at-check errors occurred. The shorter task times and reduced number of at-check errors in the VirtualWire condition are immediately evident, as are the uneven distributions that result in the failure of the raw data to satisfy normality assumptions. Together with the formal statistical analysis, this visualisation provides supportive evidence for the claim that VirtualWire can improve not only the speed with which novice users can produce breadboarded circuits but also the accuracy and correctness of their work. It is particularly noteworthy that VirtualWire reduced the number of at-check errors, or those that a user failed to catch without a specific prompt. We suggest this type of error may be particularly challenging for a user to detect and debug during real circuit creation activities, when there may be confusion as to when errors were committed, which components they involve and even whether they lie in software or in hardware.

7.2 Study 2: Qualitative Evaluation

To complement the quantitative findings, we performed an interview-based qualitative study with 12 new participants aged 20-27 (M: 22.6,

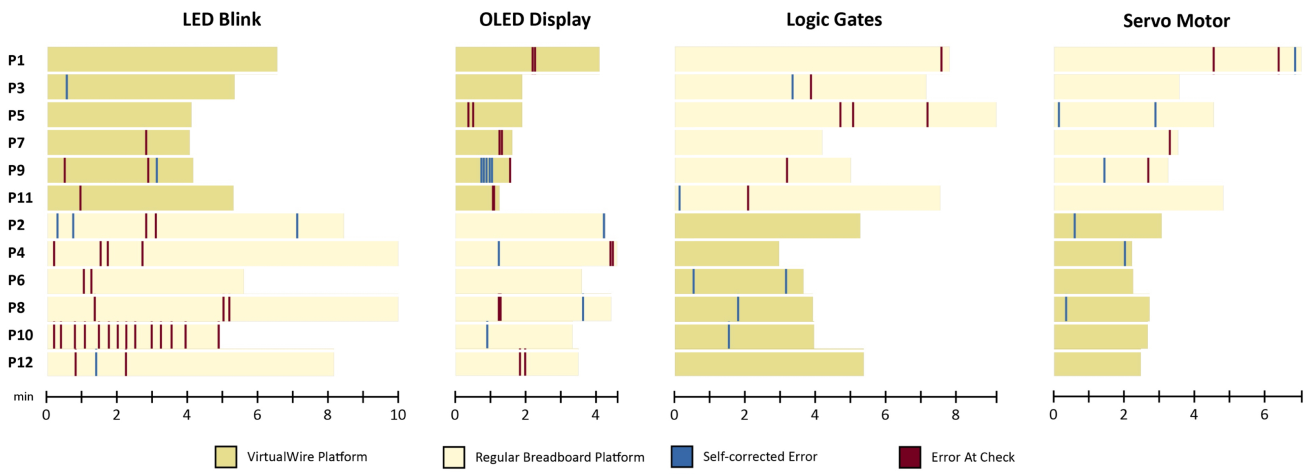


Figure 12: Visualization per user of the breadboarding sessions with four different circuits, in the VirtualWire and baseline condition. Errors are visualized at the time of occurrence.

SD: 2.6, 3 females). Participants used VirtualWire to assemble three circuits on a breadboard, and then to compare these sessions with their prior experience in building circuits. Participants consisted of undergraduate students, graduate students or recent graduates from various engineering departments and industrial design. All participants had some prior experience with physical computing and Arduino, with six participants self-rating themselves as advanced makers (M: 5.6 out of 7, SD: 0.5) and six as novices (2.1 / 7, SD: 1.1). The study lasted one hour, and participants were compensated with 10 USD in local currency.

The study followed a simple format: after a brief demonstration of VirtualWire and a warm up session to familiarize themselves with the system (max 10 minutes), similarly to the quantitative study, participants were asked to build three circuits of various complexity and with different numbers of wires: 1) rewiring logic gates, 2) connecting a 7-segment display to the Arduino, and 3) connecting the OLED display to the Arduino with the I^2C and SPI protocols. The task order was balanced among participants.

7.2.1 Results. Interviews were transcribed and analyzed by two researchers using open and axial coding methods. Three key findings emerged, relating to the utility of VirtualWire, the ease with which it could integrate with existing workflows and the applications and domains it might be particularly useful for.

It terms of utility, all participants, regardless their level of expertise, praised the system's ability to reduce the clutter and fragility of physical breadboards: *"it is physically difficult to place wires on a crowded breadboard, but in VirtualWire its so much easier (P4)."* This was useful for several reasons. Firstly, it was much easier to read the breadboard as they reported they could see at a glance what wires are going where. This not only sped up circuit assembly and debug, but also enabled them to better optimize their use of space: *"there is much more space on the breadboard that can be used (P4)"* compared to a traditional breadboard. Participants also noted they could *"work on the software and spend less time on the physical breadboard (P11)"*, which hints at improved integration between the software and hardware aspects of a physical computing project [5]. P3 highlighted the value of this integration: *"From a beginner's perspective, the process of drawing the circuit [and] transferring it to a breadboard seems disconnected and burdensome. With this system, the steps are better connected, helping beginners be more confident in prototyping."*

Most participants noted that VirtualWire integrates well with normal prototyping workflows, rather than disrupt or change practices. For example, P6 stated that *"it is especially nice when you must rewire one or multiple components"* and you can use the mouse and graphical tools to tidy the wiring. P1 also commented that the possibility of saving the wiring at any stage of completion boosted his confidence. P9 went further observing that *"the undo and redo features in the software are a huge advantage . . . I can rely on the software to remember the individual steps I made wiring the circuit."* We note these features are part of standard software environments and the innovation in VirtualWire is to extend their reach from PC based design tools out into the actual physical breadboard a user is working on. VirtualWire also allows participants to seamlessly produce a final PCB board layout ready for production: *"when I transfer from breadboard to PCB I take a lot of pictures as a reference. With [VirtualWire] I don't think I'll need to do that (P3)"*.

Finally, based on their past experiences, participants speculated on several possible applications for VirtualWire, including educational training (P2, P5, P7), testing components and sensors (P1, P5, P6, P9), and exploring different variations of a circuit (P10, P11). Other participants compared VirtualWire with Autodesk TinkerCAD[11], a Fritzing-like tool with a built-in simulator. P10 appreciated VirtualWire moves beyond such tools as while *"in TinkerCAD, there is a simulation feature, here you can try for real."*

[11]https://www.tinkercad.com

8 DISCUSSION AND LIMITATIONS

This paper presented VirtualWire, highlighted its potential uses and described two studies that illustrate how these translate into concrete benefits to users. Specifically, the quantitative study indicates that VirtualWire is substantially faster than traditional prototyping with breadboards, showing a substantial 37% improvement in completion times. This is due, at least in part, to the fact that VirtualWire also leads to significantly fewer errors (53%)—users who make fewer mistakes are likely to complete tasks more quickly. Indeed, a detailed examination of errors shows a strong reduction in the number of mistakes that are not independently identified—they are only isolated after an explicit prompt (reported as at-check errors in the study). The qualitative study corroborates these results. Participants stated they were able to quickly familiarize themselves with VirtualWire and create neat, space-efficient designs. They also appreciated the ability to apply standard productivity techniques from the digital domain, such as undo/redo and saving intermediary versions, to a physical wiring task. Participants speculated this would enable them to spend less time on physical breadboarding and more on programming, a shift they viewed would be particularly advantageous in domains such as education and in tasks such as tinkering or exploring and testing alternatives. Together, these results provide strong evidence that VirtualWire can improve on the ability of novices to rapidly, effectively and meaningfully prototype circuits.

Despite these positives, we note there are numerous opportunities for technical improvement of the VirtualWire prototype. Perhaps the most substantial limitation of system relates to the size and electrical properties of the breadboard—an issue VirtualWire shares with closely related prior work [14, 36, 37]. Although the system was designed with the mean circuit size of a typical maker's project in mind (see Design Rationale section), VirtualWire cannot currently support large circuits. Study participants reported that they expected to make more efficient use of the breadboard space when wiring in software, but ultimately the system has an upper bound of 16 distinct electrical nets. This limitation stems from use of the 16x16 AD71519 analog crosspoint-array switch. The crosspoint switch is also the root cause of VirtualWire's parasitic capacitance between adjacent rows (65pF) and the resistance associated to each connection (88Ω per wire). Although this number is higher than the resistance of jumper wires used with breadbaords (typically less than 1Ω), it is similar to that reported for CircuitStack [33] — between 1Ω and 104Ω depending on the length of the connection. With the current implementation, a user needs to work around these constraints by, for example, dividing a complex circuit into modules that can be designed, tested and fabricated separately on different mini-breadboards. If higher current is needed, the input voltage to the system can be raised (up to 12V for a maximum dissipation of 1W), physical wires can be placed on top of the breadboard or multiple connections between points on the board can be realized using parallel routes. Resolving these issues may also be possible with alternative implementations based on technologies such as a larger crosspoint-array switch, FPGAs or FPAAs; future work should explore these options.

Further limitations relate to the level of assistance provided by the system. VirtualWire cannot determine whether the topology of a circuit is correct, automatically detect wrongful placement of components [9, 22, 36], or provide suggestions for how to complete a circuit [15]. In fact, the system does not currently detect or indicate whether components were placed in the suggested breadboard locations. Developing and studying an integration with such advanced functionality, via the Fritzing and Eagle platforms common to both VirtualWire and these prior projects, is a clear next step for this work. Additionally, the PCBs fabricated by VirtualWire are limited in their utility: they are large and have a fixed Arduino shield form factor. While this is convenient, future work should enable better integration of designs with prototyping and fabrication tools for customized boards [39]. Finally, future work should also investigate the impact of VirtualWire on long-term learning and how it might facilitate moving from prototypes to products [13].

9 CONCLUSIONS

This paper presents VirtualWire, a system that allows users to rapidly create and modify circuit layouts using standard existing software tools (e.g., Fritzing, Eagle) and have their designs and changes immediately, dynamically and automatically realized as electrical connections on a real breadboard. The goal is to allow users to combine the quick, exploratory qualities of circuit breadboarding with real components with the clarity, convenience and edit-ability of working with digital representations of circuit topology. We describe the design rationale, present the systems' technical capabilities and showcase example uses. Our quantitative and qualitative evaluations show that VirtualWire reduces the time taken and errors that occur during circuit assembly, while also improving desirable qualities such as the *"neatness"*, legibility and compactness of circuit layouts and integrating seamlessly with existing high-value digital functionality such as undo/redo. Finally, we highlight the technical limitations of the system and discuss opportunities for future research, such as improved integration with prior work, and implementation of more comprehensive PCB fabrication and customization options.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Ephrem Ryan Alphonsus and Mohammad Omar Abdullah. 2016. A review on the applications of programmable logic controllers (PLCs). *Renewable and Sustainable Energy Reviews* 60 (2016), 1185 – 1205. https://doi.org/10.1016/j.rser.2016.01.025

[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* (Qu&#233;bec City, QC, Canada) *(UIST '17)*. ACM, New York, NY, USA, 331–342. https://doi.org/10.1145/3126594.3126637

[3] Arduino.cc. 2012. *Arduino Projects Book.* Retrieved Sep 10, 2018 from https://store.arduino.cc/usa/arduino-starter-kit

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

[5] 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* (San Jose, California, USA) *(CHI '16)*. ACM, New York, NY, USA, 3485–3497. https://doi.org/10.1145/2858036.2858533

[6] Varun Perumal C and Daniel Wigdor. 2015. Printem: Instant Printed Circuit Boards with Standard Office Printers &#38; Inks. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software &#38; Technology* (Charlotte, NC, USA) *(UIST '15)*. ACM, New York, NY, USA, 243–251. https://doi.org/10.1145/2807442.2807511

[7] Alvaro Cassinelli and Daniel Saakes. 2017. Data Flow, Spatial Physical Computing. In *Proceedings of the Eleventh International Conference on Tangible, Embedded, and Embodied Interaction* (Yokohama, Japan) *(TEI '17)*. ACM, New York, NY, USA, 253–259. https://doi.org/10.1145/3024969.3024978

[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* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 109–120. https://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* (Tokyo, Japan) *(UIST '16)*. ACM, New York, NY, USA, 677–686. https://doi.org/10.1145/2984511.2984566

[10] Steve Hodges, Nicolas Villar, Nicholas Chen, Tushar Chugh, Jie Qi, Diana Nowacka, and Yoshihiro Kawahara. 2014. Circuit stickers: peel-and-stick construction of interactive electronic prototypes. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1743–1746.

[11] Hsin-Liu (Cindy) Kao, Christian Holz, Asta Roseway, Andres Calvo, and Chris Schmandt. 2016. DuoSkin: Rapidly Prototyping On-skin User Interfaces Using Skin-friendly Materials. In *Proceedings of the 2016 ACM International Symposium on Wearable Computers* (Heidelberg, Germany) *(ISWC '16)*. ACM, New York, NY, USA, 16–23. https://doi.org/10.1145/2971763.2971777

[12] Yoshihiro Kawahara, Steve Hodges, Benjamin S. Cook, Cheng Zhang, and Gregory D. Abowd. 2013. Instant Inkjet Circuits: Lab-based Inkjet Printing to Support Rapid Prototyping of UbiComp Devices. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (Zurich, Switzerland) *(UbiComp '13)*. ACM, New York, NY, USA, 363–372. https://doi.org/10.1145/2493432.2493486

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

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

[15] 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* (Glasgow, Scotland Uk) *(CHI '19)*. ACM, New York, NY, USA, Article 403, 13 pages. https://doi.org/10.1145/3290605.3300633

[16] Eric Markvicka, Guanyun Wang, Yi-Chin Lee, Gierad Laput, Carmel Majidi, and Lining Yao. 2019. ElectroDermis: Fully Untethered, Stretchable, and Highly-Customizable Electronic Bandages. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. ACM, New York, NY, USA, Article 632, 10 pages. https://doi.org/10.1145/3290605.3300862

[17] David A. Mellis and Leah Buechley. 2011. Scaffolding Creativity with Open-source Hardware. In *Proceedings of the 8th ACM Conference on Creativity and Cognition* (Atlanta, Georgia, USA) *(C&#38;C '11)*. ACM, New York, NY, USA, 373–374. https://doi.org/10.1145/2069618.2069702

[18] 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* (Brisbane, QLD, Australia) *(DIS '16)*. ACM, New York, NY, USA, 1270–1281. https://doi.org/10.1145/2901790.2901833

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

[20] Koya Narumi, Steve Hodges, and Yoshihiro Kawahara. 2015. ConductAR: An Augmented Reality Based Tool for Iterative Design of Conductive Ink Circuits. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (Osaka, Japan) *(UbiComp '15)*. ACM, New York, NY, USA, 791–800. https://doi.org/10.1145/2750858.2804267

[21] Koya Narumi, Xinyang Shi, Steve Hodges, Yoshihiro Kawahara, Shinya Shimizu, and Tohru Asami. 2015. Circuit Eraser: A Tool for Iterative Design with Conductive Ink. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI EA '15)*. ACM, New York, NY, USA, 2307–2312. https://doi.org/10.1145/2702613.2732876

[22] Yoichi Ochiai. 2014. Visible breadboard: System for dynamic, programmable, and tangible circuit prototyping with visible electricity. In *International Conference on Virtual, Augmented and Mixed Reality*. Springer, 73–84. http://link.springer.com/chapter/10.1007/978-3-319-07464-1_7

[23] Hyunjoo Oh, Tung D. Ta, Ryo Suzuki, Mark D. Gross, Yoshihiro Kawahara, and Lining Yao. 2018. PEP (3D Printed Electronic Papercrafts): An Integrated Approach for 3D Sculpting Paper-Based Electronic Devices. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. ACM, New York, NY, USA, Article 441, 12 pages. https://doi.org/10.1145/3173574.3174015

[24] Ronald J. Portugal. 1973. Breadboard for electronic components or the like. https://patents.google.com/patent/USD228136 US Patent USD228136S.

[25] Raf Ramakers, Kashyap Todi, and Kris Luyten. 2015. PaperPulse: An Integrated Approach for Embedding Electronics in Paper Designs. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. ACM, New York, NY, USA, 2457–2466. https://doi.org/10.1145/2702123.2702487

[26] Joel Sadler, Kevin Durfee, Lauren Shluzas, and Paulo Blikstein. 2015. Bloctopus: A Novice Modular Sensor System for Playful Prototyping. ACM Press, 347–354. https://doi.org/10.1145/2677199.2680581

[27] Peter Sprent. 2011. *Fisher Exact Test*. Springer Berlin Heidelberg, Berlin, Heidelberg, 524–525. https://doi.org/10.1007/978-3-642-04898-2_253

[28] 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* (Qu&#233;bec City, QC, Canada) *(UIST '17)*. ACM, New York, NY, USA, 321–330. https://doi.org/10.1145/3126594.3126618

[29] Evan Strasnick, Sean Follmer, and Maneesh Agrawala. 2019. Pinpoint: A PCB Debugging Pipeline Using Interruptible Routing and Instrumentation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. ACM, New York, NY, USA, Article 48, 11 pages. https://doi.org/10.1145/3290605.3300278

[30] M. Tawfik, E. Sancristobal, S. Martin, R. Gil, G. Diaz, A. Colmenar, J. Peire, M. Castro, K. Nilsson, J. Zackrisson, L. Hakansson, and I. Gustavsson. 2013. Virtual Instrument Systems in Reality (VISIR) for Remote Wiring and Measurement of Electronic Circuits on Breadboard. *IEEE Transactions on Learning Technologies* 6, 1 (Jan 2013), 60–72. https://doi.org/10.1109/TLT.2012.20

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

[32] Nicolas Villar, James Scott, Steve Hodges, Kerry Hammil, and Colin Miller. 2012. . NET gadgeteer: a platform for custom devices. In *International Conference on Pervasive Computing*. Springer, 216–233.

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

[34] E. Williams. 2014. *Make: AVR Programming*. Maker Media. https://books.google.co.kr/books?id=jgggmAEACAAJ

[35] Te-Yen Wu, Jun Gong, Teddy Seyed, and Xing-Dong Yang. 2019. Proxino: Enabling Prototyping of Virtual Circuits with Physical Proxies. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 121–132. https://doi.org/10.1145/3332165.3347938

[36] 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* (Qu&#233;bec City, QC, Canada) *(UIST '17)*. ACM, New York, NY, USA, 343–349. https://doi.org/10.1145/3126594.3126646

[37] 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* (Qu&#233;bec City, QC, Canada) *(UIST '17)*. ACM, New York, NY, USA, 311–319. https://doi.org/10.1145/3126594.3126634

[38] Junichi Yamaoka, Mustafa Doga Dogan, Katarina Bulovic, Kazuya Saito, Yoshihiro Kawahara, Yasuaki Kakehi, and Stefanie Mueller. 2019. FoldTronics: Creating 3D Objects with Integrated Electronics Using Foldable Honeycomb Structures. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. ACM, New York, NY, USA, Article 628, 14 pages. https://doi.org/10.1145/3290605.3300858

[39] Junyi Zhu, Lotta-Gili Blumberg, Yunyi Zhu, Martin Nisser, Ethan Levi Carlson, Xin Wen, Jessica Ayeley Shum, Kevin abd Quaye, and Mueller Stefanie. 2020. CurveBoards: Integrating Breadboards into Physical Objects to Prototype Function in the Context of Form. In *Proceedings of CHI '20 (in press) (CHI '20)*.