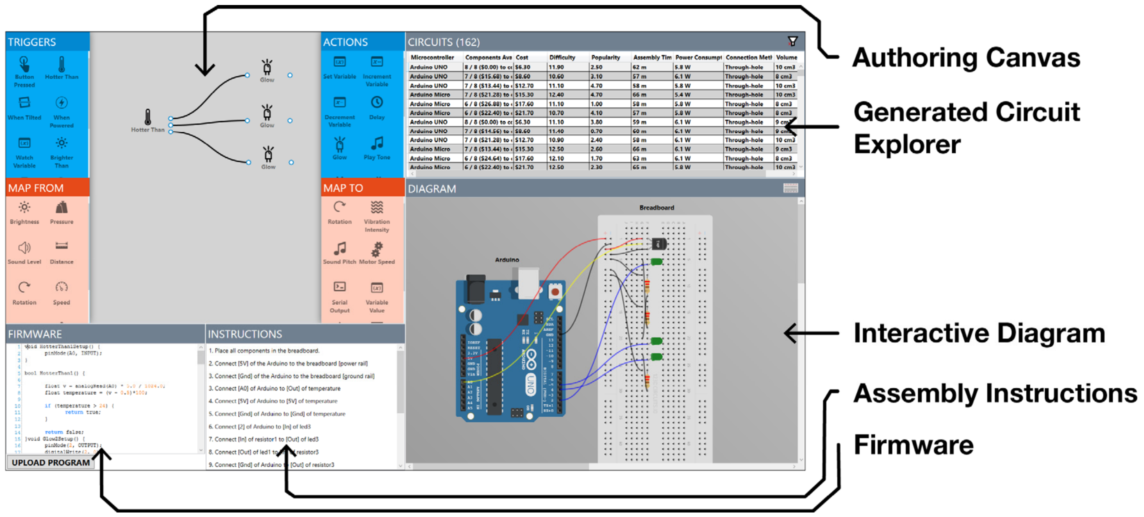
Trigger-Action-Circuits: Leveraging Generative Design to Enable Novices to Design and Build Circuitry

Fraser Anderson, Tovi Grossman, George Fitzmaurice
User Interface Research Group, Autodesk Research
{first.last}@autodesk.com



Microcontroller	Components Ava	Cost	Difficulty	Popularity	Assembly Tim	Power Consumpt	Connection Metl	Volume
Arduino UNO	8 / 8 ($0.00) to cc	$6.30	11.90	2.50	62 m	5.8 W	Through-hole	10 cm3
Arduino UNO	7 / 8 ($15.68) to	$8.60	10.60	3.10	57 m	6.1 W	Through-hole	8 cm3
Arduino UNO	7 / 8 ($13.44) to	$12.70	11.10	4.70	58 m	5.8 W	Through-hole	10 cm3
Arduino Micro	7 / 8 ($21.28) to	$15.30	12.40	4.70	66 m	5.4 W	Through-hole	10 cm3
Arduino Micro	6 / 8 ($26.88) to	$17.60	11.10	1.00	58 m	5.8 W	Through-hole	8 cm3
Arduino Micro	6 / 8 ($22.40) to	$21.70	10.70	4.10	57 m	5.8 W	Through-hole	8 cm3
Arduino UNO	8 / 8 ($0.00) to cc	$6.30	11.10	3.80	59 m	6.1 W	Through-hole	9 cm3
Arduino UNO	7 / 8 ($14.56) to	$8.60	11.40	0.70	60 m	6.1 W	Through-hole	9 cm3
Arduino UNO	7 / 8 ($21.28) to	$12.70	10.90	2.40	58 m	6.1 W	Through-hole	10 cm3
Arduino Micro	7 / 8 ($13.44) to	$15.30	12.50	2.60	66 m	6.1 W	Through-hole	9 cm3
Arduino Micro	6 / 8 ($24.64) to	$17.60	12.10	1.70	63 m	6.1 W	Through-hole	8 cm3
Arduino Micro	6 / 8 ($22.40) to	$21.70	12.50	2.30	65 m	5.8 W	Through-hole	10 cm3

```
void HotterThan1Setup() {
       pinMode(A0, INPUT);
}

bool HotterThan1() {

       float v = analogRead(A0) * 5.0 / 1024.0;
       float temperature = (v - 0.5)*100;

       if (temperature > 24) {
              return true;
       }

       return false;
}void Glow2Setup() {
       pinMode(2, OUTPUT);
       digitalWrite(2, 0
```

1. Place all components in the breadboard.
2. Connect [5V] of the Arduino to the breadboard [power rail]
3. Connect [Gnd] of the Arduino to the breadboard [ground rail]
3. Connect [A0] of Arduino to [Out] of temperature
4. Connect [5V] of Arduino to [5V] of temperature
5. Connect [Gnd] of Arduino to [Gnd] of temperature
6. Connect [2] of Arduino to [In] of led3
7. Connect [In] of resistor1 to [Out] of led3
8. Connect [Out] of led1 to [In] of resistor3
9. Connect [Gnd] of Arduino to [Out] of resistor3

Figure 1: Overview of the Trigger-Action-Circuits interface, a system that uses a generative design approach to enable novices to construct functional electronic circuits.

ABSTRACT
The dramatic decrease in price and increase in availability of hobbyist electronics has led to a wide array of embedded and interactive devices. While electronics have become more widespread, developing and prototyping the required circuitry for these devices is still difficult, requiring knowledge of electronics, components, and programming. In this paper, we present Trigger-Action-Circuits (TAC), an interactive system that leverages generative design to produce circuitry, firmware, and assembly instructions, based on high-level, behavioural descriptions. TAC is able to generate multiple candidate circuits from a behavioural description, giving the user a number of alternative circuits that may be best suited to their use case (e.g., based on cost, component availability or ease of assembly). The generated circuitry uses off-the-shelf, commodity electronics, not specialized hardware components, enabling scalability and extensibility. A user study demonstrated that TAC helps users avoid problems encountered during circuit design and assembly, with users completing their circuits significantly faster than with traditional methods.

Author Keywords
Prototyping; circuitry; generative design; circuit generation;

ACM Classification Keywords
H.5.2. Information interfaces and presentation (e.g., HCI): User Interfaces.

INTRODUCTION
The ability to rapidly evaluate designs via prototyping is a powerful and widely used approach amongst designers, makers and researchers in many fields. In recent years, several techniques and products have been developed to allow novices to prototype circuitry and electronic devices without extensive technical knowledge. Hardware platforms such as Arduino or Phidgets allow users with minimal electronics knowledge to construct functioning circuits, and visual programming languages such as Wyliodrin enable the authoring of higher-level software that interfaces with such hardware. While these tools have reduced the barrier to entry, they still require some technical background.

Despite advances in hardware development platforms, novices are still intimidated by circuitry. The development of circuits still requires extensive knowledge of electrical theory as well as knowledge of, and about, a large library of components. While more approachable hardware platforms such as LittleBits and Phidgets [11] have been developed, such platforms are constrained to the use of proprietary components and subsequently, the functionality that is supported by the manufacturer.

Similarly, while specialized programming languages can simplify programming activities and make them more approachable, they still require a base level of programming knowledge (e.g., control flow, Boolean logic, memory, etc.), which many designers and makers do not have. For example, a recent study by Booth et al. showed that only six out of twenty participants (some of which had programming backgrounds), could successfully complete a simple physical computing task [2].

In this paper, we present *TAC* (*Trigger-Action-Circuits)*, a system supporting novice users in the design and assembly of functional electronic devices. The system uses a generative design approach, allowing users to specify desired functionality at an abstract behavioural level using triggers (i.e., inputs) and actions (i.e., outputs), as well as continuous 'to-from' mappings. From this specification, TAC generates a variety of candidate circuits using its database of components, and presents the alternatives to the user so that they can choose the most appropriate circuit for their task. TAC is able to generate and upload the associated firmware to a microcontroller, and creates the corresponding diagram and assembly instructions to guide the user through the construction process.

TAC is currently developed for use with Arduino-compatible microcontrollers, and can support numerous input and output components. While this initial implementation is sufficient to support a wide range of circuits and behaviors, the concepts which we present could be extended to support other hardware platforms and component libraries. We demonstrate the use of TAC for the design and assembly of a variety of circuits, including those from the Arduino Starter Kit. A study shows that TAC helps users avoid problems encountered during circuit design and assembly, with users of the system able to complete the "Love-o-Meter" circuit in an average of 36 minutes, while all participants in the baseline condition unable to finish within the 45 minute time frame.

RELATED WORK

Recent developments in the rapid prototyping of electronics have enabled users to quickly implement, assemble, and evaluate their ideas with greater ease and fidelity. We build on this prior work, and also draw inspiration from the field of generative design, which leverages computation to automatically synthesize design variants.

Assembling Circuits

Novices can have substantial difficulty working with and assembling electronics. Booth et al. [2] conducted a study in which novices were asked to construct a simple circuit with an Arduino, given only a high-level description of the circuit's behaviour. Participants encountered a number of challenges along the way, including choosing the wrong components, using the wrong logic and variables, and wiring components incorrectly. Only 6 of the 20 participants successfully completed the task, highlighting the need for approaches and tools to simplify this process. Mellis et al. [24] conducted a series of workshops probing a similar problem – how novices would construct electronics using circuit boards. These workshops were much more in-depth, but revealed some similar findings, such as erroneous component selection and debugging, but also called for new tools to provide "better abstractions in circuit design software".

Recently, Drew et al. developed the Toastboard, an intelligent breadboard that can assist novices during circuit debugging [9]. This device provides LED indicators on the breadboard itself, along with a software interface that gives more detailed information to the user, including potential troubleshooting tips. Similarly, Bifrost [23] instruments both hardware and software to allow the user to trace the state of their program and compare expected to actual values. In contrast to these hardware solutions, which aids in circuit debugging, TAC presents a software solution to aid in circuit design and assembly.

Rapid Prototyping of Electronic Devices

Several systems have developed solutions that enable users to integrate custom circuitry into their projects. Some of these approaches, such as Inkjet Circuits [17] and Circuit Stickers [14] have enabled users to design and fabricate circuits using readily-available hardware such as inkjet printers. Custom hardware platforms, such as Phidgets, PaperPulse [26], RetroFab [25], MakerWear [18], Physikit [15] and work by Hartmann et al. [12, 13], provide hardware solutions for users to develop systems that make use of specific electronics with little effort. However, these systems are limited to supporting the proprietary hardware modules developed specifically for the respective systems. In contrast, TAC makes use of commercial, off-the-shelf components and supports a wide array of Arduino-compatible microcontrollers and standard components.

Ellustrate [21] enables novice users to sketch out circuits on a tablet, and guides them during the design and fabrication processes. However, this approach is limited to simpler electronic designs with no high-level behaviour design.

There are also number of software-centric approaches that aim to simplify the programming of circuits and electronic devices. ACAPpella [7] and iCAP [8] allow for programming by demonstration of context-aware applications, letting the user demonstrate the trigger they wish to recognize directly, with no programming involved. However, their approach is limited to specialized hardware and a narrow set of recognizable triggers. In a similar fashion, the context toolkit [6, 29] allows for simple composition of recognizable contexts, but still requires knowledge of programming. In contrast, TAC presents users with a high-level visual programming language, lowering the threshold for use and simplifying the specification of the desired behaviour. Additionally, TAC uses off-the-shelf, commercially available hardware. Both the supported behaviours as well as the components are modifiable by end users, enabling users with technical knowledge to increase the capabilities of the system.

A number of visual programming languages have been developed to ease hardware programming and data flow management, such as Jigsaw [16], NodeRED[1], and LabVIEW[2]. Similarly, Pineal [20] leverages a visual programming language to enable users to prototype interactive objects using a smart phone. Unfortunately, these languages are simple visual representations of complex programming concepts (e.g., flow control, variables), or have limited hardware support. In contrast, IFTTT[3] allows for the high-level specification of behaviours through trigger-action programming, but is limited to select commercial products (e.g., Phillips Hue, Twitter) and may require significant technical knowledge to support custom hardware devices.

Generative Design

Generative design has emerged as a means to enable designers, engineers and artists to specify high level rules, goals, constraints, or problems and have the computer produce and present potential solutions [10]. In contrast to traditional design approaches where users select, modify, and create all elements of design, a generative design approach "provides tools to vary designs beyond direct manipulation of specific design elements" [32]. Enabling the designer to operate at a high-level and leveraging computational power to explore alternatives allows for a greater number of designs to be evaluated, and can enable the creation of designs that would not have been possible by humans alone [4]. This approach could also be used as a pedagogical tool for novice designers [5], providing a platform for introducing key concepts of the target domain.

These prior systems that leverage generative design typically allow users to specify geometry, forces, or other physical constraints [19] and then the system presents a set of 3D objects that meet these requirements using approaches such as topology optimization [1] or genetic algorithms [3]. In contrast, this work enables designers to specify the desired behavioural requirements of an electronic device, with the system generating the required circuitry. While the objective is similar to traditional geometry-based generative design, (i.e., the high-level specification of design goals), the domain, implementation, and use case are different and pose unique challenges.

There is existing work within the field of electronic design automation (EDA) to leverage computation to optimize the design of circuitry [22]. Such approaches focus on the low-level design and optimization of circuitry [30, 31], enabling circuits that are more robust or efficient than what could be designed by hand. These existing works support highly-technical engineers in refining and creating complex circuitry, whereas the current work focuses on enabling novices to author and prototype electronic devices. Recent work by Ramesh et al. [27] explores the promise of generative design for circuits using a constraint solver, but little attention is paid to how users would author the behaviours, explore alternatives, or assemble the device.

TRIGGER-ACTION CIRCUITS

TAC is a system that generates circuitry, firmware, and assembly instructions from a user's high-level behavioural description of the desired functionality.

User Walkthrough

To illustrate a typical use case of TAC, a scenario is described in which a designer would like to construct a device that monitors the temperature of a storage room. The device should display the current temperature on an LCD panel, turn on a warning light if the temperature exceeds 40°C, and sound an alarm if the temperature exceeds 80°C.

Authoring

To begin authoring the behaviour of the circuit, the user begins by dragging nodes from the side panels onto the canvas. Using Heat, Glow, Play Tone and variable elements, the user configures their parameters and connects them with links. The resulting visual program (Figure 2) contains a relatively small number of elements and requires minimal user interaction to define. The high-level naming and pictorial representation of the behaviours allow the user to quickly see the intended function of the program.

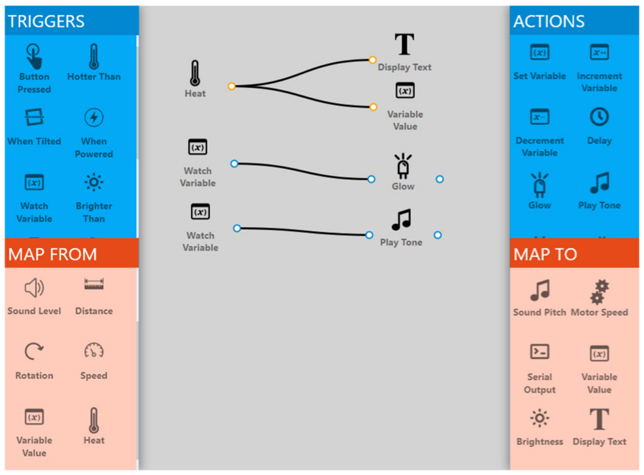


Figure 2: The authored behaviour of the circuit within the authoring canvas.

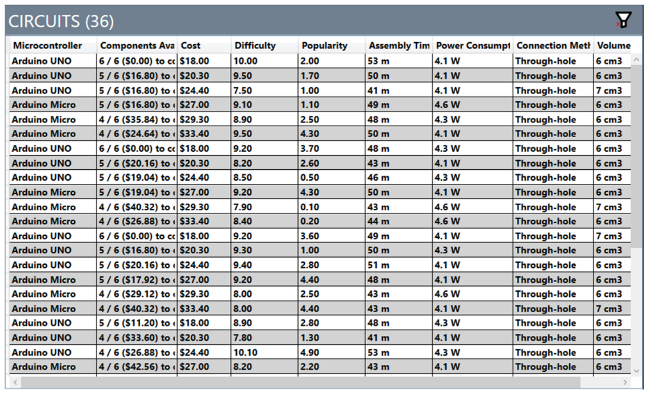
Circuit Selection

During the authoring process, a table of candidate circuits (Figure 3) is updated in real-time, allowing the user to see

[1] www.nodered.org
[2] www.ni.com/labview
[3] www.ifttt.com

the effects of adding each behaviour to the workspace. Each row in the table represents a single candidate circuit, with the values in that circuit representing summary values for the entire circuit. Given that there may be parameters the user is interested in that are not specified in the behaviour (e.g., cost, size, power consumption), the system generates several variants using its database of known components. After completing the authoring, the user inspects the table of potential circuits that will be able to perform the specified behaviour and begins to explore the circuits to determine the most suitable circuit.

As the user is interested in rapidly prototyping the circuit using components they have on-hand, they sort by '*Components available*'. The user notices that some circuits use an Arduino Micro board controller, which they do not have on hand, so they filter those out using the filtering tool in the circuit diagram. After exploring a few circuits and visually comparing diagrams, they select one that has low difficulty and they have all the components for. They also inspect the lowest-cost circuit to get an understanding of the cost if they wanted the circuit in larger quantities.


CIRCUITS (36)

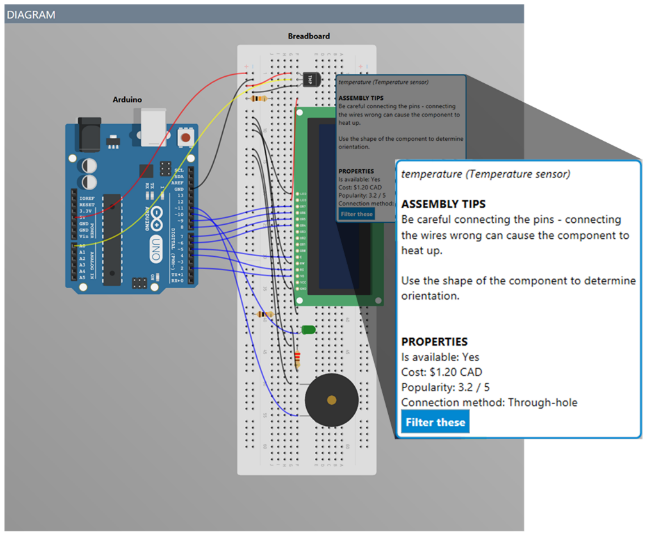
Microcontroller	Components Ava	Cost	Difficulty	Popularity	Assembly Tim	Power Consumpt	Connection Meth	Volume
Arduino UNO	6 / 6 ($0.00) to cc	$18.00	10.00	2.00	53 m	4.1 W	Through-hole	6 cm3
Arduino UNO	5 / 6 ($16.80) to ‹	$20.30	9.50	1.70	50 m	4.1 W	Through-hole	6 cm3
Arduino UNO	5 / 6 ($16.80) to ‹	$24.40	7.50	1.00	41 m	4.1 W	Through-hole	7 cm3
Arduino Micro	5 / 6 ($16.80) to ‹	$27.00	9.10	1.10	49 m	4.6 W	Through-hole	6 cm3
Arduino Micro	4 / 6 ($35.84) to ‹	$29.30	8.90	2.50	48 m	4.3 W	Through-hole	6 cm3
Arduino Micro	4 / 6 ($24.64) to ‹	$33.40	9.50	4.30	50 m	4.1 W	Through-hole	6 cm3
Arduino UNO	6 / 6 ($0.00) to cc	$18.00	9.20	3.70	48 m	4.3 W	Through-hole	6 cm3
Arduino UNO	5 / 6 ($20.16) to ‹	$20.30	8.20	2.60	43 m	4.1 W	Through-hole	6 cm3
Arduino UNO	5 / 6 ($19.04) to ‹	$24.40	8.50	0.50	46 m	4.3 W	Through-hole	6 cm3
Arduino Micro	5 / 6 ($19.04) to ‹	$27.00	9.20	4.30	50 m	4.1 W	Through-hole	6 cm3
Arduino Micro	4 / 6 ($40.32) to ‹	$29.30	7.90	0.10	43 m	4.6 W	Through-hole	7 cm3
Arduino Micro	4 / 6 ($26.88) to ‹	$33.40	8.40	0.20	44 m	4.6 W	Through-hole	6 cm3
Arduino UNO	6 / 6 ($0.00) to cc	$18.00	9.20	3.60	49 m	4.1 W	Through-hole	7 cm3
Arduino UNO	5 / 6 ($16.80) to ‹	$20.30	9.30	1.00	50 m	4.3 W	Through-hole	6 cm3
Arduino UNO	5 / 6 ($20.16) to ‹	$24.40	9.40	2.80	51 m	4.1 W	Through-hole	6 cm3
Arduino Micro	5 / 6 ($17.92) to ‹	$27.00	9.20	4.40	48 m	4.1 W	Through-hole	6 cm3
Arduino Micro	4 / 6 ($29.12) to ‹	$29.30	8.00	2.50	43 m	4.6 W	Through-hole	6 cm3
Arduino Micro	4 / 6 ($40.32) to ‹	$33.40	8.00	4.40	43 m	4.1 W	Through-hole	7 cm3
Arduino UNO	5 / 6 ($11.20) to ‹	$18.00	8.90	2.80	48 m	4.3 W	Through-hole	6 cm3
Arduino UNO	4 / 6 ($33.60) to ‹	$20.30	7.80	1.30	41 m	4.1 W	Through-hole	6 cm3
Arduino UNO	4 / 6 ($26.88) to ‹	$24.40	10.10	4.90	53 m	4.3 W	Through-hole	6 cm3
Arduino Micro	4 / 6 ($42.56) to ‹	$27.00	8.20	2.20	43 m	4.1 W	Through-hole	6 cm3

Figure 3: Generated circuit explorer which allows users to explore the possible circuits synthesized by the system. From this view, users can sort based on criteria, and select the row to populate the diagram, assembly instructions and firmware.

Assembly

The user begins assembly by inspecting the components in the diagram, and placing them in the breadboard (Figure 4). As they place the component, the user hovers over each one in the diagram, revealing various parameters and properties, as well as assembly tips that will help them avoid common pitfalls. These tooltips can also provide contextual information, such as the name of the component or its typical function within common circuits.

Once the components are placed, the user wires the circuit together, following the interactive instructions (Figure 5, right). By clicking on each instruction, the corresponding wires in the diagram are highlighted, with the unrelated wires dimmed out. This allows the user to clearly see what connections are to be made.



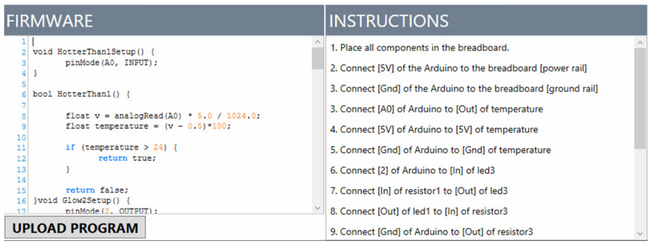
temperature (Temperature sensor)

ASSEMBLY TIPS
Be careful connecting the pins - connecting the wires wrong can cause the component to heat up.

Use the shape of the component to determine orientation.

PROPERTIES
Is available: Yes
Cost: $1.20 CAD
Popularity: 3.2 / 5
Connection method: Through-hole

Figure 4: Rendered circuit diagram with tooltip for the temperature sensor shown. The tooltip is enlarged for readability within this paper.


FIRMWARE

```
|
void HotterThan1Setup() {
        pinMode(A0, INPUT);
}

bool HotterThan1() {

        float v = analogRead(A0) * 5.0 / 1024.0;
        float temperature = (v - 0.5)*100;

        if (temperature > 24) {
                return true;
        }

        return false;
}void Glow2Setup() {
        pinMode(2, OUTPUT);
```

INSTRUCTIONS

1. Place all components in the breadboard.
2. Connect [5V] of the Arduino to the breadboard [power rail]
3. Connect [Gnd] of the Arduino to the breadboard [ground rail]
3. Connect [A0] of Arduino to [Out] of temperature
4. Connect [5V] of Arduino to [5V] of temperature
5. Connect [Gnd] of Arduino to [Gnd] of temperature
6. Connect [2] of Arduino to [In] of led3
7. Connect [In] of resistor1 to [Out] of led3
8. Connect [Out] of led1 to [In] of resistor3
9. Connect [Gnd] of Arduino to [Out] of resistor3

Figure 5: Generated firmware and assembly instructions.

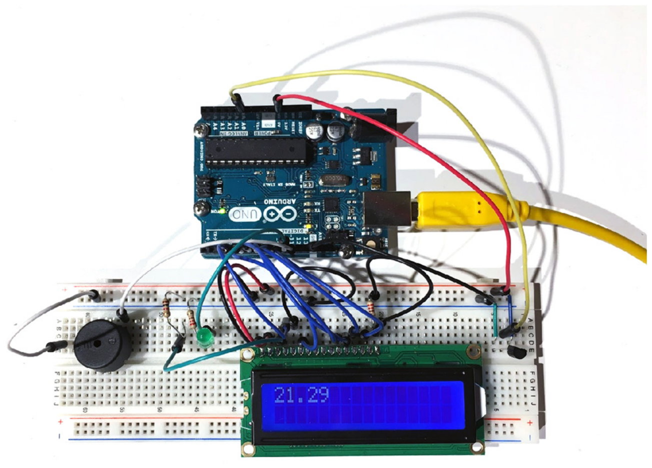


Figure 6: The assembled, functioning circuit.

Upload and Testing

Following assembly, the user may inspect the code to gain an understanding of the underlying program. Once satisfied, they can upload the code directly to the Arduino (Figure 5, left), with the interface providing feedback when the upload is complete. Once uploaded, the users can test and try out their design (Figure 6).

INTERFACE ELEMENTS

The user interface is divided into five panels for each of the tasks the user must complete to construct their circuit.

Authoring Canvas

A behaviour is specified through a visual programming language where users can specify two types of connections: Trigger-Action (TA) mappings, or From-To (FT) mappings. TA mappings allow users to specify an action to complete when a trigger event occurs (e.g., when a button is pressed, turn on a light). FT mappings allow users to specify linear relationships between input parameters and output parameters (e.g., map the angle of a knob to the speed of a motor). These two concepts provide a simple way to address much of the functionality for many prototypes and proof-of-concept applications, and provide a low threshold [28] for novices to begin using the system to explore their designs. Triggers can only be linked to Actions, and Map-From can only be linked to Map-To nodes. To support more complex behaviours with delays, for instance, Actions can be chained together, allowing a single Trigger to cause a sequence of Actions to execute.

Within this framework, variables and logical operators such as 'AND' are supported as behaviours, which raises the ceiling of what users can do with the software. Variables are accessible through top-level behaviours in all TA and FT panels. Users can invoke actions when variables take a given value, set variables when a particular event occurs, and variables can also have their value mapped to an input, or have their value used as input to a 'map from' behaviour. While the use of variables introduces some complexity for the user, it dramatically increases the available functionality of the system, and allows for much richer circuit design.

Behaviours can also define parameters that the user can specify. For instance, the temperature threshold that a 'Hotter Than' behaviour triggers on is specified as a user-defined parameter. If users need to specify multiple parameters, they can clone the parameter set, which creates an additional set of parameters for that behaviour that the user can specify through a dialog box accessible by double-clicking the behaviour. This is useful in instances where multiple triggers are needed that stem from a single component, for instance, setting three separate temperature thresholds. Without this cloning functionality, users would have to create three separate 'Hotter Than' nodes, and the resulting circuits would use three separate temperature sensors.

Generated Circuit Explorer

To facilitate circuit selection, TAC provides users with a spreadsheet-like interface to inspect and explore generated circuits. Each potential solution to the desired behaviour is presented as a row in the generated circuit explorer. Users can sort and navigate the columns to find their desired circuit.

To assist the user in selecting a circuit, TAC computes several metrics for each circuit. The table can be sorted by each of these metrics by clicking on the associated column header. *Cost* is the sum of the cost of all components used in the circuit. *Components available* is computed based in the local database of what components are on-hand – this is currently manually maintained but a future version could be automated and integrated with a component purchasing process. *Difficulty* is computed as a function of the number of components and the encoded connection methods (e.g., surface mount components are weighted as being more difficult than through-hole components). *Volume* and *power consumption* are computed based on the component's defined parameters. *Popularity* and *assembly time* are using manually coded estimates for each component.

Circuit Diagram

Once a circuit is selected in the explorer, the corresponding diagram is created and displayed on the canvas. Users can zoom and pan the canvas to inspect the circuit and terminal names. A button at the top of the diagram allows users to render the components on a breadboard (default), or remove the breadboard and show the components directly wired together. By clicking each wire, all other wires become more transparent, allowing the selected wire to be highlighted making its connections more apparent. Additionally, the instruction corresponding to that wire becomes highlighted in the assembly instructions, giving the user additional context for the connection.

Users are able to filter circuits using the component diagram. By hovering over each component, users can view details about that component, and filter circuits containing that component. This functionality is useful for users who do not have specific components on hand, or who identify components as being too expensive or otherwise unsuitable for their application.

Assembly Instructions

TAC automatically generates instructions once a circuit is selected from the table. The assembly instructions provide a step-by-step guide for the user to wire the resulting circuit together. Each step is described in text (e.g., 'Connect the ground of LED1 to GND of Arduino Uno'), which exposes users to some of the technical terms used in circuitry to help build electronics knowledge. In addition, as steps in the interface are selected, the related connection highlights in the circuit diagram, giving the user a visual reference for the connection.

Instructions can be further augmented, if one of the components specifies assembly hints in its definition. For instance, the LEDs in the system specify 'Ensure the component is oriented the proper way – direction matters'. As connecting some components incorrectly can cause damage (e.g., capacitors, temperature sensors) or cause the system to function incorrectly, these tips are intended to help novices avoid problems with circuit assembly.

Firmware
Once generated, the firmware can be inspected and directly edited by the user. This functionality may be useful to novices interested in learning more about electronics and programming, as it exposes them to the code that executes their behaviour. Exposing the firmware may encourage tinkering, as users can modify and upload the program without fear of causing an irreversible error as the original program can be regenerated by the system. Currently, changes directly to the code do not impact the rest of the interface (e.g., updating the authoring canvas or parameters).

Presenting the firmware allows users to create arbitrarily complex programs by exposing the full functionality of the microcontroller. More advanced users may find this feature useful as they can use the visual programming language to define the basic functionality and component interactions, use the circuit generation and filtering to select components and provide the wiring, then use the firmware editor to add more advanced or specific functionality not exposed through existing behaviours.

Once firmware has been selected and/or modified, clicking the upload button will compile and upload the firmware to the microcontroller that is attached to the USB port.

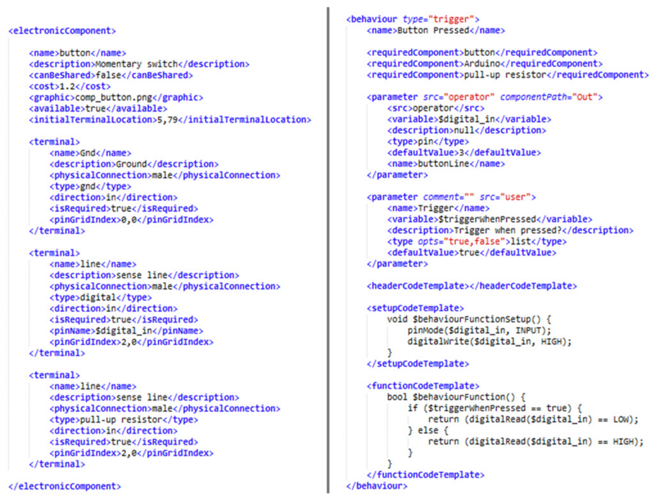
IMPLEMENTATION
TAC is a desktop WPF application written in C#.

Components
Each component that TAC supports is defined using a declarative specification within an XML file (Figure 7). This format allows for the addition or modification of components by third-party authors. Each component is defined by a number of top-level parameters as well as a list of terminals. The top-level parameters describe the type of component (i.e., which requirement it fulfils), as well as its name, description, cost, graphical image, and whether it's currently available to the user. An additional flag indicates if the component can be re-used or shared by other behaviours. Each of its terminals represents a (potential) electronic connection to another component in the circuit. Each terminal is described by its name, type of connection, direction of connection, terminal location within the image, and whether the terminal can be shared. If specific terminals are accessed by behaviours directly (e.g., reading temperature from an analog out terminal), that terminal will be given a name to allow behaviours to reference it.

The current implementation of TAC contains a variety of components to support a broad spectrum of behaviors. Several sensors and switches are currently supported including a temperature sensor, light sensor, accelerometer, infrared distance sensor, momentary switch, and sound sensor. A number of actuator components are also supported, including a number of LEDs, an audio buzzer, a DC motor, servo motor, and a vibrotactile motor. Currently, two microcontrollers are fully supported: the Arduino UNO and the Arduino Micro. The database also contains a number of passive components, such as resistors and capacitors.

Behaviours
The supported high-level behaviours (i.e., Triggers, Actions, Mappings) within TAC are also defined using an XML syntax (Figure 7). As with components, this allows end users with technical knowledge to define and share new behaviours. A behaviour defines the components it depends on, the parameters it supports, as well as several code fragments defining the functionality. The required components list must match those 'types' defined in the component library. Each parameter defines a name, as well as a source and type. The *parameter source* defines how the parameter will be defined (i.e., by the system, component, or by the user). The *parameter type* defines what microcontroller resource will be used, if any (e.g., analog input, PWM output, etc.). A behaviour also defines code fragments that will be placed into a larger template to assemble the complete functionality. Behaviours can provide code that is run during initialization (setup), each time the state is checked (function), as well as to the global area of the program to define global variables or include additional program directives (header).

Using this approach, both simple and more complex behaviours are possible. Examples of supported triggers are: when the temperature exceeds a threshold, when a device is tilted, or when a button is pressed.



```
<electronicComponent>

    <name>button</name>
    <description>Momentary switch</description>
    <canBeShared>false</canBeShared>
    <cost>1.2</cost>
    <graphic>comp_button.png</graphic>
    <available>true</available>
    <initialTerminalLocation>5,79</initialTerminalLocation>

    <terminal>
        <name>Gnd</name>
        <description>Ground</description>
        <physicalConnection>male</physicalConnection>
        <type>gnd</type>
        <direction>in</direction>
        <isRequired>true</isRequired>
        <pinGridIndex>0,0</pinGridIndex>
    </terminal>

    <terminal>
        <name>line</name>
        <description>sense line</description>
        <physicalConnection>male</physicalConnection>
        <type>digital</type>
        <direction>in</direction>
        <isRequired>true</isRequired>
        <pinName>$digital_in</pinName>
        <pinGridIndex>2,0</pinGridIndex>
    </terminal>

    <terminal>
        <name>line</name>
        <description>sense line</description>
        <physicalConnection>male</physicalConnection>
        <type>pull-up resistor</type>
        <direction>in</direction>
        <isRequired>true</isRequired>
        <pinGridIndex>2,0</pinGridIndex>
    </terminal>

</electronicComponent>
```

```
<behaviour type="trigger">
    <name>Button Pressed</name>

    <requiredComponent>button</requiredComponent>
    <requiredComponent>Arduino</requiredComponent>
    <requiredComponent>pull-up resistor</requiredComponent>

    <parameter src="operator" componentPath="Out">
        <src>operator</src>
        <variable>$digital_in</variable>
        <description>null</description>
        <type>pin</type>
        <defaultValue>3</defaultValue>
        <name>buttonLine</name>
    </parameter>

    <parameter comment="" src="user">
        <name>Trigger</name>
        <variable>$triggerWhenPressed</variable>
        <description>Trigger when pressed?</description>
        <type opts="true,false">list</type>
        <defaultValue>true</defaultValue>
    </parameter>

    <headerCodeTemplate></headerCodeTemplate>

    <setupCodeTemplate>
        void $behaviourFunctionSetup() {
            pinMode($digital_in, INPUT);
            digitalWrite($digital_in, HIGH);
        }
    </setupCodeTemplate>

    <functionCodeTemplate>
        bool $behaviourFunction() {
            if ($triggerWhenPressed == true) {
                return (digitalRead($digital_in) == LOW);
            } else {
                return (digitalRead($digital_in) == HIGH);
            }
        }
    </functionCodeTemplate>
</behaviour>
```

Figure 7: Sample definitions for a 'button' component (left) and 'button pressed' trigger (right).

Circuit Generation
To generate the circuits, TAC uses a breadth-first, recursive, dependency resolution. As each trigger and action enumerates the class of components it needs (e.g., 'button'), and each component lists the components it requires (e.g., 'microcontroller', or 'resistor'), the algorithm is able to explore all possible circuit solutions to the current mapping. As each component is added and each dependency is resolved, a list of consumed and available

terminals is maintained. For example, no two components would be able to use the same 'A0' port on an Arduino, as it is not shareable; if a second component needing analog input were added to a circuit already using the 'A0' port, it would be assigned to 'A1'.

The circuit resolution halts when all dependencies have been met. As the algorithm compiles possible circuits, it maintains the state of each, compiling statistics such as the total cost, number of components, and total volume. While this approach is efficient and suitable for the current behaviour and component library, a more sophisticated approach will likely be necessary as the number of supported components increases. More complex filter criteria may be useful to help pare down the increased number of possible circuits that are returned by the system.

Variable Types

TAC uses three variable types to enable the generation of circuits: *user*, *component*, and *system*.

User variables are parameters which require user input or configuration of a behaviour. These parameters are presented with a plain-language description, and reasonable default values. Behaviours can leverage user-defined variables to provide inputs and parameters to their functionality. For example, the '*Hotter than*' trigger provides a 'threshold' variable which the user specifies to define the target threshold for the trigger to execute. Similarly, the '*Display text*' action leverages these parameters to allow the user to specify a string (or series of strings) as input, as well as a Boolean input value (presented as a checkbox) to specify if the strings should be displayed in order of input, or randomly.

Component variables are specified by the behaviour, and defined by the component that satisfies that behaviour. For instance, the '*Hotter than*' behaviour requires a sensor that is able to sense temperature based on an analog voltage reading. Each component that satisfies this behaviour must specify the mapping between voltage and temperature (in Celsius). This mapping is defined through a component-provided variable which specifies the functional relationship between voltage and temperature.

System variables have their values specified by the system during the circuit-resolution process. These variables are used in both behaviours are components, and are used to specify pin mapping. With the '*Hotter than*' example, the behaviour specifies that it is expecting to read a voltage from $analog_pin which corresponds to the voltage provided to the temperature sensor. The component providing the analog value annotates which of its terminals provides the $analog_pin corresponding to the temperature. After mapping and circuit resolution, the system replaces the appropriate values with the mapped values to ensure that the behaviour is reading the value from the appropriate pin (e.g., A0, A1, etc).

Firmware Authoring

TAC generates Arduino compatible code in response to the behaviours created by the user. As parameterized code fragments are associated to each supported Behaviour, TAC can combine and compile each of the fragments into a single executable program using the following steps.

First, each of the parameters is replaced within the code fragment with its respective value (e.g., the user specified value, or the system determined value). The system also appends unique identifiers to function names and variables within a trigger or action, allowing for the many-to-many association. This replacement process operates across all defined triggers and actions. Once processed, the triggers and actions are compiled into a master template which assembles the header, setup, and function code for all defined behaviours into an Arduino-compatible format with setup() and loop() function declarations. Once complete, the assembled code is written to a file, then compiled and uploaded to the connected Arduino board.

Assembly Instruction Creation

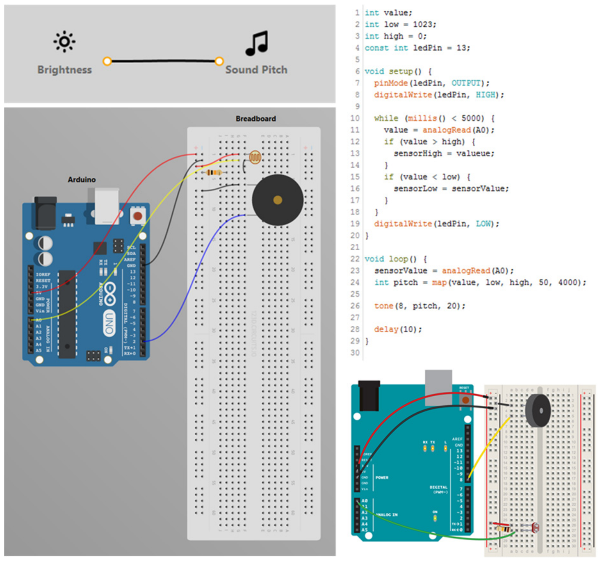
Within TAC, instruction generation is a relatively straightforward process. As each terminal-to-terminal connection is retained during the circuit generation process, the system has a record of which component is connected to which pins. Using this knowledge, the system can generate the appropriate connection list using a lookup table. This approach is sufficient to enable users to understand which terminals should be connected, and supplements the diagram which the users can cross-reference.

VALIDATION

To validate that the system could support the authoring of a wide range of circuitry, the authors replicated the functionality of the fourteen projects contained within the Arduino starter kit. These projects span a wide range of functionality, from creating and sensing digital and analog signals, serial communication, monitoring and changing program state, and include a wide variety of electronic components. While the functionality of each project was replicated (in terms of inputs and outputs), the generated circuit and firmware were different between the provided solution and the one produced by TAC due to the algorithmic way in which TAC generates the solutions.

For clarity and due to space limitations, only two of the fourteen solutions are provided here, and contrasted with the solutions that accompany the starter kit.

One representative project in the Arduino starter kit is the 'Light Theremin', where a buzzer produces a tone, and the pitch of that tone is controlled by the amount of light falling on a sensor. This project requires the use of the continuous mapping functionality (i.e., Map-From and Map-To) within TAC. To author this functionality within TAC, the designer connects a 'Map From Brightness' to a 'Map To Sound Pitch', and provides two sample mappings which will



```
int value;
int low = 1023;
int high = 0;
const int ledPin = 13;

void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);

  while (millis() < 5000) {
    value = analogRead(A0);
    if (value > high) {
      sensorHigh = valueue;
    }
    if (value < low) {
      sensorLow = sensorValue;
    }
  }
  digitalWrite(ledPin, LOW);
}

void loop() {
  sensorValue = analogRead(A0);
  int pitch = map(value, low, high, 50, 4000);

  tone(8, pitch, 20);

  delay(10);
}
```

Figure 8: Sample of solution for Light Theremin project. The left column shows the program (top) and TAC-generated circuit (bottom), while the right column shows the provided code and circuit from the Arduino starter kit. Note the solutions are similar in components and circuitry.

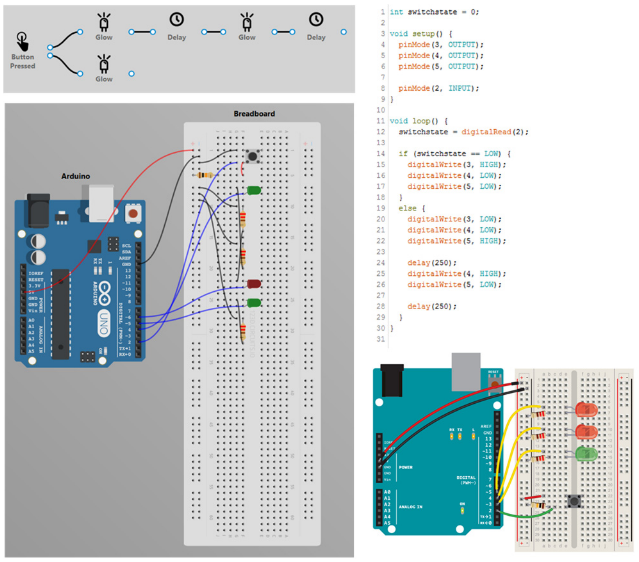
Be used in the linear interpolation mapping (e.g, 10 Lux→100Hz, 50Lux→2000Hz). The six generated circuits all contain four components, and generate 31 lines of code, while the Arduino starter kit solution uses four components and 29 lines of code.

A second project from the starter kit is the 'Spaceship interface', which consists of 3 LEDs and a button. When the button is pressed, two of the LEDs illuminate in an alternating pattern; when the button is not pressed, the third LED lights up. With TAC, this is accomplished by using the 'Button pressed' trigger with two conditions (pressed and not-pressed), linked to 'Glow' and delay actions (Figure 9). All of the 108 resulting generated solutions to this behaviour use 9 components, and generate 105 lines of code, while, the provided Arduino starter kit solution use 9 components and 30 lines of code.

In general, the generated solutions contain more complex code, and often have more complex wiring schemes than the handcrafted, tailored alternatives in the starter kit examples. However, this validation demonstrates the breadth of the system, and also illustrates how limited input from the user (e.g., Mapping brightness to sound pitch) can be sufficient for the system to create the circuitry, software, and assembly instructions for the user.

USER STUDY

Though TAC is able to generate a broad set of circuitry and software that covers the content in the introductory Arduino material, it is not clear if this proposed paradigm is more approachable to users looking to design their own circuitry. To better understand this, we conducted a study which



```
int switchstate = 0;

void setup() {
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);

  pinMode(2, INPUT);
}

void loop() {
  switchstate = digitalRead(2);

  if (switchstate == LOW) {
    digitalWrite(3, HIGH);
    digitalWrite(4, LOW);
    digitalWrite(5, LOW);
  }
  else {
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
    digitalWrite(5, HIGH);

    delay(250);
    digitalWrite(4, HIGH);
    digitalWrite(5, LOW);

    delay(250);
  }
}
```

Figure 9: Sample of solution for Spaceship Interface project. The left column shows the program (top) and TAC-generated circuit (bottom), while the right column shows the provided code and circuit from the Arduino starter kit.

assessed how well novice users were able to design, select, and assemble desired circuitry using TAC.

As a representative task, we used the 'Love-o-Meter' project from the Arduino starter kit, which was also studied by Booth et al. [2]. This task is of moderate difficulty for a novice, can be completed in under an hour, uses a variety of components and programming concepts, and is grounded in prior work. Participants completed this task using either TAC (TAC condition), or using the Arduino IDE (Arduino condition) as in Booth et al [2].

Participants

Twelve individuals (8 male, M = 35, SD = 7.8 years of age) were recruited to participate in the study. The experiment lasted approximately one hour and participants received a $25 gift card as compensation for their time.

Before participating, participants self-reported their prior experience with programming and electronics. This information was used to balance the participants between the two conditions to ensure the populations were comparable. Mann-Whitney U tests indicated there was no significant difference between the two populations in terms of self-rated electronics experience ($U = 16.0$; $p = 0.82$) or programming experience ($U = 12.0$; $p = 0.39$).

Procedure

Participants were informed that they would be building a circuit using an Arduino. They were also given a brief tutorial on breadboards demonstrating how connections can be made with a breadboard (i.e., row-wise and column-wise connections). Depending on the condition, they were either given a brief overview of the Arduino IDE or the TAC interface. For the Arduino condition, participants were

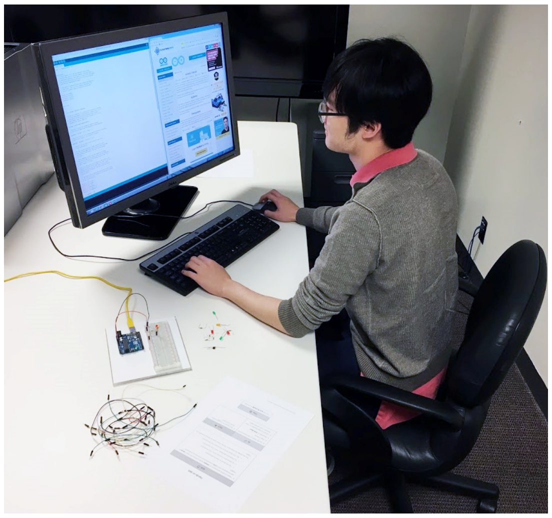


Figure 10: Photograph of the user study setup, with the participant building the circuit without the use of TAC.

shown the editor, the upload and verify functionality, as well as the web browser (Google Chrome). Within the TAC condition, participants were shown the authoring canvas, the generated circuit explorer, diagram area, as well as the instruction and firmware windows.

After the introduction, participants were given a textual description of the functionality that they were asked to replicate. The text explained that their target circuit should use three LEDs which light up in sequence, with more LEDs lighting up as a temperature sensor detects hotter temperatures. Participants were given 45 minutes to complete the task, and were given all the components they would need including some distractor components. They were informed that they could use the internet to search for help, but were instructed not to search for the exact solution (e.g., do not search for 'light three LEDs in sequence in response to temperature'). This procedure is the same as in Booth et al. [2] and describes the functionality of the Love-o-Meter circuit presented in the Arduino starter kit.

Results

All six of the participants using TAC completed the task successfully, while none of the participants without TAC were able to complete the task within the allotted time (Figure 11). These results are similar to those presented by Booth et al. [2] which found the majority of participants unable to complete the circuit. For the participants using TAC, the average completion time was 36 minutes (SD = 7.4 minutes), and a Mann-Whitney U test shows this is significantly lower than the Arduino condition ($U = 0$[4]; $p < 0.01$) where all participants exceeded the 45 minute time frame. As none of the participants in the Arduino condition finished, the task was decomposed into eight subtasks representing observable milestones such as 'uploaded a program' 'connected temperature sensor' and 'controlled an LED based on temperature value'. A Mann-Whitney U test shows a significant difference between the Arduino and TAC conditions ($U = 0.0$; $p < 0.01$) in terms of number of subtasks completed; all participants in the TAC condition completed all 8 tasks, participants in the Arduino condition completed 4.5 tasks on average. Two participants struggled, completing only 1 or 2 tasks, while three participants nearly arrived at the solution, completing 6 or 7 of the 8 subtasks.

[4] Note that U is zero because all values in the Arduino condition are greater than the TAC condition.

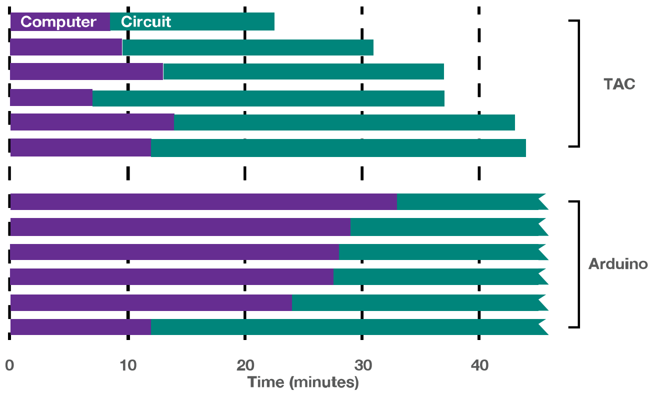


Figure 11: Graph of each participant's total time during the study, separated by time spent on the computer, and time spent with the circuitry.

The time users spent actively using the computer (coding, reading, etc.) versus the time spent working with the circuitry (assembling the circuit, looking at the components) was also compared. A Mann-Whitney U test shows the time spent on the computer was significantly different between the conditions ($U = 2.5$; $p < 0.01$). The time spent with the circuitry was not different between the two groups, likely because none of the participants in the Arduino condition were able to complete the task, so the time spent with circuitry is not reflective of the total time it would've taken if they had finished.

Observations

Overall, the study demonstrated that TAC enabled people without significant experience with circuitry to design and construct functional electronic circuits. One participant felt that after using TAC '*I could tackle a much harder project (and want to) that uses sound and social media*' (P9).

Participants found the behaviour authoring interface intuitive, and most participants using TAC arrived at the correct behaviour within 10 minutes. Some tried to use To-From mappings unsuccessfully before switching to the Trigger-Action paradigm. Additionally, one participant used multiple 'Hotter Than' behaviours resulting in the use of 3 temperature sensors in the subsequent circuits, which they corrected after seeing the diagram.

Several elements of the interface were observed to be quite useful for the novice participants, in particular the rendered circuit diagram. Most of the participants did not follow the step-by-step written instructions, but instead opted to wire the circuit by looking at the diagram. Occasionally, these participants would click on a wire in the diagram to highlight it and see the related text (e.g., to see what pin it was connected to). Participant comments reiterated the perceived utility of the diagram '*The mappings between instructions and diagram was helpful*' (P15). '*I liked how the instructions highlighted the wire that needed to be connected*' (P5). This behaviour of consulting the diagram was not limited to participants using the TAC system. Participants in the Arduino condition often consulted photos or diagrams when trying to determine how to wire and place components, even when there were textual descriptions on the same page.

While participants were able to successfully navigate and select a circuit within the generated circuit explorer, many were unsure which circuit they should choose. Some opted to minimize cost, while others chose at random. Few seemed to note the 'Components Available' column was relevant, indicating which circuits could be built with the components they were given. User feedback suggests that the '*Circuit options [are] a little intimidating*' (P7). One explanation for this is that the task was a laboratory study, and elements such as cost and popularity may not have relevance. Further work is needed to understand what elements should be provided in the circuit explorer, or what use cases may not need the explorer and instead just have a single circuit presented. More generally, exploration of large, generatively designed datasets remains a challenging problem for both novice and expert users.

While most participants used the firmware window solely to upload the code to the microcontroller, one participant with some programming experience used it to edit the parameters during testing. Before uploading, he first skimmed the code to get a basic understanding, then, when trying to determine which threshold parameter should be used to trigger the lights, he edited the code directly and uploaded it. While the same operation can be done using the visual programming interface, it took less effort and was perhaps more intuitive to directly modify the code.

LIMITATIONS AND FUTURE WORK

Our work has shown that the use of goal-driven design to define circuitry can enable novice users to design and construct electronic devices that they may otherwise be unable to. However, there are still a number of limitations to TAC and areas of future work.

While TAC can generate circuits and software, neither the code, nor the diagram is optimized for learnability. More research is needed to determine how to make the code more human-readable, to provide context on the function of each part of the code, and the purpose of the fragment within the larger program. If solved in future projects, then users could more effectively learn electronics from the system's output, and it could be capable of generating custom tutorials for students. While optimizations can be made to enhance the learnability of the generated circuits, the process of automating the circuit creation inevitably removes some opportunity for learning and exploring; more research is needed to examine these trade-offs.

Currently, the authoring interface has limited support for higher-level software constructs, such as looping and complex branching conditions. Further work on the VPL is needed to solve this issue, as well as the issue of changes made directly to the generated firmware not being reflected in the visual authoring environment.

The system is currently limited to generating Arduino-based circuitry and further work is needed to support the generation of circuits that do not include a microcontroller. For example, having an LED glow when a button is pressed only requires a button, resistor, LED and battery, yet with TAC, supporting this behaviour also requires an Arduino, which drastically increases the cost and complexity.

The current system is focused on novice users, however, the application of generative design and design synthesis has many use cases amongst expert users as well. More research is needed to understand potential use cases of professional embedded developers, and to better determine how generative design can be applied to assist their workflows. For instance, a generative design approach to circuitry could allow experts to exchange equivalent circuits easily, allow for rapidly sharing projects in multiple forms for the maker community, and enable rapid prototyping with available parts while easing the transition to manufacturing large quantities with equivalent functionality. Realizing these powerful opportunities requires addressing several challenges, such as being able to accurately encode enough information to ensure that two components or circuits are actually 'equivalent' in terms of the desired functionality.

Lastly, as the circuit and software generation is an automated process, it isolates the user from how the circuit functions, making debugging difficult if something goes wrong. As such, the user does not know if the issue is with their program, or their assembly of the circuits (and components). One potential method to mitigate this is by integrating a software simulator so the user can verify their logic is correct before assembling the circuit. However, more research is needed to provide tools that allow users to more accurately debug within this paradigm, perhaps integrating new technology, such as Toastboard [9].

CONCLUSION

This paper presented TAC, the first system capable of leveraging generative design to create a multitude of design variants for circuitry. The system has a low threshold for novices to begin designing circuitry, and provides the circuit diagram, code, and assembly instructions to enable them to construct a functioning circuit. Through validation by recreating existing projects, and a user study comparing it to the Arduino baseline, TAC was shown to be useful and usable by novice circuit designers, and more efficient than novices using more traditional approaches.

REFERENCES

1. Martin Philip Bendsoe and Ole Sigmund. 2013. *Topology optimization: theory, methods, and applications*. Springer Science & Business Media. Retrieved March 10, 2017 from https://books.google.ca/books?hl=en&lr=&id=ZCjsCAAAQBAJ&oi=fnd&pg=PA1&dq=topology+optimization&ots=y0ffg_7F5J&sig=q5YI0tzfMgsMoph42afZ0bp0HKs
2. Tracey Booth, Simone Stumpf, Jon Bird, and Sara Jones. 2016. Crossed Wires: Investigating the Problems of End-User Developers in a Physical Computing Task. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (CHI '16), ACM, 3485–3497. http://doi.org/10.1145/2858036.2858533
3. Luisa Caldas. 2008. Generation of energy-efficient architecture solutions applying GENE_ARCH: An evolution-based generative design system. *Advanced Engineering Informatics* 22, 1: 59–70.
4. Amaresh Chakrabarti, Kristina Shea, Robert Stone, et al. 2011. Computer-Based Design Synthesis Research: An Overview. *Journal of Computing and Information Science in Engineering* 11, 2: 021003-021003-10. http://doi.org/10.1115/1.3593409
5. Scott C. Chase. 2005. Generative design tools for novice designers: Issues for selection. *Automation in Construction* 14, 6: 689–698.
6. Anind K. Dey, Gregory D. Abowd, and Daniel Salber. 2001. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-aware Applications. *Hum.-Comput. Interact.* 16, 2: 97–166. http://doi.org/10.1207/S15327051HCI16234_02
7. Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. 2004. A CAPpella: Programming by Demonstration of Context-aware Applications. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '04), ACM, 33–40. http://doi.org/10.1145/985692.985697
8. Anind K. Dey, Timothy Sohn, Sara Streng, and Justin Kodama. 2006. iCAP: Interactive Prototyping of Context-Aware Applications. In *Pervasive Computing*, Kenneth P. Fishkin, Bernt Schiele, Paddy Nixon and Aaron Quigley (eds.). Springer Berlin Heidelberg, 254–271. Retrieved November 12, 2014 from http://link.springer.com/chapter/10.1007/11748625_16
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. *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (UIST '16), ACM, 677–686. http://doi.org/10.1145/2984511.2984566
10. Bruno Ferreira and António Leitão. 2015. Generative Design for Building Information Modeling. *Real Time-Proceedings of the 33rd eCAADe Conference*, 635–644. Retrieved March 10, 2017 from http://papers.cumincad.org/data/works/att/ecaade2015_118.content.pdf
11. S. Greenberg and C. Fitchett. 2001. Phidgets: easy development of physical interfaces through physical widgets. ACM, 209–218.
12. Björn Hartmann, Leith Abdulla, Manas Mittal, and Scott R. Klemmer. 2007. Authoring Sensor-based Interactions by Demonstration with Direct Manipulation and Pattern Recognition. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '07), ACM, 145–154. http://doi.org/10.1145/1240624.1240646
13. Björn Hartmann, Scott R. Klemmer, Michael Bernstein, et al. 2006. Reflective Physical Prototyping Through Integrated Design, Test, and Analysis. *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology* (UIST '06), ACM, 299–308. http://doi.org/10.1145/1166253.1166300
14. Steve Hodges, Nicolas Villar, Nicholas Chen, et al. 2014. Circuit Stickers: Peel-and-stick Construction of Interactive Electronic Prototypes. *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems* (CHI '14), ACM, 1743–1746. http://doi.org/10.1145/2556288.2557150
15. Steven Houben, Connie Golsteijn, Sarah Gallacher, et al. 2016. Physikit: Data Engagement Through Physical Ambient Visualizations in the Home. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (CHI '16), ACM, 1608–1619. http://doi.org/10.1145/2858036.2858059
16. Jan Humble, Andy Crabtree, Terry Hemmings, et al. 2003. "Playing with the Bits" User-Configuration of Ubiquitous Domestic Environments. In *UbiComp 2003: Ubiquitous Computing*, Anind K. Dey, Albrecht Schmidt and Joseph F. McCarthy (eds.). Springer Berlin Heidelberg, 256–263. Retrieved November 14, 2014 from http://link.springer.com/chapter/10.1007/978-3-540-39653-6_20
17. 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. *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (UbiComp '13), ACM, 363–372. http://doi.org/10.1145/2493432.2493486
18. Majeed Kazemitabaar, Jason McPeak, Alexander Jiao, Liang He, Thomas Outing, and Jon E. Froehlich. 2017. MakerWear: A Tangible Approach to Interactive Wearable Creation for Children. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (CHI '17), ACM, 133–145. http://doi.org/10.1145/3025453.3025887
19. Sivam Krish. 2011. A practical generative design method. *Computer-Aided Design* 43, 1: 88–100.

20. David Ledo, Fraser Anderson, Ryan Schmidt, Lora Oehlberg, Saul Greenberg, and Tovi Grossman. 2017. Pineal: Bringing Passive Objects to Life with Embedded Mobile Devices. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (CHI '17), ACM, 2583–2593. http://doi.org/10.1145/3025453.3025652
21. Joanne Lo, Cesar Torres, Isabel Yang, et al. 2016. Aesthetic Electronics: Designing, Sketching, and Fabricating Circuits Through Digital Exploration. *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (UIST '16), ACM, 665–676. http://doi.org/10.1145/2984511.2984579
22. Don MacMillen, Raul Camposano, D. Hill, and Thomas W. Williams. 2000. An industrial view of electronic design automation. *IEEE transactions on computer-aided design of integrated circuits and systems* 19, 12: 1428–1448.
23. Will McGrath, Daniel Drew, Jeremy Warner, Majeed Kazemitabaar, Mitchell Karchemsky, David Mellis and Björn Hartmann. 2017. Bifrost: An Interface for Visualizing and Debugging the Behavior of Embedded Systems. *Proceedings of ACM User Interface and Software Technology*, ACM.
24. David A. Mellis, Leah Buechley, Mitchel Resnick, and Björn Hartmann. 2016. Engaging Amateurs in the Design, Fabrication, and Assembly of Electronic Devices. *Proceedings of the 2016 ACM Conference on Designing Interactive Systems* (DIS '16), ACM, 1270–1281. http://doi.org/10.1145/2901790.2901833
25. Raf Ramakers, Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2016. RetroFab: A Design Tool for Retrofitting Physical Interfaces Using Actuators, Sensors and 3D Printing. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (CHI '16), ACM, 409–419. http://doi.org/10.1145/2858036.2858485
26. Raf Ramakers, Kashyap Todi, and Kris Luyten. 2015. PaperPulse: An Integrated Approach for Embedding Electronics in Paper Designs. *ACM SIGGRAPH 2015 Posters* (SIGGRAPH '15), ACM, 9:1–9:1. http://doi.org/10.1145/2787626.2792650
27. Rohit Ramesh, Richard Lin, Antonio Iannopollo, Alberto Sangiovanni-Vincentelli, Björn Hartmann, and Prabal Dutta. 2017. Turning Coders into Makers: The Promise of Embedded Design Generation. *Proceedings of the 1st Annual ACM Symposium on Computational Fabrication* (SCF '17), ACM, 4:1–4:10. http://doi.org/10.1145/3083157.3083159
28. Mitchel Resnick, Brad Myers, Kumiyo Nakakoji, et al. 2005. Design principles for tools to support creative thinking.
29. Daniel Salber, Anind K. Dey, and Gregory D. Abowd. 1999. The Context Toolkit: Aiding the Development of Context-enabled Applications. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '99), ACM, 434–441. http://doi.org/10.1145/302979.303126
30. Adrian Thompson, Paul Layzell, and Ricardo Salem Zebulum. 1999. Explorations in design space: Unconventional electronics design through artificial evolution. *IEEE Transactions on Evolutionary Computation* 3, 3: 167–196.
31. Vesselin K. Vassilev, Dominic Job, and Julian F. Miller. 2000. Towards the automatic design of more efficient digital circuits. *Evolvable Hardware, 2000. Proceedings. The Second NASA/DoD Workshop on*, IEEE, 151–160. Retrieved March 25, 2017 from http://ieeexplore.ieee.org/abstract/document/869353/
32. Loutfouz Zaman, Wolfgang Stuerzlinger, Christian Neugebauer, et al. 2015. Gem-ni: A system for creating and managing alternatives in generative design. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ACM, 1201–1210. Retrieved March 10, 2017 from http://dl.acm.org/citation.cfm?id=2702398