SHF:SMALL:Simplifying Electronics Design for Students and Experts

Steven Swanson
Department of Computer Science & Engineering
University of California, San Diego

1 Introduction

We have entered a golden age of democratized innovation. The Maker movement has made it fashionable to design and build your own gadgets, contraptions, clothing, *objets d'art*, and household articles. Making interests many people, but the learning curve for building many items, especially those involving electronics, is daunting—often prohibitively so.

The electronics required to support many projects are not complicated. For instance, a review of projects on web sites like Hackaday [**?**] and Kickstarter [**?**] shows that, in many cases, the electronics requirements are modest. In many cases, designing them just requires the rote application of well-known engineering principles. For other projects, the majority of the electronics required are simple, and only a small fraction presents challenging engineering problems that require significant creativity to solve.

Although designing these devices relies on well-understood engineering principles, it is not an easy task. Understanding which principles apply and how to apply them requires the designer to scale an intimidating learning curve that includes the design of electronic circuits, techniques for building printed circuit boards (PCBs), knowledge of the available electronic components, and computer programming for microcontrollers and/or other computing platforms.

We propose to automate all or most of this rote engineering work to allow the designer to focus on *design*. We will codify common engineering principles in a software tool chain that will drastically reduce the effort required to design simple electronic systems. The same toolchain will allow more sophisticated designers quickly assemble more complex systems by combining configurable, flexible components with well-defined interfaces.

Our system will make it easier for both novice and experienced designers to quickly design the electronic circuitry for a wide range of applications. We will accomplish this by raising the level of abstraction at which designers specify and implement the systems they wish to build. The result will be a set of novel tools for designing circuits, specifying the requirements of circuits, and reusing circuit components.

Hardware Made EZ will comprise two different interfaces that target novice and expert users, respectively.

The foundation of Hardware Made EZ is a hybrid visual/text-based programming language called C-Code that combines conventional schematic capture with a high-level, text-based programming language. C-Code will provide two core capabilities: First, it will let designers build rich, parameterizable circuit modules. These modules can contain arbitrarily complex code for generating, tuning, or refining re-usable circuit components. C-Code will use modern type-based interfaces, class-based programming paradigms, and software engineering techniques to make the modules robust, flexible, and re-usable. Second, C-Code will allow designers to assemble those modules into larger, complex circuits that are similarly robust. Based on the interfaces the modules provide, C-Code will identify and prevent many common design errors.

Augmenting conventional circuit design with modern programming language and software engineering techniques will make it easier for experienced designers to build complex circuits quickly and we expect it to relax the learning curve required for those learning to build their first circuits. However, C-Code still requires the designer to understand circuit schematics and to deal with many low-level aspects of circuit design.

The final layer in our systems, called Jet, exposes this declarative layer through a user-friendly, web-based interface. Jet) will enable design through a simple, intuitive "drag-and-drop" interface. The designer specifies the device's design (e.g., its shape and where its displays, buttons, USB ports, etc. reside), Jet converts it into a set of C-Code components necessary to implement the device. Then, the C-Code toolchain generates the printed circuit board (PCB) design files and bill of materials (BOM) for the device while a Jet specific tool provides the 3D-printing design files to build the device's enclosure and generates instructions on how to assemble it.

Jet will include a drag-and-drop visual programming environment for specifying the device's behavior, allowing novice programmers to quickly and intuitively test their gadgets and develop the software they will run.

Jet will allow novices (e.g., high school students, college freshmen, and budding hobbyists) to quickly start building simple devices. We plan to evaluate Jet's usefulness in this regard by developing a junior high, high school, and freshmen computer science courses around building and programming electronic devices with Jet. We will evaluate the ability of these curricula to improve retention in computer- and electronics-related fields and to teach engineering, electronics, and computing concepts and skills. The results of these studies will guide the ongoing development of the curricula, Jet, and C-Code.

The proposed work will make the following specific contributions:

- **C-Code** We will develop a novel visual/text-based hybrid programming language for specifying parameterizable electronic circuits.
- **C-Lib** We will develop a library of configurable, re-usable electronic circuit components using C-Code.
- **Jet** We will develop and evaluate a system that leverages C-Code to allow novices designers to quickly and easily design simple electronic devices.
- **Jet-based curricula** We will develop Jet-based curricula for budding designers of multiple ages and skill levels.

In the following sections we outline the challenges that make designing even simple electronic systems a daunting challenge for novices and describe the limitations of the current tools to designing electronics. Then we describe the approach that C-Code will take to address these challenges. Finally, we describe Jet and the curricula we will develop around it.

2 Background

Designing the electronics for a device is both easy and difficult. From one perspective, it is difficult: Designing a printed circuit board require at least a basic understanding of PCBs, PCB design tools, best practices in designing PCB schematics and board layouts, electronic circuits, microcontroller programming, and the available electric components (resistors, capacitors, integrated circuits, etc.). Since modern PCB design tools have limited support for design re-use, an inexperienced designer must wrestle with every details of the design from how what high-level functions the circuit must perform to the part number for each individual electrical component.

From another perspective, designing a circuit is quite easy: For an experienced engineer designing most devices is simply an exercise in applying well-known principles and circuit design idioms to assemble a circuit that meets the design objectives. Indeed, the designer is likely to draw on a "mental library" of circuit design idioms (e.g., power supplies or microcontroller configurations) to assemble the new design.

The problem that Hardware Made EZ attacks is the formalization of the experienced designer's knowledge, experience, and "mental library" so that a novice designer can quickly and reliably apply them to build complex circuits without worrying about every low-level detail of the design. Likewise, more experienced designers can focus their attention on more challenging engineering problems that require creative solutions to novel problems.

Hardware Made EZ breaks the design of a device into three stages: Creating the specification, designing the architecture, and implementation.

2.1 Creating the Specification

The specification for a device is a description of what the device should be and what capabilities it should have. It corresponds to what a product designer might provide to a hardware engineer in charge of implementation.

For instance, a fitness monitor's specification would include a target form factor and the size and type of display as well as a set of capabilities: measuring body movement, keeping time, logging data, being powered by a battery, and communicating with other devices via bluetooth.

The specification would also include a description of the computational requirements for the device. This might be a program that would run on the device or a quantified description of computing power (e.g., 100 MIPS).

Current state-of-the-art Currently available tools for defining device specifications are *ad hoc*, and include drawings, sketches, prose, pseudocode, and, perhaps, example implementations of the necessary functions in a programming language.

Our Goal A more user-friendly approach to specifying the form and function of a device would allow the user to quickly produce a machine-readable specification that a software toolchain could convert to an implementation. Such an interface will necessarily have to balance expressive power with ease of use. Ideally, it would be possible to specify the behavior of the device without sophisticated programming experience.

Different users would benefit from different interfaces for describing specifications. A graphical "drag and drop" interface would be best suited for novice designers, while more experienced designers could use a text-based programming language.

2.2 Designing the architecture

Based on this specification, the hardware designer would break the design into several components and make some decisions about how the device would work. In our example, she would identify a microcontroller capable of implementing the device's functionality, decide which kind of motion sensor to use, determine the specification for the power supply and radio, etc.

This portion of the design process requires the largest body of expert knowledge. In order to select an appropriate part to implement some aspect of the device, they must be aware of the available options and understand the trade-offs among them. For instance, if the device must keep time, the design will include a "real time clock." Digikey uses 75 different features to characterize the 2057 real time clocks they sell at prices range from \$0.26 to \$56.

Several companies, including Adafruit [1] and Sparkfun [2], help with this stage of the design process. They provide "break out or boards" (BOBs) that include all the electrical components necessary for a particular task and include "headers" that make the BOB compatible with a prototyping "breadboard." These web sites make part selection either but severely limit selection. They also charge a premium for BOBs. Adafruit, for instance, sells two real time clock: a highly accurate \$17.50 BOB and a cheaper, less-accurate BOB for \$9.00. The clock component on the \$9 BOB costs just \$1.50 purchased alone.

Current state-of-the art An experience circuit designer applies his or her expertise to the design problem, optimizing the design for the particular application at hand. Or, a less experienced designer combines less flexible BOB-like components to create a less optimized design.

Our Goal Our tools will assemble a suitable architecture by drawing on a library of well-understood components. The component would, for instance, include a set of requirements (e.g., a 5 V power supply) and provide a set of capabilities (e.g., detecting motion). The tool could then assemble and connect a collection of components so that they would provide all the necessary features and have no unmet requirements.

If portions of the design are to complex for the automated tool or use components of which the tool is unaware, the tool will still automatically design the system architecture as long as the designer can specify the *interface* of the components. In this case the tool can generate the majority of the architecture leaving the designer to focus on the novel, challenging, or interesting portions of the design.

2.3 Implementation

The final stage of the design process involves a large set of fine-grain decisions about how to implement the design. These include configuring the circuit by calculating the correct values for discrete components (e.g., resistor, capacitors, and inductors) and the selecting parts (based on physical size, cost, availability, and other characteristics) for those components.

Configuring the Circuit This stage is particularly tedious and error-prone. The datasheets for complex components routinely contain example schematics and formulas that designer should use to compute the parameters for discrete components. For instance, Figure 1 reproduces three partial schematics for the datasheet for a lithium-ion polymer (LiPo) battery charger. The data sheet also contain extensive documentation on how the designer should select the discrete components that surround the central integrated circuit (IC). For instance:

- Resistor R_{ISET} configures the charging current, I_{CH}, according to this formula: $\mathrm{I_{CH}} = 1800\,V/\mathrm{R_{ISET}}$.
- $\mathrm{R_{ISET}}$ should be a precisely calibrated (i.e., 1%) metal film resistor.
- Resistor $\mathrm{R_x}$ configures the charging voltage: $\mathrm{V_{bat}} = 4.2 + 3.04 \times 10{-}6 * \mathrm{R_x}$.
- Capacitive load on the ISET pin can limit the acceptable value of $\mathrm{R_{ISET}}$: $\mathrm{R_{ISET}} < 1/(6.28 \times 2 \times 10^5 \times \mathrm{C})$. An RC filter (the 10 K resistor and $\mathrm{C_{filter}}$ it lower right) can shield the $\mathrm{R_{ISET}}$ pin from this interference.
- The capacitors on the VIN and BAT pins must be as close to the IC as possible.

Values of other components are not discussed, so the designer must figure out their values on her own. For instance, the current carrying capabilities and M1 and D1 are not specified. Likewise, the datasheet provides no guidance on the value of $\mathrm{C_{filter}}$.

Faced with this fragmentary and incomplete documentation, the designer is left with a series of puzzles she must solve. While none of them is particularly difficult for an experienced designer, they all take time and open the door for errors. For a novice engineer, they represent a serious obstacles to implementing a simple device feature: Charging a battery.

To make matters worse, the only mechanism for checking whether a particular configuration is correct is by painstakingly reviewing the datasheet and making sure that the design meets all the requirements.

To draw an analogy to software, this is similar to, instead of providing an full implementation of printf(), the C standard library provided an implementation of write() and then a description of how the programmer should use write() to implement printf(), including a partial an incomplete set of code fragments show how parts of printf() might be implemented. The result, would be a mess: each programmer would be forced to solve the same problem independently, and each of the solution would likely contain errors.

The job for the experienced and novice designer would be much easier if the manufacturer provided a machine readable description of the IC and how it should be configured.

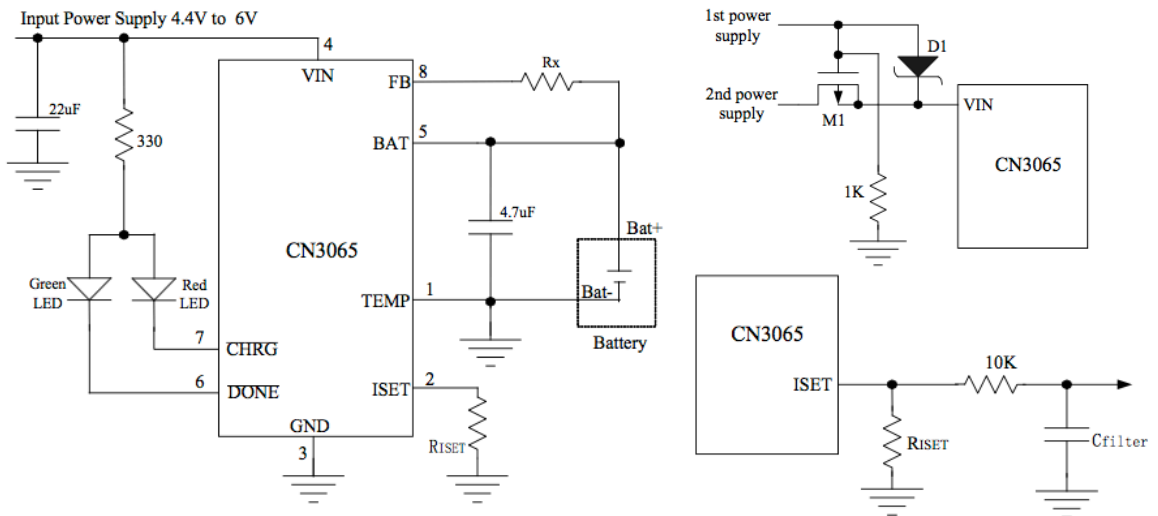


Figure 1: **Datasheet reference schematics for a simple IC** These three schematics provide partial documentation about how the LiPo charger should be configured using external components.

Part Selection Next, the designer must select with specific parts to use in manufacturing the PCB for the device. Selecting which components to use is challenging due to the vast number of choices. For instance, consider R_{ISET} in our LiPo charger and assume we calculate that we need a 100 Ohm metal film resistor for our design. Digikey lists 19 different options. For the more generic 10 KOhm resistor and 22 uF capacitors there are 473 and 1467 options, respectively. Alternatives include package types, different sizes, and temperature ranges. Some are readily available in small quantities. Some are not.

An experienced designer can quickly (although somewhat tediously) home in on appropriate part, but for the novice, the range of options is potentially overwhelming.

Board Layout and Routing The final task of the designer is to set the size and shape of the PCB and physically arrange the parts on the PCB and then connect the components with PCB traces.

The board layout must match the design constraints set forth in the specification. For example, the display and the buttons need to be positioned correctly and the whole PCB must fit within the space allowed.

The layout must also satisfy electrical requirements. For instance, in our LiPo charger, the 22 uF and 4.7uF on VIN and BAT bust be as close to the charger as possible.

Connecting the board's devices with metal traces (i.e., "routing the board") also requires significant effort, even if an "autorouter" is employed to automate much of the work. Depending on the type of signal the trace will carry, they need to be as short as possible, avoid vias (connections between layers), be separated from other signals by a certain distance, or be length-matched with other signals to preserve signal integrity. The signals for power and ground place additional constraints on routing the board. Finally, depending on the design's complexity, the designer may also need to set the "stack up" for the board which determines how many routing, power, and signal layers the board will use.

For sophisticated boards with high-speed signals, routing the board is often considered to be more "art" than science, and expert experience is important to achieving good results. However, for many (or even most boards) good routing results are possible with a good autorouter and careful application of a small set of guidelines and rules.

Our Goal Configuring components, selecting parts, and laying them out on the PCB are all amenable to automation.

Instead of an ad hoc datasheet written in prose and illustrated with fragmentary schematics, a "smart datasheet" would define useful parameters (e.g., the charging current and supply voltage for the LiPo charger) and include code to calculate the necessary resistances, capacitances, and/or circuit topology.

Rather than specify specific parts to include the schematic, the smart datasheet would declare the require characteristics (e.g., the resistance, power rating, and precision of a resistor) and allow the tool flow to select a suitable part from a database.

The smart datasheet could also provide geometric constraints on the board layout that would be combined with constraints from the design specification to guide component placement.

Auto-routers are already quite sophisticated and can route many designs.

2.4 Discussion

We recognize that the fully general problem of design PCBs cannot be completely automated, since some PCBs will push the boundaries of what is possible or include constraints that tools are not yet able to enforce. However, for many PCBs all or most of the process is automatable.

In the next section, we outline the design of a set of tools that will allow designers of different skill levels to produce high-quality device designs as quickly as quickly as possible.

3 Hardware Made EZ

Hardware Made EZ will address the shortcomings and opportunities described in the previous section. To achieve our goal, we will rely on

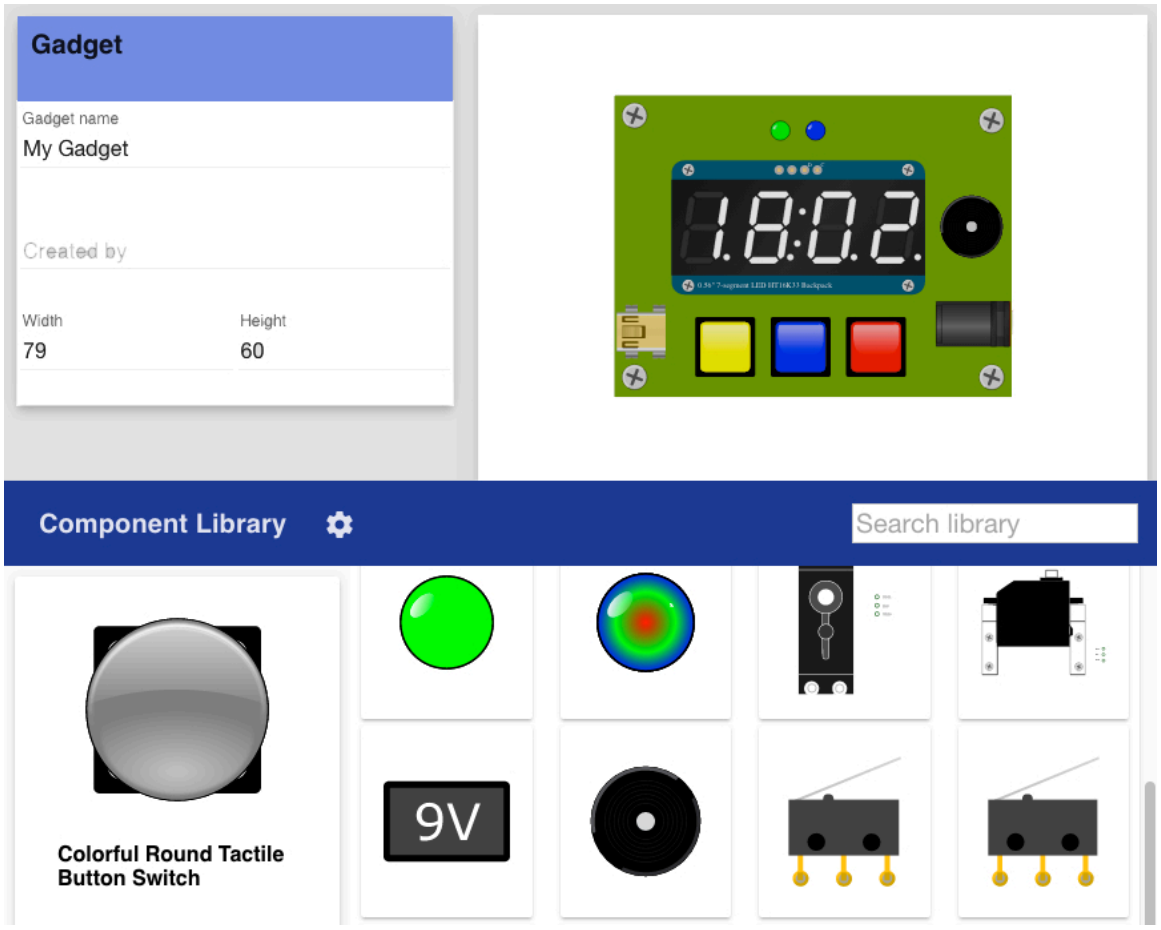
Design principles:

- **Variable skill level** Hardware Made EZ tools will support designers at many different skill levels. Novices can quickly design simple devices and experts will be able to create and implement complex designs more quickly.
- **Incremental adoption** Hardware Made EZ will inter-operate with the existing approach to designing devices so that users can incrementally apply our tools and techniques rather than repeat prior design work with new tools.
- **Multi-modal design** Hardware Made EZ will allow designers to interact with the tool at multiple stages in the design process (i.e., between specification and architecture design and between architecture design and implementation).
- **Programming language techniques** Hardware Made EZ will leverage existing software engineering and programming language techniques such as interfaces, types, and aspects to provide an expressive design language that makes it easy to build working devices.
- **Software engineering practices** Hardware Made EZ will employ modern software engineering techniques such as class-based design, information hiding, and modules to facilitate reuse and testing.

4 Device Design for Beginners

For novices, the technical challenges presented by building even a very simple device can be overwhelming. To address the needs of this group, we will develop a simple "drag-and-drop" tool called Jet for designing and programming devices that allows the designer to focus on the design aspects of the device rather than the details of its implementation.

Figure 2 shows a mockup of the proposed tool. It presents the user with a 3D view of the device they are designing and allows them to add *design elements* to it. Design elements include the aspects of the device that affect how it looks and how it will interact with the outside world. These include things like buttons, displays, decorations, openings in the device's case, and connections cables (e.g., USB). The design elements also include features that might be invisible. For instance, the user could specify that a clock device would include the ability to keep time. The ability to keep time is a design element, even though it does not impact the appearance of the device.


Gadget

Gadget name

My Gadget

Created by

Width

79

Height

60

Colorful Round Tactile Button Switch

Figure 2: **The proposed Jet interface** The Jet interface will allow novice designers to design and program devices.

Jet will also let user specify the shape of the device, its color, and the material it will be built from.

To program the device, Jet will provide a Scratch-like [**?**] visual programming environment. The user will be able to program their device and watch the program run on a simulated version of their device.

Once the designer is satisfied with their design, the Jet toolchain will generate the PCB design files, bill of materials, 3D fabrication files, and source code required to assemble and program the device.

Below, we describe our plans for the user interface, programming environment, case design, and other aspects of the Jet toolchain.

4.1 The Jet User Interface

Jet's goal is to make designing devices as simple as possible while still providing feedback to the designer and the design constraints they face.

The Jet interface will comprise **X** main areas: The Device Workspace, the Component Library, the Bill of Materials, and Design Helper.

The Device Workspace The Device Workspace will let user add, remove, and configure the design elements of the device. It will show a 3D view of the device and allow the user to drag design elements across it surface and provide feedback about where components will fit and where they will not. The workspace will also provide an interface to let the user configure design elements (e.g., setting the size, color and brightness of an LED).

Unlike existing tools for designing electronic systems (e.g., schematic and PCB editing software), the Device Workspace will use user interface idiom similar to those found in drawing programs and other tools that novice designers are more familiar with.

The workspace will also integrate 3D modeling tools similar to TinkerCad [**?**] that provide an intuitive interface for defining 3D shapes to serve as the Devices body.

The Component Catalog The Jet Component Catalog provides the list of design elements that the designer can add to a device. The catalog starts with conventional information that might be found in PCB design library. This includes the dimensions of components, the location of the electrical connections, and metadata like part numbers and manufacturers.

The Jet Catalog includes additional information including 3D models of the design elements, more detailed electrical information (e.g., that two pins comprise an I2C bus), documentation for libraries that can control the component, code for those libraries, and code to simulate the element in side Jet when the user is programming the device inside Jet.

In addition, to this technical information, the catalog will also provide links to materials like data sheets, tutorials, and online video describing the component and how it works.

Fortunately, much of this information in readily available web sites like Adafruit [1] and Sparkfun [2] provide tutorial information and libraries for controlling a wide variety of design elements. As a result, our main effort in developing the catalog will be in developing an XML-based file format that integrate all these different types of information. We envision, eventually, that sites like Adafruit and Sparkfun would provide this information with the products they develop and provide.

The Bill of Materials and Design Helper The Bill of Materials and the Design Helper provide feedback to the designer about the design decisions they make and how they will affect the device. The Bill of Materials will provide a list of all the parts that will go into the device. These will include the design elements they added explicitly, the any utility elements, the Jet toolchain inferred would be necessary to support them, the case, the PCB, and other items like screws and fasteners.

For each of these items it will provide the cost and access to tutorial content about them.

The Design Helper is an interface to the Jet tool chain that provides information about design constraints that the designer faces. For instance, if the Jet tool chains finds that none of the available microcontrollers

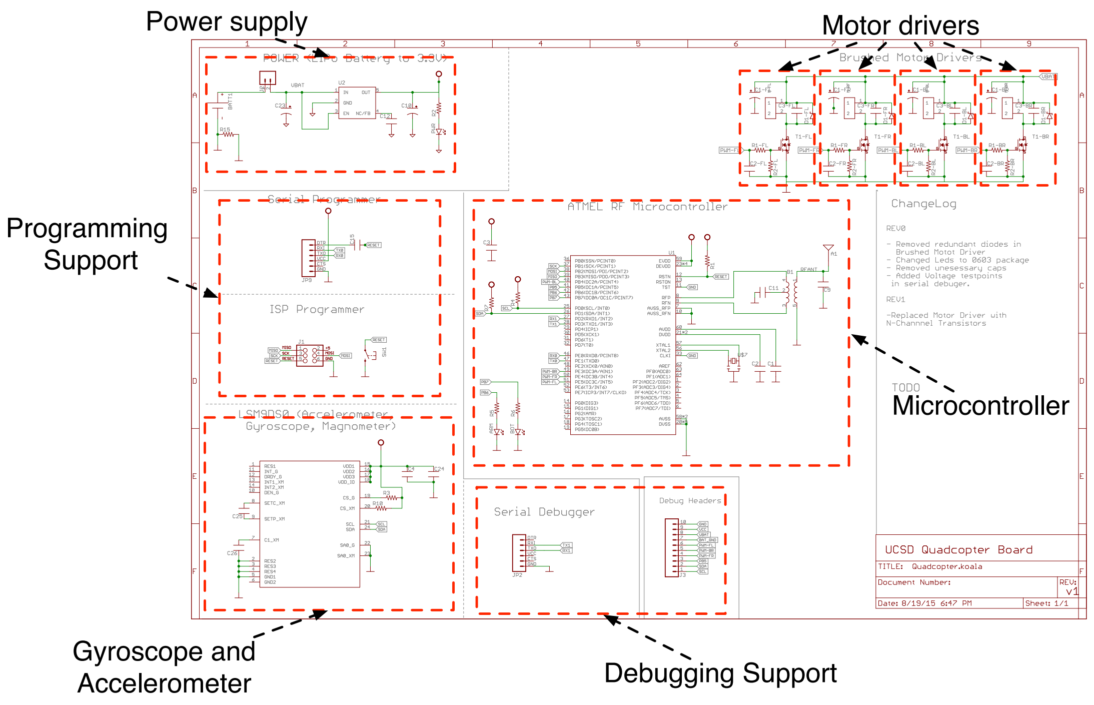


Figure 3: **Example schemamatic** Schematic capture provides a rich visible language for describing an electrical circuit, but it makes it hard to apply software engineering principles like reuse, parameterization, information hiding, and separation of concerns.

have enough pins to control the LEDs the designer has added, the Design Helper will let the designer know and provide suggestions for how to remedy the problem.

4.2 Evaluating Jet

The effectiveness of Jet lies in how easy it is for novice designers to use it in practice. To evaluate Jet and refine its interface, we plan to use Jet in undergraduate computer science and interdisciplinary design courses and collect data about what aspects of Jet help student achieve the course objectives. Section **??** provides detail about the curricula that Jet will enable and how our experience in using the curricula will let us refine Jet's design.

5 C-Code

Existing tools for describing the electrical connectivity between components on PCBs are based substantially on "schematic capture." Connections between components are specified with lines drawn between schematic symbols. Schematic capture is an intuitive mechanism for describing simple circuits, since the schematic designer can rely on a rich set of visual idioms to make the meaning of the schematic clear.

For instance, Figure 3 shows the schematic for a simple remote controlled "quadcopter" built as part of an undergraduate capstone design class. The quadcopter circuitry comprises several mostly-independent components: The microcontroller, the accelerometer and gyroscope, the power supply, the four motor controllers, and various programming and debugging interfaces. The designer has separated these components visually using dashed lines and provided documentation for each of them by embedding text in the schematic.

There are several aspects of this schematic that made it more difficult than necessary to create, modify, and maintain. First, each of the components could easily be used in the design of many other devices. Indeed, several of the parts of this schematic were copied from open-source hardware designs. Of course, any bugs in the original were copied as well, and this design will not benefit from any improvements (or bug fixes) to the original design.

Second, the design contains four identical motor drivers. It happened that this part of the design required several refinements during development, and the process of making the same changes to all four was error-prone and tedious.

Third, the design contains three LEDs with their corresponding current-limiting resistors. Each of the LEDs has different voltage requirements (since they are different colors), and the appropriate value for the resistors depends on the supply voltage. During the design process, we changed from 3.3 V supply to 2.7 V, and we had to manually recompute the values for those resistors. Several of the components in the motor control have similar dependencies on, for instance, the current draw and inductive properties of the motors and the type of battery used.

5.1 The C-Code Library

Design reuse is a central goal of C-Code. To demonstrate the potential of design reuse in designing devices, provide examples to users who want to design their own reusable components, and make it easier for new users to get started, C-Code will provide a standard library of parameterizable, re-usable components. These will include, among other things, power supplies, micro controllers, sensors, motor controllers, wireless communication modules, debugging interfaces, and user interface components (e.g., buttons, knobs, displays, and LEDs).

As an example library component, consider the example of an LED like the one used in Figure 3.

The simplest circuit to control an LED takes an input power source with voltage Vdd and a connection to ground. It includes the LED and resistor, $R1$, to limit current through the LED.

Each LED has a *forward voltage*, V_{fwd}, and a maximum current it can handle, $Imax$. The designer can use Ohm's law to determine the appropriate resistance for $R1$.

$$R1 = \frac{V_{dd} - V_{LED}}{I_{max}}$$

MORE ABOUT LED EXAMPLE HERE

5.2 Case Study: A Quadcopter in C-Code

AN EXAMPLE HERE

6 Using Jet in the Classroom and Beyond

Jet's goal is to make it easier for non-experts to design devices, so demonstrating its success demands that we put Jet into the hands of those non-experts and use their feedback to guide its future developments. We will accomplish this by developing a course at UCSD that use Jet and by releasing Jet to the wider world and listening to the feedback it generates.

6.1 Jet on Campus: Robot Parade

Our first plan to use Jet in the classroom is for class called "Robot Parade" that will be a companion course for one of our introductory computer science course (known as CSE11 at UCSD). CSE11 teaching the basics of programming and objects over a ten-week quarter.

Robot Parade will be part of a large initiative by UCDSD's school of engineering to integrate project-based learning into the freshmen curriculum. We expect to offer Robot Parade at least once a year over the course of this award, and we hope to scale up the class and be able to offer it to many (if not all) incoming freshmen.

Robot Parade will provide students several opportunities not offered by CSE11:

- Robot Parade will be taught in small sections allowing closer interactions between students and course staff.

- Robot Parade will let students apply their new programming skills in a novel context (i.e., programming robots).
- Robot Parade will give students experience working on programming tasks in small groups.
- Robot Parade will let students write programs that interact with the real world.
- Robot Parade will give students experience combining programming the design of physical objects.
- Robot Parade will force students to address multi-dimensional design constraints as part of solving problems.

Robot parade will achieve these goals using a combination of hands-on programming projects, short lectures, and first-hand design experience. Below we provide a course outline and describe our plans for evaluating Robot Parade's effectiveness.

6.1.1 Course Overview

Over the course of 10 weeks, students will learn to design and program simple, Arduino-based robots in a education lab environment. The class culminates in a small group design project in which student design and program a robot to perform a task of their choice.

- **Week 1** Description of course and its goals. Group discussion of what robots are and what they do.
- **Weeks 2-4** Each week, students will learn how to control different "components" of a robot by programming. Components include things like motors, LEDs, bump sensors, distance sensors, servos, and speakers.

  Each week we will also discuss a programming technique useful in programming robots. Example include simple feedback-based control and finite state machines.
- **Week 5** Students use Jet to design a robot that incorporates the components they learned about in weeks 2-4. The design will be subject to cost constraints, size constraints, and the limitations of the Arduino microcontrollers that serve as the robots "brain."
- **Week 6** The students will assemble their robots using basic soldering techniques.
- **Week 7-10** The students will program their robot to perform a task of their choosing.x

The course will culminate with "Robot Parade" where students will demonstrate their robots to the class. The demonstration will be open to other members of the department.

6.1.2 Evaluating Robot Parade

Robot Parade's goal is improving learning and retention among a diverse range of freshmen students. To measure its efficacy, we will administer the Computing Attitudes Survey [**?**] and the Computer Science Attitudes Survey [**?**] to the students who take the class. As a control group, we will also administer the surveys students enrolled in CSE11, but not enrolled in Robot Parade. We will develop the experimental protocol under the supervision of UCSD's Internal Review Board.

Based on the outcome of these surveys, our experience teaching the class, and discussion with students in the class we will refine Robot Parade to improve its effectiveness over time.

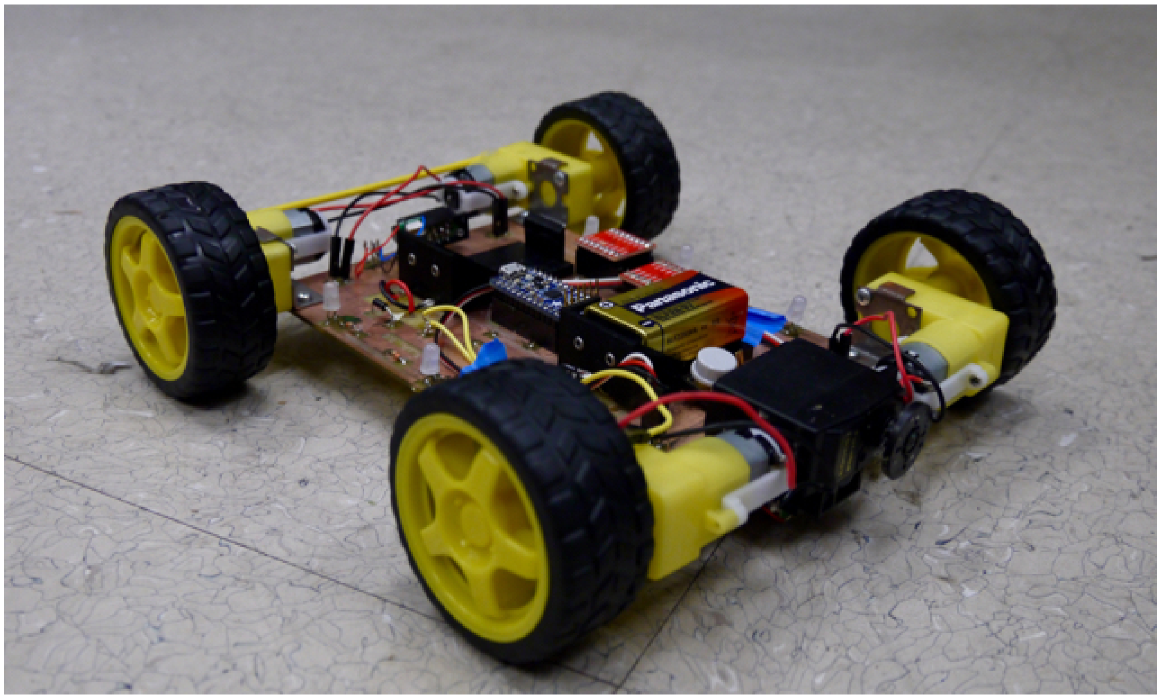


Figure 4: **A Prototype Robot Parade Robot** A robot designed and built by a freshmen students as part of the Robot Parade pilot. The student reported that building and designing robots was a significant factor is deciding to remain in the CS program

6.1.3 Preliminary Success

We already have evidence that courses like Robot Parade can be effective in improving retention. This summer, our lab participated in our department's Summer Program for Incoming Students (SPIS). SPIS bring admitted freshmen onto campus for a 4-week course in computer science to "jump start" their studies in the fall. We offered a Robot Parade-like project to four of the SPIS students. They build robots using more conventional design tools rather than Jet under the guidance of my graduate and undergraduate research students. Figure **??** depicts one of the robots they built. The students had a great time.

Last, week I learned that one of the students from our SPIS groups had been having a very difficult time in CSE11 and had considered dropping out of the major. However, she reported that her experience building robots in SPIS had been an important factor in her deciding to continue in the major. Our hope is that Jet will let us offer design-based, hands-on experiences to many more students and convince some of them to stick with computer science as well.

6.2 Other On-campus Opportunities

UCSD also hosts the California State Summer School for Mathematics and Science (COSMOS) program each summer. COSMOS is similar to SPIS but open to a large range of students. The PI has already discussed a Jet-based module for next summer's COSMOS.

UCSD is building an on-campus "Maker Space" for instructional use and for student-led projects. The PI is on the steering committee for the Maker Space, and plans to working with other instructors and teachers to find potential applications for Jet and the other tools we produce. This experience will also shape how the tools evolve over time.

6.3 Outreach to K-12 Schools and the Community at Large

We plan to work with local schools and other groups interested in education (e.g., local "Maker spaces") to use and refine Jet. We have discussed the idea for Jet and Robot Parade with educators from several local magnet schools about using Jet for classroom or club activities. Once Jet is complete, we will work with these educators to develop appropriate Jet-enabled curricula for their students and then work with them to offer those curricula to their students.

We also plan to make Jet freely available online so people from all over the world can use it to design devices. We expect users will find applications for Jet we have not dreamed of that they will provide invaluable feedback on how Jet can be improved.

7 Results from prior and ongoing NSF support

Steven Swanson is supported in part by NSF award CCF-1219125 ($500,000.00, 7/1/2012-6/30/2015), "SHF: Small: Reengineering Database Systems for Fast SSDs." The project focuses redesigning database systems to take advantage of emerging non-volatile memory technologies.

Intellectual merit This work exploited emerging advanced non-volatile memory technologies to accelerate high-performance data management workloads. As a result of the project, we developed a novel database logging protocol and a novel architecture for a programmable SSD that can accelerate database workloads. The project resulted directly in three publications in top venues [3, 4, 5]. The publications are available for download from the PIs home page.

Broader impact One Ph.D. and one masters student have received their degrees for work directly related to this project. Three other students are working on the project and their work will form parts of there dissertations.

8 Intellectual merit and broader impact

Intellectual merit The proposed work applies well-known software enigneering and system design techinques to the domain of electronic system design. The PI has lead many successful large-scale research efforts that have spanned the hardware/software boundary that applied similar techniques in other contexts. Further, the PI has lead the devolpment of several complex printed circuit board design as well as taught courses that and related subjects. For the past two years he has been working with master students and undergraduates building simple electronic devices in a classroom/lab setting. The research units the PI represents (Computer Science and Engineering and the San Diego Supercomputing Center) have extensive experience in all these fields and ample facilities to support the research.

Broader impact The proposed work will have broad impacts both in making it easier for designers of all kinds to succesfully develop electronic systems and in expanding access to project-based, hands-on educational experiences in hardware and software design. The tools the proposed work will create will be well-positioned to have a significant impact on the practice of PCB design. The tools will interoperate with existing open-source or freely available tools, and we plan to release the tools to encourage their wide adoption. The curricula we develop around the tools will allow learners of all kinds to gain hands on experience with hardware and software concepts. Our preliminary experience building hardware with, for example, college freshmen has shown that it can be a powerful force for increasing confidence and retention in the discipline.

References

[1] Adafruit Industries. http://www.adafruit.com.

[2] SparkFun Electronics. https://www.sparkfun.com.

[3] J. Coburn, T. Bunker, M. Shwarz, R. K. Gupta, and S. Swanson, “From ARIES to MARS:transaction support for next-generation solid-state drives,” in *Proceedings of the 24th International Symposium on Operating Systems Principles (SOSP)*, 2013.

[4] S. Seshadri, M. Gahagan, S. Bhaskaran, T. Bunker, A. De, Y. Jin, Y. Liu, and S. Swanson, “Willow: A user-programmable ssd,” in *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’14)*, 2014.

[5] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, “Near-data processing: Insights from a micro-46 workshop,” *Micro, IEEE*, vol. 34, pp. 36–42, July 2014.