

Proxino: Enabling Prototyping of Virtual Circuits with Physical Proxies

Te-Yen Wu¹Jun Gong¹Teddy Seyed²Xing-Dong Yang¹¹ Dartmouth College

{te-yen.wu.gr, jun.gong.gr}@dartmouth.edu
xing-dong.yang@dartmouth.edu

² University of Calgary

teddy.seyed@ucalgary.ca

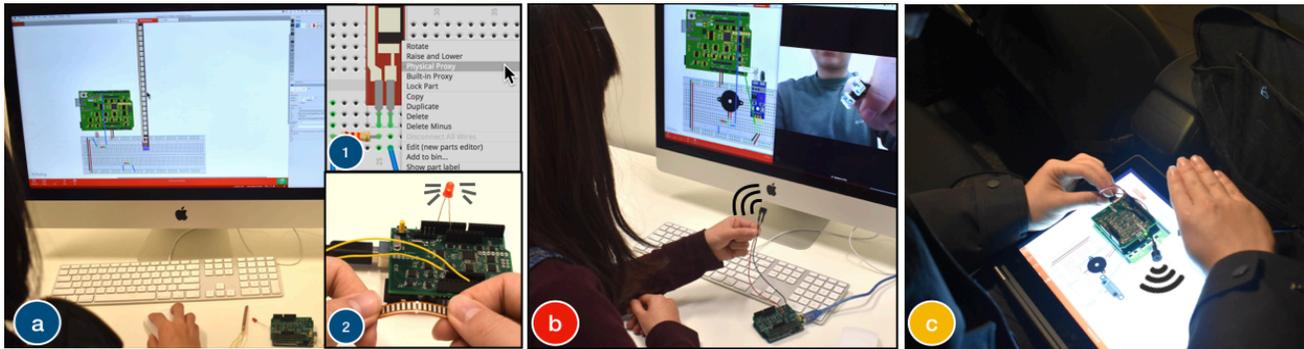


Figure 1. Proxino allows users to (a) interact with virtual circuits using physical proxies (e.g. linking a virtual component to a physical flex sensor and interacting with it), (b) remotely collaborate and share resource with others (e.g. a buzzer can be controlled by a remote IR transmitter and receiver), and (c) prototyping circuits ubiquitously with build-in proxies (e.g. using built-in proximity sensor as the physical proxy of distance sensor).

ABSTRACT

We propose blending the virtual and physical worlds for prototyping circuits using physical proxies. With physical proxies, real-world components (e.g. a motor, or light sensor) can be used with a virtual counterpart for a circuit designed in software. We demonstrate this concept in Proxino, and elucidate the new scenarios it enables for makers, such as remote collaboration with physically distributed electronics components. We compared our hybrid system and its output with designs of real circuits to determine the difference through a system evaluation and observed minimal differences. We then present the results of an informal study with 9 users, where we gathered feedback on the effectiveness of our system in different working conditions (with a desktop, using a mobile, and with a remote collaborator). We conclude by sharing our lessons learned from our system and discuss directions for future research that blend physical and virtual prototyping for electronic circuits.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
UIST '19, October 20–23, 2019, New Orleans, LA, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6816-2/19/10...\$15.00
<https://doi.org/10.1145/3332165.3347938>

Author Keywords

Circuit construction, breadboard, remote collaboration

CSS Concepts

- Human-centered computing → Human computer interaction (HCI) → Interactive systems and tools → User interface toolkits

INTRODUCTION

Software-based tools for circuit prototyping (e.g., Tinkercad Circuits [10]) are increasingly used by novice makers in projects that involve electronics. Typically, these tools include a virtual breadboard to help a user construct and test a circuit. However, circuits designed and created on a virtual breadboard cannot be interacted with physically. This becomes an issue when prototyping interactive artifacts that involve input or output (I/O) components (e.g. sensors and motors), as a user typically needs to interact with the components physically, in a real environment.

To address this challenge, we propose a new breadboard circuit prototyping environment that blends the virtual and physical world, by enabling for the use of real I/O components (e.g. a motor, or flex resistor) as a physical proxy of their virtual counterparts, in a circuit designed and constructed in software. The physical proxies themselves behave as though the entire circuit created virtually is real. Using this approach, a user can then interact with virtual components physically (Figure 1a).

By allowing users to take the advantage of both the virtual and physical world, we create a new environment that

Session 1B: Software and Hardware Development

enables several other scenarios for novice makers, in which I/O circuits can be designed, constructed, and tested. One example scenario is when the user is working on a project but missing a crucial I/O component to test an idea, such as a motion detector for security. The user can now receive help from a remote collaborator with whom they can share their virtually created circuit. The remote collaborator uses a real IR sensor as a remote proxy of the circuit and shows the user how it functions when motion is detected (Figure 1b). This way, the user does not need to have the sensor to test and iterate upon the idea. Essentially, it is now possible for both users to share their physical resources (e.g. sensors and actuators) using remote proxies.

Another example is when a user desires to continue learning a circuit but has no access to their breadboard or physical I/O components (e.g. they are using a tablet or mobile device on a long bus ride). The user can now continue to develop the circuit on a tablet and use the built-in I/O components (e.g. proximity sensor or accelerometer) of the tablet as a proxy for physical interactions. (Figure 1c).

To demonstrate technical feasibility and new possibilities enabled by this new circuit prototyping environment, we developed a tool, called Proxino. The tool is comprised of three parts: (1) frontend software, developed as a part of Fritzing [7], that allows novice users to create virtual breadboard circuits and program them in Arduino, (2) a hardware device in the form of an Arduino UNO shield, that interfaces physical proxies and a virtual breadboard circuit, and (3) a backend server that runs the virtual breadboard circuit in a simulator and handles data synchronization between the shield, the user's Arduino code, and remote nodes during remote collaboration sessions.

The contributions of this paper are: (1) the concept of using real I/O components as a physical proxy to interact with virtual circuits; (2) an exploration of the application space of using a physical proxy; (3) the implementation of a prototype, Proxino; (4) a system evaluation and initial user feedback of our tool; and (5) a set of applications and interactions enabled by Proxino.

RELATED WORK

Hardware Circuit Prototyping Tools

Prototyping interactive artifacts with electronics has become increasingly accessible due to the recent development of open-source hardware platforms (e.g., Arduino [2], Phidgets [19], or Microsoft .NET Gadgeteer [38]). However, novice makers often lack basic knowledge in electronics before they can start building things. Tools that leverage generative design [12] are very useful for the beginners to test their ideas. One challenge with this approach is that (novice) users may forego the opportunity to learn and enjoy the process of designing their own circuits. However, physically constructing electronic circuits on a breadboard have been shown to be prone to many types of errors [15, 32]. A study conducted by Booth, et al. [15] showed that hardware errors are almost always inevitable despite experience level in

UIST '19, October 20–23, 2019, New Orleans, LA, USA

electronics. Common errors reported from their study include miss-wiring, incorrect component, missing components, and bad seating. Projects like CircuitStack [39] can mitigate wiring errors, while many other hardware errors (e.g., bad seating) do not occur in software. As such, software circuit prototyping tools are becoming increasingly popular.

Aside from using a virtual breadboard for prototyping, many other tools have been developed in the past several years to assist in the creation of electronic devices [16, 21-26, 30, 33-35]. For example, with Exemplar [18] and PICL [15], a user can create sensor-based interactive artifacts using “programming by demonstration”. Children can develop electronic hardware using augmented toys like LEGO bricks embedded with sensors and actuators [35]. A major difference between these approaches and breadboard prototyping, is that these tools are all restricted to a specific platform, which means that a user does not have complete freedom in creation (e.g. platform might be missing an accelerometer). In contrast, a breadboard is much more flexible but of course more difficult to use.

Aside from hardware errors, adjusting breadboard components (e.g., different resistors) is also challenging on a physical breadboard. To address this problem, VirtualComponent [27] provides a mixed reality system with a custom breadboard that preloads several possible components into PCB modules, and a user can dynamically place components and see values and potential errors. A similar approach can be found in Scanalog [37], whose goal is to facilitate the design and debugging of analog circuits through a dataflow programming paradigm using functional modules, instead of components and breadboard circuit. LabView [8] is a commercial product, which also uses a dataflow paradigm, but for the design of complex electronic systems. The tool was designed for professional users and requires a specialized hardware setup.

Software Circuit Prototyping Tools

Many commercial software has been developed to facilitate PCB design [1, 14]. However, these professional tools while effective, are not especially tailored for novice makers. As such, software emphasized on breadboard circuit prototyping [10, 11, 28] has gained wide adoption in the novice maker communities. For example, TinkerCad Circuit [10] allows users to construct a breadboard circuit using a virtual breadboard and electronic components. It also offers a simulator for users to test and debug the function of their circuit designs. The software also supports remote collaboration and most noticeably the simulation of the behavior of common input and output components. One key limitation is that a user's design is entirely virtual, and thus cannot be directly interacted with in the physical environment.

Circuit Debugging Tools

The fact that circuit errors are unavoidable motivated a separate line of research and commercial products in supporting circuit debugging. For example, Diligent Electronics Explorer [6] is a breadboard that allows users to

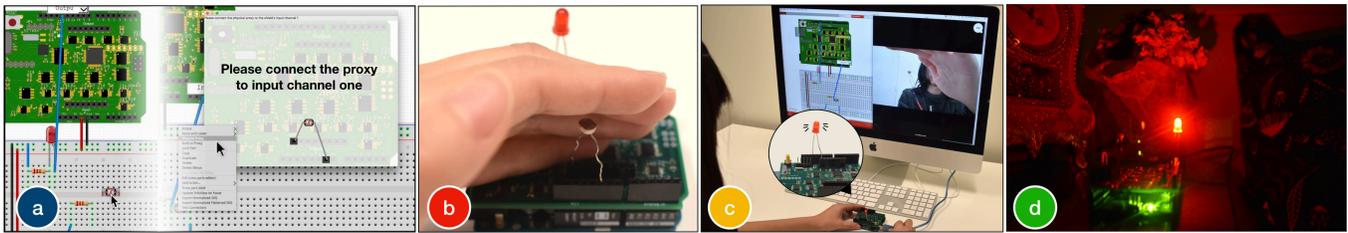


Figure 2. System walkthrough. (a) A user designs a circuit on software and links the photoresistor as a physical proxy, where an instruction window will pop up. (b) The user can test the circuit by physically interacting with the component (c) The user can remotely collaborate with others for help. The circuit in the both sides are connected and working together (d) Finally, the circuit can be directly deployed with a WiFi module.

debug breadboard circuits through built-in oscilloscope, pattern generators, and logic analyzer. The challenge is that novices may not have the background to operate these tools. As a result, an area of research arose focusing on easing the debugging process through the visualization of a circuit’s internal state. Toastboard [17], Bifrost [31], and CurrentViz [40] are examples. A key observation of these works is that sensing the location, type, and value of the breadboard components play a key role in easing the debugging process. Although little research has focused on this space, CircuitSense [41] has shown the feasibility of recognizing some of the common electronic components using a sensor setup that is somewhat bulkier than a regular breadboard.

PHYSICAL VS VIRTUAL CIRCUIT

The benefits of virtual and physical circuits are complementary to each other. Taking advantage of both sides can enable a much more powerful tool. This section provides a brief discussion on some of the benefits of physical vs virtual circuit prototyping.

Physical circuits. Circuit prototyping using a physical breadboard and real electronic components is the default approach for many novice users. The benefit is obvious as the constructed circuit is real and thus can be tried or tested immediately in a real environment. This is important for a number of reasons. First, the behavior of a circuit can be precisely observed and adjusted in the environment, where the circuit components are operating. This is particularly true for circuits involving sensors or actuators. For example, the threshold of a photoresistor can be precisely determined in a real environment where the sensor is being used. Similarly, a user can only tell if the torque of a motor is strong enough for a certain project by testing the real motor itself (without doing all the math and physics). Second, with real input and output components, the finished project can be demonstrated to others. Finally, the project can be deployed right away if needed. This is not possible using software as the user must first manually replicate the design on a physical breadboard.

Virtual breadboards. Circuit prototyping with software has become increasingly popular among novice makers. A user can now use virtual breadboards and circuit components to create their projects. Testing and debugging a finished circuit is also possible with a simulator. Circuit design using software has a number of unique benefits over a physical breadboard. For example, the user does not have to maintain

an inventory of common electronic components that may have varying types and values (e.g., ICs, resistors, or capacitors, etc.). This is a common barrier for novice makers who begin prototyping. Additionally, features like copy/paste, undo/redo, tooltips, autocomplete [29], and remote collaboration [10] have made circuit prototyping through software an efficient and less error-prone experience for novices. Finally, virtual circuits can be easily stored, documented, and shared among different users, groups and even across different locations via the Internet. This is not as easy using a physical breadboard. However, the major drawback of using software is the disconnection between a user’s finished virtual circuit from the physical world. Although many state-of-the-art simulators are effective in replicating circuit behaviors, a user’s interaction with the circuit remains entirely virtual.

PROXINO WALKTHROUGH

To embrace the benefits of physical and virtual circuits, we propose using a physical proxy as the link between the virtual world and physical world. That is, users can augment parts of their virtual circuit with virtual components using real components, when they need to physically interact with components. This section demonstrates a running example to illustrate some of the capabilities of our system Proxino. The example is a Nightlight circuit, which uses a photoresistor to sense the intensity of room light and automatically turns on an LED if the light intensity drops below a threshold value.

Alice is a beginner, who reads an online tutorial of a Nightlight circuit and starts slowly to build it using Proxino’s software. Before she starts, she is worried because she does not have the right type of resistor. However, using the software, she is glad that it allows her to specify a resistor value virtually. She starts to drag a photoresistor, an LED, and a resistor one-by-one into the virtual breadboard. She copy-and-pastes a second resistor for convenience. She sets the resistor value and connects her chosen components one by one, following the tutorial. She uses undo a few times to recover from errors (Figure 2a).

Once the circuit is completed, Alice wants to see how it works in her room. She brings up a context menu on the software by right-clicking the virtual photoresistor. In the menu, she clicks “Physical Proxy” to link the photoresistor to a physical proxy. She then follows the instructions shown in the software to connect her physical photoresistor to the

Session 1B: Software and Hardware Development

desired pins on the Arduino shield (Figure 2a). She repeats the same procedure to setup the LED. Once finished, she codes in the built-in IDE by following the sample Arduino code. She clicks the “Run” button to execute the circuit in a simulator. However, the LED does not turn on when she covers the photoresistor using her hand (Figure 2b).

Alice calls her friend, Derek for help on Skype. Derek is an experienced maker, who accesses Alice’s virtual breadboard remotely from his home computer to help her with debugging (Figure 2c). The circuit seems fine when inspected, so Derek suspects that Alice’s photoresistor or LED is broken. Unfortunately, Alice does not have a second photoresistor. Derek offers to try his photoresistor at his home and uses it as a remote proxy. Alice disconnects her photoresistor from the shield on her side, while Derek connects his on his side. Derek is correct. Alice’s LED turns on as expected, when he covers the photoresistor (Figure 2c).

The next day, Alice purchases a new photoresistor to replace her defective one. Just like a real breadboard circuit, she can deploy the finished project in her room. The system includes the photoresistor, LED, an Arduino and the shield. Alice further uses Arduino Uno WiFi Rev2 for the communication between the proxies and virtual circuit running on her desktop computer. Alice is happy that there is no breadboard, resistors, and messy wires in her room (Figure 2d).

PROXY DESIGN SPACE

We present the design space of using a physical proxy for common circuit components in four dimensions.

A1: Component Type

Different types of electronic components can be used as a proxy for a virtual circuit. To facilitate the discussion, we describe our classification of components into 3 categories based on their roles in supporting physical interactions with a breadboard circuit: (1) *supporting components*, (2) *input components*, and (3) *output components*.

Supporting components are the building blocks used to form functional circuits for input and output components. Common supporting components include integrated circuits (ICs) (e.g., H-bridges and amplifiers) and basic components like resistors, capacitors, inductors, diodes, and transistors, etc. Physical supporting components preserve the behavior of the circuits, and thus may lead to more realistic signals. However, some supporting components have many different types and values. Maintaining an ample inventory of them can be challenging for novice makers, especially if they don’t entirely understand the function of all the components. For our current implementation of the system, we did not support components which are not interactive.

Input components are input devices (e.g., buttons) or sensors (e.g., photoresistor). Many input components need to be used with basic components or ICs to function properly or safely. For example, a PIR sensor needs an amplifier for increased signal amplitude for high-resolution data processing. Input components can be either virtual or physical. For example, a

UIST '19, October 20–23, 2019, New Orleans, LA, USA

user often needs to interact with input components physically to provide input or collect data from the environment. Therefore, a physical proxy can be helpful. Virtual input components are useful in scenarios where physical interaction is not a need, or the required physical components are unavailable (e.g. a user did not purchase one).

Output components are output devices such as LEDs or motors. Similar to input components, many output components also require an appropriate basic component or IC to work properly. A DC motor is an example as it requires a motor driver or transistor to operate with an Arduino. Some output components can benefit more from a physical proxy than others. For example, a real servo motor has a clear advantage over the virtual one if the user wants to test how well it works to actuate a physical object. A virtual LED, on the other hand, can perhaps be as useful as a physical one for the sake of feedback. However, lighting a real LED could give the user a more engaging experience.

A2: Proxy Type

The physical proxy can be a *singleton* or *module*. The singleton has only one I/O component. Many singletons do not function properly on its own without a supporting component(s). A module, on the other hand, is an electrically functional circuit, composed of an I/O component and one or more supporting components. For example, a photoresistor module is composed of a photoresistor (input component) and a resistor (base component), which works independently if powered.

A3: Proxy Location

Virtual circuit prototyping can take place collaboratively among different users either locally or remotely. During remote collaboration, the physical proxy can be *collocated* with the user or located in a *remote* location. This allows users in different locations to have shared access to distributed physical resources only available in a remote site.

A4: Proxy Module Form

I/O modules can go beyond their *regular form* and utilize the built-in I/O devices on a *smartphone*, *tablet*, or *laptop*. For example, the user can develop a virtual circuit on a tablet and use the device’s built-in accelerometer as the physical proxy of the virtual one. This way, the software and the physical proxies of the virtual components are integrated into a single device, largely increasing the mobility of the system.

EXAMPLE APPLICATIONS

We implemented 5 applications to demonstrate each point in the proxy application space through three areas.

Circuit Prototyping Using Software and Physical Proxy

Physical input and output components (A1, A2)

One of the most useful use cases of Proxino is to create the virtual circuit in software and use physical proxies for input and output. Not only does this allow a user to interact with the circuit physically, it also prevents the user from maintaining a large inventory of basic components and ICs in order to learn, practice, and prototype breadboard circuits.

Session 1B: Software and Hardware Development

Additionally, hardware errors, such as loose cabling can be largely avoided by using the software. All of these can potentially make breadboard circuit prototyping more accessible to the novice users. The example with Alice shown earlier demonstrates a use case of this scenario.

Virtual input component and physical output component (A1)

The flexibility of what can be used as a proxy satisfies varying needs for users. If Alice wants to modify her project to be a smoke detector, she can try her new idea by replacing the photoresistor with a gas sensor in the software and adjust the sensor value virtually to see if it works with her current circuit and LED (Figure 3a). This way, she does not need to buy the sensor first before she can test out her idea.

Physical input component and virtual output component (A1)

Alice likes the idea of the smoke detector, but after she purchases the gas sensor, she figures that she wants more than just an LED. She tries out different output components in the software and finds that the buzzer, which makes a noise through her computer's speaker as an alert, is a better option for her instead.

Remote Collaboration and Resource Sharing (A3)

Using the software allows two remote users to collaborate on a project via a shared virtual breadboard. With the physical proxy, the benefit of Proxino goes beyond remote collaboration because the users now have a shared access to distributed physical resources, such as I/O component and environment only available in a remote site. In the scenario with Alice, she does not have to travel to Derek's home to use the photoresistor. It is also possible for the users to access the physical environment of a *remote* site. For example, a user can test how well his/her light flicker detector work in a remote collaborator's home by having a photodiode on the collaborator's side as a proxy without having to travel to a location with a flickering ceiling light (Figure 3b).

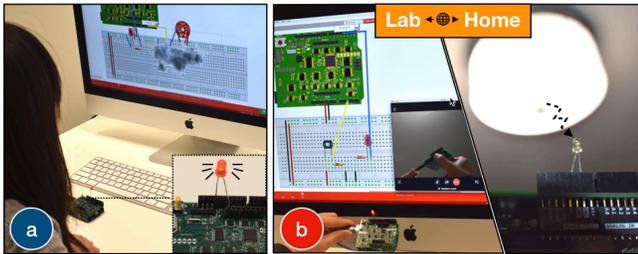


Figure 3. The applications of Proxino. (a) a user drags virtual gas to adjust the value of a gas sensor to test if it works with a physical led. (b) a user test how well the light flicker detector works at a remote collaborator's home.

Ubiquitous Circuit Prototyping (A4)

With Proxino, circuit prototyping is more accessible in *mobile* scenarios as a user only needs to carry the Proxino hardware and I/O components. Circuit can be designed on a tablet. This way the user can try their ideas anywhere, and it can be useful for test the projects in the real environments, where the circuits will be deployed.

UIST '19, October 20–23, 2019, New Orleans, LA, USA

In many situations, the built-in I/O devices in mobile devices may also be enough for quickly testing ideas when inspiration strikes. This way, the proxy is not in its *regular form*, allowing further reduction of the need to carry I/O components. For example, the user can use the built-in proximity sensor of the tablet as a proxy to simulate a HC-SR04 ultrasonic distance sensor and quickly test an idea similar to the Distance Alarm System [4] where a buzzer rings upon the user's hand covering the sensor. A remote collaborator in a stationary environment can take the lead on typing Arduino code because doing so on a mobile device can be challenging.

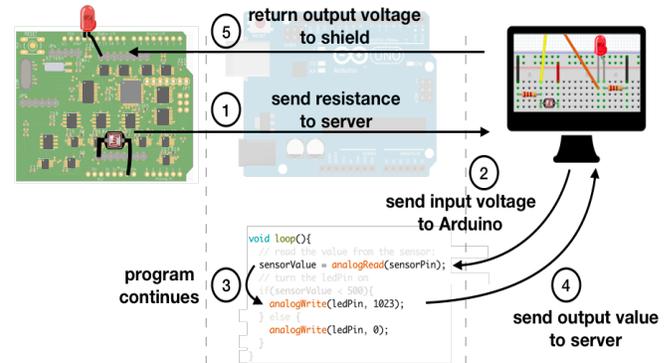


Figure 4. The system workflow.

PROXINO UNDER THE HOOD

One area not described in the example with Alice is programming. Typically, when a user starts an electronics project from scratch, they need to write code (in our case Arduino code) to read or write to I/O components that enable controlling the behavior of the circuit. With Proxino, the construction of a real circuit is not required by a user. Instead, the system uses a circuit simulator. Therefore, Arduino program's read or write functions (e.g., `analogRead()` or `analogWrite()`) now communicate with the simulator. The process can be described as the following:

- When a user constructs a virtual circuit in Fritzing, the circuit is also created in the simulator by the system.
- The Arduino shield takes the real-time input from the input proxy by measuring its analog value (e.g., voltage, capacitance, and resistance).
- It then sends data to the simulator running in the background of a laptop computer via the shield's hosting Arduino.
- The data is used along with the mathematical model of the user's virtual circuit by the simulator to recover the component's input voltage. Using Alice's photoresistor circuit as an example, our hardware captures the changes in the resistance of the photoresistor through an input pin of the shield. It passes the data to the simulator, which then recovers the input voltage of the photoresistor using the voltage divider equation.
- If a read function in the user's program is called, the input voltage is returned to the program. For example, when the `analogRead()` is executed, the system converts the current

Session 1B: Software and Hardware Development

input voltage into an integer value between 0 and 1023 to overwrite the return value of the `analogRead()`.

Output uses a similar approach, as when a write function is executed, our system generates PWM signals based on the user's specification of duty cycle in `analogWrite()`, which was then plugged into the simulator to calculate the right peak value for the shield to drive the proxy. Figure 4 demonstrates the system workflow.

PROXINO SHIELD

Our custom shield was developed to handle I/O proxies that are both a singleton or a module. Many singletons do not function properly on their own, thus measuring the analog signals from them can be difficult. For example, the resistance of a photoresistor cannot be measured unless inside a module, where a resistor is used with the photoresistor to form a voltage divider. Similarly, a DC motor does not work without an actual motor driver. The shield handles these situations using a *data acquisition circuit* and a *proxy driver circuit*. Depending on the type of input proxy, the data acquisition circuit forms a voltage divider, RC, or pull-down circuit, allowing the device to measure input voltage, resistance, capacitance, or switch status for varying applications for novice makers. The proxy driver circuit can generate PWM signals in a frequency of 500Hz at a proper duty cycle and peak value to drive different types of output singleton.

A small number of I/O singletons are self-functional without the need to be inside a module with supporting components. Examples include the temperature sensor, proximity sensor, and servo motor. Therefore, getting input or output to them is straightforward. The user can even use an Arduino directly. Modules are similar, but in cases when a module proxy is incomplete (e.g., missing a resistor), its I/O component is treated as a singleton so using the shield is necessary. This is the easiest way to allow us to properly interact with the corresponding I/O component without significantly increasing the complexity of the shield.

SOFTWARE IMPLEMENTATION

Our software implementation includes (1) a front-end breadboard circuit design and programming tool and (2) a back-end server handling circuit simulation, data synchronization between the simulator and a user's Arduino code, and remote collaboration.

Front-end

Virtual circuit design environment

We implemented our virtual circuit design interface as a part of Fritzing [7], a circuit prototyping tool common in maker community. Our tool also provides a user with options to interact with virtual I/O components through physical proxies in the form of either an electronic component or the built-in sensors or actuators of a mobile device. The software also allows a user to manually adjust the value of an input component (e.g., resistance of a flex resistor) to see how the constructed circuit behaves accordingly. This is useful for testing and debugging the circuit virtually. Remote

UIST '19, October 20–23, 2019, New Orleans, LA, USA

collaboration is supported by enabling multiple users from different locations to construct a circuit at the same time, using a shared virtual breadboard.

Arduino programming environment

Users can develop an Arduino program in Fritzing's built-in Arduino IDE. Despite the use of a virtual circuit and physical proxies, the user programs the circuit as though it is real. From the system's perspective however, it is important to synchronize the execution of the user's code, especially for input functions like `analogRead()`, as the simulator runs slower than the code. Again, with the photoresistor example, nothing beyond `analogRead()` should be executed until after the simulator calculates the input voltage of the photoresistor. Output is similar, as the execution of `analogWrite()` should be paused until the output PWM signal for the LED proxy arrives from the simulator. Our system achieves this by injecting our custom code into the user's program. For example, we replace the calls to `analogRead()` and `analogWrite()` with our custom functions, that pause the user's code until data from the simulator arrives (Figure 5). Our system modifies the user's code only after the user clicks the "Run" button, prior to the code being compiled.

```
void loop(){
+ myLoop();
// read the value from the sensor:
- sensorValue = analogRead analogRead_proxino(sensorPin);
// turn the ledPin on
+ if(sensorValue < 500){
- analogWrite(analogWrite analogWrite_proxino(ledPin, 1023));
+ } else {
- analogWrite(analogWrite analogWrite_proxino(ledPin, 0));
+ }
}

+ int analogRead_proxino(int pin){
+   int value = analogRead(pin);
+   if (pin_data_type[pin] == PIN_DATA_TYPE_INPUT_FROM_SERVER){
+     if (pin_status[pin] == 1){
+       wait_until_data_updated(pin);
+     }
+   }
+ }
```

Figure 5. Our system modifies the user's program automatically by replacing `analogRead()` by a custom function, called `analogRead_proxino()`, to pause the user's code until data from the simulator arrives.

Note that our system relies on continuous communication between the simulator running on the hosting computer and the program running on the Arduino. Some applications may require interrupts (e.g. such as the communication to a computer) be disabled temporarily while reading data from an input component. For example, in programming a capacitive sensor to count the CPU cycles required to pull up a sensor pin, it is often necessary to have all interrupts disabled (e.g., by calling `noInterrupts()`) while reading sensor data. This may cause a system failure in our case, as Proxino relies on the communication between an Arduino and the simulator to exchange data for input and output. Our solution to this problem is to run interrupt-sensitive logic on the Proxino backend server instead of on the Arduino board and have the result to be returned to the board. For example, the number of CPU cycles it takes for a sensor pin to be pulled up can be estimated by running the counting logic on the hosting computer, as the processor cycle of the user's program is known. From a user's perspective, they do not make any changes in their program.

Session 1B: Software and Hardware Development

Backend

Circuit simulator

The circuit simulator was implemented using LiveSpice [9], an open source circuit simulation tool for analog circuits. We modified its source code to allow the tool to take real-time input data from physical proxies and calculate output voltage values based on a user's interaction with a proxy. Since LiveSpice does not provide the mathematical models for the common I/O components, we selectively implemented several of them, such as LEDs, hobby motors, vibrating motors, and buzzers.

Network Server

We developed a custom server to handle communication for remote collaboration. The server takes all changes that occur on remote nodes, such as user interactions with remote proxies or modifications on the shared circuit. It communicates these changes to the simulator and reports the results back to remote clients. The server is written in Node.js and communicates with each client using Socket.IO.

HARDWARE IMPLEMENTATION

Our shield features a total of 16 I/O pins for interacting with proxies. Half are used for input components and the remaining half are for output components. The shield uses pins 8 and 9 to communicate with its hosting Arduino board. It also provides pin accesses to all other pins on the Arduino Uno board (Figure 6). This is useful in the cases when the user wants to use Arduino directly, like Alice's Wi-Fi example.

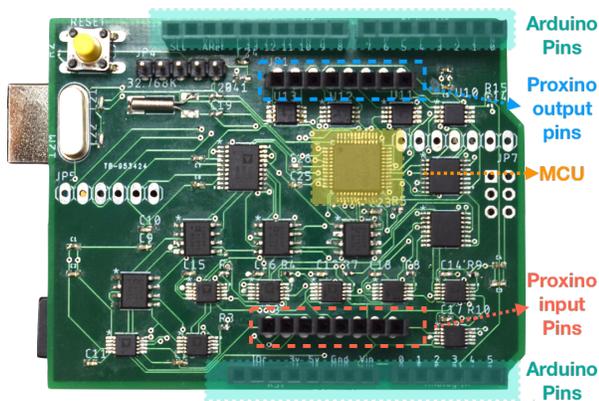


Figure 6. Proxino PCB in the form of an Arduino shield.

We built the shield around a Nuvoton ARM Cortex-M4 microcontroller with a data acquisition circuit and proxy driver circuit. The *data acquisition circuit* is composed of an 8-channel digital-to-analog converter (DAC) (AD5628BRUZ, Analog Device inc.), a digital potentiometer (AD5270BRMZ-100, Analog Device inc.), an 8-channel analog-to-digital converters (ADC) (AD7928BRUZ, Analog Device inc.), and an amplifier (AD8066ARZ, Analog Device inc.) serving as a voltage buffer for the ADC (Figure 7 left). The *proxy driver circuit* is composed of an 8-channel DAC and eight high-output-drive amplifier (TLV4112, Texas Instruments inc.), connected to the Arduino's VIN pin for extra current supply (Figure 7 right). This circuit can

UIST '19, October 20–23, 2019, New Orleans, LA, USA

generate output voltage up to 6V and current up to 500 mA, needed for different types of motors. The shield is compatible with an Arduino UNO and communicates with it using the UART protocol with a baud rate limit of 115200 bps.

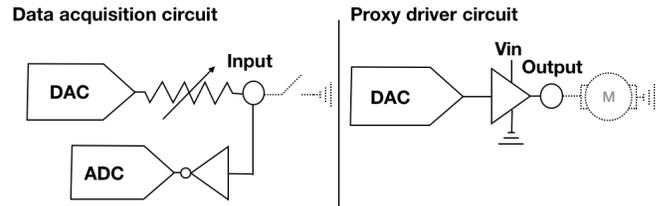


Figure 7. The left circuit is a data acquisition circuit, used for input components. The right circuit is a proxy driver circuit designed for output components.

SYSTEM EVALUATION

The goal of this experiment was to measure the difference between the output of our system versus that of a real circuit. We included four common analog input signals that included resistance, capacitance, voltage, and switch status. We also tested the PWM output of our system with and without being processed by an IC (e.g., a motor driver).

Analog Input Circuits

We describe our tested circuits for the 4 analog input signals:

Resistance: Comprised of the potentiometer and a resistor of $10\text{K } \Omega$ connected in series. Examining the Arduino Starter Kit [5], we found that this setup is representative of circuits for commonly used input components based on resistance, such as the photoresistor, the flex resistor, the force sensor, or the slide potentiometer. We used a digital potentiometer to produce different levels of input resistance.

Voltage: Comprised of a function generator and a resistor of $100\text{K } \Omega$ connected in series. This setup can also be found for many of the common input components based on voltage, including the temperature sensor, the proximity infrared sensor, and a piezo sensor. We used a function generator to produce input voltages of varying amounts.

Capacitance: Includes a variable capacitor and a resistor of $1\text{M } \Omega$ connected in series. This setup represents a circuit typically used for capacitive sensing. We used a variable capacitor to produce different levels of capacitance.

Switch: Comprised of a push button and a $1\text{K } \Omega$ resistor connected in series.

Analog Output Circuits

We compared the PWM output of our system with the original output from an Arduino UNO. We generated the PWM signals of different duty cycles in a circuit containing a resistor of 100Ω connected with an LED in series. Aside from the LED itself, this type of circuit is commonly seen in many other entry level projects to drive output components, like a buzzer, or a vibration motor. Additionally, we included in our experiment another common type of circuit composed of an output component and an IC, and we used a PNP transistor to drive a DC Hobby Motor 130.

Session 1B: Software and Hardware Development

Data Collection 4

Data collection for both input and output was carried out using the tested circuits on a real breadboard with an Arduino Uno, and also on a virtual breadboard with Proxino, where the behaviors of the circuits were simulated.

Input

For input, the ground truth data includes the resistance values ranging from 100 Ω to 100K Ω with an interval of 1024 Ω , voltage values ranging from 0 V to 5 V with an interval of 100 mV, and capacitance values ranging from 100 p to 470 p with an interval of 4 p. The switch status was produced by clicking the push button 10 times. We also included in the ground truth, the corresponding return values from Arduino's `analogRead()` based on the physical circuits. Note that Arduino's capacitance readings were recorded using Arduino's Capacitive Sensing Library [3].

For the testing data, we recorded the actual resistance, voltage, capacitance, and switch status measured by the shield. We also recorded the corresponding return values for `analogRead()` calculated by the simulator based on the virtual representation of the tested circuit. We were interested in knowing how the value of `analogRead()` generated by Proxino and Arduino UNO differ comparatively.

Output

For the ground truth data, we generated 500 Hz PWM signals ranging from 0 to 100% duty cycles at a 4% interval using Arduino's `analogWrite()`. Next, the corresponding PWM at the output pins of the Arduino UNO was recorded as ground truth for the tested circuits.

For the testing data, we recorded the PWM signals calculated by the simulator, before the signals were passed to the shield. Additionally, we recorded the corresponding PWM signals captured at the output pins of the shield. We were interested in knowing how the output PWM signals generated by the simulator differ from the real ones, and how the output PWM captured at the shield differ from the real ones. We compared the PWM signals by their duty cycle and amplitude.

Result

Analog Input

We report, in the left column of Table 1, the measurement error of Proxino's shield versus the ground truth. Additionally, we report the mean difference in the return value of `analogRead()` calculated by our system versus that of a real circuit using the Arduino UNO.

For the resistance, the measurement error of our shield was 5.92%. This is acceptable as it is close to a resistor's 5% tolerance range. The value of `analogRead()` generated by Proxino and Arduino UNO differed by 17 units. This difference was mainly caused by the measurement error from both Arduino UNO and our shield.

The measurement error of our shield in capacitance was 12.9%, which is again acceptable as it is within the 20% tolerance range of the capacitor. This amount of error could be negligible for capacitive sensing. The average return value

UIST '19, October 20–23, 2019, New Orleans, LA, USA

of Proxino and Arduino's touch library differed by 1082 units out of a total range of 65535 units. In turn, the return value of `analogRead()` generated by Proxino and Arduino UNO is deferred by 6 and 3 units respectively. These differences can be negligible for many of the entry level applications and circuits for novice makers.

Input Type	Error (%)	Avg. diff in Input Value
Resistance	5.9%	17 (0-1204)
Capacitance	12.9%	1082 (0-65535)
Voltage	2.4%	6 (0-1024)
Switch	0%	3 (0-1024)

Table 1. Error rate of Proxino and difference in the return value of `analogRead()` calculated by Proxino versus the Arduino Uno.

Analog Output

Table 2 summarizes the difference between the PWM signals of an Arduino UNO and those calculated by our simulator or captured at our shield. In general, the difference between the real and simulated signals are relatively small. Our shield was also able to preserve the signals from the simulator with a small amount of error in both duty cycle and amplitude. Overall, we expect that this amount of system errors would not significantly impact user experience in prototyping entry level circuits.

We measured the data transmission rate and the delay caused by data transmission and simulation. The data transmission rate of Proxino was 3 k/s, which in our implementation was bound to the serial port communication between the shield and Arduino. The delay caused by data transmission was 5 ms between the shield and our software running on a 2015 iMac. The delay caused by the circuit simulation was measured at 19 ms and 11 ms for the LED and motor circuit respectively. We expect the delay to increase with the increase of circuit complexity and network traffic.

Circuit	Diff. in PWM Duty Cycle		Diff. in PWM Amplitude	
	Simulator	Shield	Simulator	Shield
LED	0.9%	1.2%	49 mV	55 mV
Motor	1.3%	1.7%	200 mV	208 mV

Table 2. Difference in the duty cycle and peak value of the PWM signal per tested circuit.

INFORMAL USER EVALUATION

To solicit initial user feedback of Proxino, we conducted an informal user study by asking participants to create the Nightlight circuit in (1) a desktop environment, (2) a mobile environment, and (3) with a remote collaborator.

Participants

Nine participants (3 female) between the ages of 18 and 26 participated in the study. All were novice makers with limited experiences in prototyping breadboard circuits.

Session 1B: Software and Hardware Development

Apparatus

The study apparatus included Proxino along with an LED and photoresistor, serving as proxies. Participants created the circuit on a Lenovo Yoga X laptop, which was switched to tablet mode in a mobile condition in a car. The built-in ambient light sensor of the laptop was used as the proxy of the photoresistor in the mobile condition. Participants used Skype to communicate with the collaborator in the remote collaboration condition.

Task and Procedure

Participants were then asked to construct the Nightlight circuit using Proxino in three conditions: (1) on a desk in an office alone, (2) on the desk working with an expert collaborator from a remote site, and (3) as a passenger in a car. In all conditions, participants created the circuit using the software and were encouraged to try the LED and photoresistor as proxies. In the remote collaboration condition, participants constructed the circuit together with a collaborator (with 5-years of circuit prototyping experience) and were encouraged to try the proxies on both sides. In the mobile condition, participants were given the laptop in tablet mode and were encouraged to use the device's ambient light sensor as a proxy for the photoresistor (Figure 8). Note that readings from the tablet's ambient light sensor given from the Windows Sensor API is converted into the resistance value of the photoresistor (proxy) based on its datasheet, which is then treated as input data from the proxy. Thus, the behavior of the built-in ambient light sensor is the same as the photoresistor. The three conditions were counter-balanced among participants. After the study, participants reflected on their experience of using Proxino, the three scenarios, and how their experience differed from using only software or a physical breadboard.

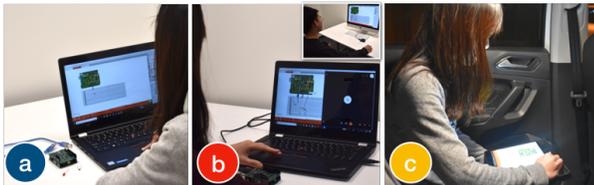


Figure 8. The three conditions of the informal study. (a) on a desk in an office (b) on the desk working with an expert collaborator from a remote site (c) as a passenger in a car

Result

All participants completed the nightlight prototype successfully in all three conditions. Through the interviews, we were able better understand their experiences and identify potential usability issues in our current implementation.

From a software perspective, participants stated that circuit prototyping using our software allowed them to construct and test their circuits quicker and less error-prone than using a physical breadboard. From the hardware perspective, participants mentioned that the most enjoyable feature of Proxino was that they were able to use much fewer physical components and jumper wires to construct the same circuit while still being able to interact with it physically. This allowed our participants to be more focused on the design of

UIST '19, October 20–23, 2019, New Orleans, LA, USA

circuit functions rather than cumbersome logistics, such as cutting wires or fixing loose pin connections.

For example, a participant told us that *“When I work on a real breadboard, I usually have to spend much of my time cutting wires into proper lengths or identifying the right type of resistors or other components. This is tedious. But Proxino allowed me to focus more on getting the right design of my circuit functions. For example, I had more time to try out resistors of different values. I knew I could use software before, but it was not a true option because I was unable to test my circuit real. Proxino gave me the best from both sides and I think it is really amazing.”* (P2).

Our study results also provided useful insights into the usability issues that are uniquely related to the introduction of a physical proxy into software circuit prototyping processes. For example, debugging a virtual circuit could now be more complicated in a blended physical and virtual environment. A participant told us that *“it was difficult for me to locate a problem I had because I was unsure about whether the cause was a defective photoresistor or LED, or something went wrong in my circuit”* (P1 and P5). We believe a better design in this case is to program the virtual photoresistor or LED to react upon the function of the circuit, even if a proxy was not used. However, in a broader sense, debugging tools combining both hardware and software, such as ToastBoard[13] and CurrentViz[36], should be considered as a part of Proxino to allow users to better understand how the virtual and physical components work together in their prototyped circuit.

In the remote collaboration condition, participants stated they were excited that *“collaborating on a physical circuit now becomes possible”* (P2, P3, P4, P5, and P7) and about *“what the remote component can bring to the table”* (P1, P5 and P7). What is missing in the current implementation but can be improved in the future is to allow collaborators to maintain a constant awareness of the status of the remote proxy. For example, participants stated *“I could not always see my collaborator's LED or how they interactive with the photoresistor”* (P1). *“Sometimes I do not feel that I have enough control over the remote proxy”* (P7). This indicates an interesting direction for future research as the issues of remote awareness in a broader sense is an open problem in CSCW and now the collaborative circuit prototyping and physical proxy have brought forth new challenges to solve.

In the mobile condition, all of our participants stated they saw themselves prototyping a circuit in mobile scenarios. They found it handy to use the tablet's built-in sensors as a proxy. For example, a participant said *“I like the idea of using the built-in sensor as a proxy. Now I can simply take my circuit with me and show it to my colleagues on a different floor without having to carry the photoresistor and Proxino device”* (P1). Participants commented that *“I do have new ideas from time to time especially during my bus ride to school, this tool is helpful in the sense that I can test my idea anywhere I want”* (P4, P7). The same group of

Session 1B: Software and Hardware Development

participants told us that they hoped “*my device can have more of such built-in proxies*” (P4, P7). It is not surprising that the existence of the I/O device in a mobile device limits what the users can do in mobile without bringing the proxies. A participant (P7) was concerned about the mobile device’s built-in sensors may not behave the same way as the physical component, and that they may have to change their Arduino code (e.g., threshold value) when switching back to a real component. A solution to mitigate this problem is to calibrate the behavior of components in software. In our previous example with Alice, the output readings of the ambient light sensor of the tablet and that of the photoresistor can be calibrated to match each other.

DISCUSSION AND FUTURE WORK

We discuss the limitations of our current implementation and directions for future research.

Kernel Options. We considered two options when implementing this feature. The first option was similar to the approach described in VirtualComponent [27], where the system employs a pre-defined set of physical components preinstalled for future use. The benefit of this approach is clear, as the constructed circuits behave as though they are real. However, the drawbacks appear to outweigh the benefits. For example, the system may fail to work if a required component is unavailable. Additionally, maintaining a library of extra components increases the size of the device, making it inconvenient in mobile situations. Finally, when deploying a completed circuit, it can be challenging for a user who will have to give up the entire preinstalled package. These problems can be mitigated to some extent by using a centralized resource management approach like those used in remote electronics laboratories [13, 20, 36], but managing such a system can be pricey.

The second approach (which we utilized) uses software to simulate circuit behavior. Software simulation is already widely used in industry and can be very precise in reproducing circuit behaviors. However, the challenge is software cannot completely replicate the real world. Phenomenon such as environmental noises or conditions cannot be fully reproduced by the software. We see this as a limitation, but not a significant drawback for a system designed for novice makers. Note that computational delay can be an issue for real-time circuit execution. Therefore, circuits need to be computationally “light” at present, but we see it as an issue solved with faster computers. In general, using a simulator has benefits in scalability, portability, and deployability in comparison to the first approach.

Hardware capability. The current implementation of our hardware does not support applications that require high-frequency signals, such as antennas or microphones. This is due to the bottleneck in the transmission rate between Arduino and our shield. This limits the applications from those involving audio I/O. This issue can be less pronounced on a different hardware platform, such as an Arduino Mega, or when using a different data transformation method to directly communicate with PC, such as USB or WiFi.

UIST '19, October 20–23, 2019, New Orleans, LA, USA

Our current hardware implementation also does not support components that require precise measurements of the input voltage signal (e.g., a strain gauge) because many of these components require specific circuits, such as a Wheatstone bridge circuit or differential amplifier circuit, to amplify the input signal. Further, our system does not support components that require a current larger than 500 mA (e.g., some models of servos, DC motors, solenoids, speakers). This is due to the cap of the current electric current supply, which is increasable with a higher output amplifier. Our future work will address these challenges with new data acquisition and proxy driver circuit functions. The new circuit board will also be redesigned carefully to ensure the compact size of the new device.

Finally, our system does not support the basic components (e.g., resistors) or ICs to be used as a physical proxy. This will also be added in our future iteration as we understand that allowing the supporting components to be used as a physical proxy enables many new applications, especially for educational purposes. For example, it may be needed by a novice user to learn how to create and fine-tune a real amplifier circuit in a noisy real-world environment.

Ubiquitous circuit prototyping. We believe that mobile and remote collaboration has a great potential to improve the experience of circuit prototyping by changing the way how breadboard circuits are constructed, tested, and shared among the users. However, a ton of research has to be done to make it happen in the future. The current user experience of remote collaboration is still quite limited. Our immediate next step is to better integrate the video conference system into our software so that the users will not have to frequently switch between the breadboard view and camera view. We will also explore new ways that can give the user a better control over a remote proxy, thus to improve experience.

CONCLUSION

In this paper, we propose a new circuit prototyping environment that allows users to extend a virtual circuit to the physical world using proxies. We explored the usability of physical proxies and developed a system to support the concept. Our system included software, which has a frontend interface and a backend server, and hardware, created in the form of an Arduino shield. To demonstrate our approach and its benefits for novice makers, we implemented several applications such as remote collaboration and ubiquitous circuit prototyping. Finally, through a system evaluation and informal user study for initial feedback, we demonstrate the effectiveness of using physical proxies for virtual circuits. We share the insights from our system and envision this work as motivation for future research into blending the physical and virtual worlds of prototyping.

Session 1B: Software and Hardware Development

REFERENCES

- [1] Altium Designer 17 Overview. <http://www.altium.com/altium-designer/>. Accessed in 2019.
- [2] Arduino. <http://arduino.cc>. Accessed in 2019.
- [3] Arduino Capacitive Sensing Library. <https://playground.arduino.cc/Main/CapacitiveSensor>. Accessed in 2019.
- [4] Arduino Distance Alarm. https://create.arduino.cc/projecthub/darwindelacruz/distance-alarm-system-0ed9e5?ref=search&ref_id=distance%20sensor&offset=2. Accessed in 2019.
- [5] Arduino Starter Kit. <https://store.arduino.cc/usa/arduino-starter-kit>. Accessed in 2019.
- [6] Digilent Electronics Explorer. <https://store.digilentinc.com/electronics-explorer-all-in-one-usb-oscilloscope-multimeter-workstation/>. Accessed in 2019.
- [7] Fritzing Software. <http://fritzing.org/home/>. Accessed in 2019.
- [8] LabView. <http://www.ni.com/en-us/shop/labview.html>. Accessed in 2019.
- [9] LiveSPICE. <http://www.livespice.org>. Accessed in 2019.
- [10] TinkerCad Circuit. <https://www.tinkercad.com/circuits>. Accessed in 2019.
- [11] VirtualBreadboard <http://www.virtualbreadboard.com/>.
- [12] 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, 331-342 DOI: <http://doi.acm.org/10.1145/3126594.3126637>
- [13] Johnson A Asumadu, R Tanner, J Fitzmaurice, M Kelly, H Ogunleye, J Belter and Song Chin Koh. 2003. A Web-based hands-on real-time electrical and electronics remote wiring and measurement laboratory (RwmLAB) instrument. In *Proceedings of the 20th IEEE Instrumentation Technology Conference (Cat. No. 03CH37412)*, IEEE, 1032-1035. DOI: <https://doi.org/10.1109/IMTC.2003.1207909>
- [14] EAGLE PCB Design and Schematic Software. 2017
- [15] Tracey Booth, Simone Stumpf, Jon Bird and Sara Jones. 2016. Crossed wires: Investigating the problems of end-user developers in a physical computing task. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI'16)*, ACM, 3485-3497. DOI: <https://doi.org/10.1145/2858036.2858533>
- [16] Kayla DesPortes, Aditya Anupam, Neeti Pathak and Betsy DiSalvo. 2016. BitBlox: a redesign of the breadboard. In *Proceedings of the 15th International Conference on Interaction Design and Children (IDC'16)*, ACM, 255-261. DOI: <https://doi.org/10.1145/2930674.2930708>
- [17] 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, 677-686. DOI: <https://doi.org/10.1145/2984511.2984566>
- [18] Adam Fourney and Michael Terry. 2012. PICL: portable in-circuit learner. In *Proceedings of the 25th annual ACM symposium on User interface software and technology (UIST'12)*, ACM, 569-578. DOI: <https://doi.org/10.1145/2380116.2380188>
- [19] Saul Greenberg and Chester Fitchett. 2001. Phidgets: easy development of physical interfaces through physical widgets. In *Proceedings of the 14th annual ACM symposium on User interface software and technology (UIST'01)*, ACM, 209-218. DOI: <https://doi.org/10.1145/502348.502388>
- [20] Ingvar Gustavsson, Thomas Olsson, Henrik Åkesson, Johan Zackrisson and Lars Håkansson. 2005. A remote electronics laboratory for physical experiments using virtual breadboards. In *Proceedings of the 2005 ASEE Annual Conference*, 12-15.
- [21] Björn Hartmann, Leith Abdulla, Manas Mittal and Scott R Klemmer. 2007. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI'07)*, ACM, 145-154. DOI: <https://doi.org/10.1145/1240624.1240646>
- [22] Björn Hartmann, Scott R Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher and Jennifer Gee. 2006. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the 19th annual ACM symposium on User interface software and technology (UIST'06)*, ACM, 299-308. DOI: <https://doi.org/10.1145/1166253.1166300>
- [23] 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 (CHI'14)*, ACM, 1743-1746. DOI: <https://doi.org/10.1145/2556288.2557150>
- [24] Steven Houben, Connie Golsteijn, Sarah Gallacher, Rose Johnson, Saskia Bakker, Nicolai Marquardt, Licia Capra and Yvonne Rogers. 2016. Physikit: Data engagement through physical ambient visualizations in the home. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI'16)*, ACM, 1608-1619. DOI: <https://doi.org/10.1145/2858036.2858059>
- [25] 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 (UbiComp'13)*, ACM, 363-372. DOI: <https://doi.org/10.1145/2493432.2493486>
- [26] Majeed Kazemitabaar, Jason McPeak, Alexander Jiao, Liang He, Thomas Outing and Jon E Froehlich. 2017.

UIST '19, October 20–23, 2019, New Orleans, LA, USA

Session 1B: Software and Hardware Development

- Makerwear: A tangible approach to interactive wearable creation for children. In *Proceedings of the 2017 CHI conference on human factors in computing systems* (CHI'17), ACM, 133-145.
DOI: <https://doi.org/10.1145/3025453.3025887>
- [27] 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* (CHI'19).
DOI: <https://doi.org/10.1145/3290605.3300407>
- [28] 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, 351-358.
DOI: <https://doi.org/10.1145/1517664.1517735>
- [29] 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* (CHI'19). DOI: <https://doi.org/10.1145/3290605.3300633>
- [30] Joanne Lo, Cesar Torres, Isabel Yang, Jasper O'Leary, Danny Kaufman, Wilmot Li, Mira Dontcheva and Eric Paulos. 2016. Aesthetic Electronics: Designing, Sketching, and Fabricating Circuits Through Digital Exploration. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (UIST'16), ACM, 665-676.
DOI: <https://doi.org/10.1145/2984511.2984579>
- [31] Will McGrath, Daniel Drew, Jeremy Warner, Majeed Kazemitabaar, Mitchell Karchemsky, David Mellis and Björn Hartmann. 2017. Bifrost: Visualizing and Checking Behavior of Embedded Systems across Hardware and Software. in *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (UIST'17), ACM, 299-310. DOI: <https://doi.org/10.1145/2901790.2901833>
- [32] David A Mellis, Leah Buechley, Mitchel Resnick and Björn Hartmann. 2016. Engaging amateurs in the design, fabrication, and assembly of electronic devices. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems* (DIS'16), ACM, 1270-1281.
DOI: <https://doi.org/10.1145/2901790.2901833>
- [33] Raf Ramakers, Fraser Anderson, Tovi Grossman and George Fitzmaurice. 2016. Retrofab: A design tool for retrofitting physical interfaces using actuators, sensors and 3d printing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (CHI'16), ACM, 409-419. DOI: <https://doi.org/10.1145/2858036.2858485>
- [34] 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* (CHI'15), ACM, 2457-2466.
DOI: <https://doi.org/10.1145/2702123.2702487>
- [35] Mitchel Resnick, Fred Martin, Randy Sargent and Brian Silverman. 1996. Programmable bricks: Toys to think with. *IBM Systems journal*, 35 (3.4). 443-452.
DOI: <http://dx.doi.org/10.1147/sj.353.0443>
- [36] Nuno Sousa, Gustavo R Alves and Manuel G Gericota. 2010. An integrated reusable remote laboratory to complement electronics teaching. *IEEE Transactions on learning technologies*. 265-271.
DOI: <https://doi.org/10.1109/TLT.2009.51>
- [37] 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* (UIST'17), ACM, 321-330. DOI: <http://doi.acm.org/10.1145/3126594.3126618>
- [38] 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.
DOI: https://doi.org/10.1007/978-3-642-31205-2_14
- [39] Chiuang 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, 687-695. DOI: <http://doi.acm.org/10.1145/2984511.2984527>
- [40] Te-Yen Wu, Hao-Ping Shen, Yu-Chian Wu, Yu-An Chen, Pin-Sung Ku, Ming-Wei Hsu, Jun-You Liu, Yu-Chih Lin and Mike Y Chen. 2017. CurrentViz: Sensing and Visualizing Electric Current Flows of Breadboarded Circuits. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (UIST'17), ACM, 343-349.
DOI: <http://doi.acm.org/10.1145/3126594.3126646>
- [41] Te-Yen Wu, Bryan Wang, Jiun-Yu Lee, Hao-Ping Shen, Yu-Chian Wu, Yu-An Chen, Pin-sung Ku, Ming-Wei Hsu, Yu-Chih Lin and Mike Y Chen. 2017. CircuitSense: Automatic Sensing of Physical Circuits and Generation of Virtual Circuits to Support Software Tools. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (UIST'17), ACM, 311-319. DOI: <http://doi.acm.org/10.1145/3126594.3126634>

UIST '19, October 20–23, 2019, New Orleans, LA, USA