Softening Hardware; Enabling the Interactive Development of Modular Mechatronic Systems

Jake R Read[1]

MIT Center for Bits and Atoms, Massachusetts, jake.read@cba.mit.edu

Leo F McElroy

Hack Club, Vermont / MIT Center for Bits and Atoms, Massachusetts, leomcelroy@gmail.com

Quentin H Bolsee

Vrije Universiteit Brussel / MIT Center for Bits and Atoms, Massachusetts, quentinbolsee@hotmail.com

We overload "Softening Hardware" to refer to making hardware development easier, and doing so by making it more like software development. We suggest doing this by virtualizing hardware devices and lifting systems integration from embedded environments into high-level interactive ones. In our "Modular-Things" project we developed a set of single purpose circuits, a networking library for real-time communication in heterogeneous hardware, and a web-based IDE for integrating complete systems of these devices. "Modular-Things" is not just a collection of dedicated circuits, it's a set of tools for developing modular hardware systems which can be composed in software. The future paradigm we envision is independent developers producing hardware modules which can be easily integrated into complete systems, in a manner analogous to how open-source software developers create libraries. This will be enabled by embedding integration information (like an API) within hardware modules themselves and by exposing this information in accessible interactive development environments.

CCS CONCEPTS • Networks → Programming interfaces; • Computer systems organization → Embedded systems; • Human centered computing → User interface toolkits.

Additional Keywords and Phrases: modular physical systems, virtualization, composability, prototyping frameworks

ACM Reference Format:

Jake Read, Leo McElroy, and Quentin Bolsée. 2023. Softening Hardware; Enabling the Interactive Development of Modular Mechatronic System. In *CHI2023* Workshop [WS2] - Beyond Prototyping Boards: Future Paradigms for Electronics Toolkits CHI '23, April 23–28, 2023, Hamburg, Germany. ACM, New York, NY, USA, 5 pages.

[1]This work was presented at the **CHI2023 Workshop [WS2] - Beyond Prototyping Boards: Future Paradigms for Electronics Toolkits**
CHI '23, April 23-28, 2023, Hamburg, Germany

This work is licensed under a Creative Commons Attribution 4.0 International License.

1 INTRODUCTION

When creating cyber-physical systems developers have to learn how to connect physical components and breakout boards to a central microcontroller (MCU), uncover low-level APIs for these devices, and then write firmware which runs on the MCU. This process presents various limitations due to compiling and flashing firmware hampering development speed, a single embedded computer having limited physical resources (i.e. memory or GPIO pins), and the coupling of functionality across hardware, software, and firmware layers. An emerging approach to mitigating some of these issues is to embed interpreted languages into microcontrollers with technologies like MicroPython. This solution however primarily addresses the ergonomics and speed of development but not limitations associated with embedding all functionality in a single onboard computer, or challenges of linking low level control to high level user facing application interfaces (for example a 3D printer interfacing with slicing software using G-code).

We envision a future of electronics prototyping where users can integrate functional systems by composing networks of many microcontrollers and computers. Such a future demands technology for independent developers to create modules and make use of modules created by others. We suggest an approach to developing such technology based on the principles of layering and discoverability. We developed a set of infrastructure tools for making modular cyber-physical systems and used these tools to create a machine building toolkit consisting of various single purpose circuits. These infrastructure tools allow people to develop modular hardware and integrate that hardware together as software. They include an embedded networking library and a web-based IDE for discovering circuits and integrating them into complete systems in a high-level programming language (JavaScript).

We have deployed our modular hardware systems in university classroom settings for open-ended machine building. We will deploy at scale through Hack Club's global network of teenage technologists by shipping a kit for constructing personal pen plotting robots to hundreds of teenagers around the world in the next six months. The machine will serve as a gateway device to digital fabrication and will allow new programmers with experience generally limited to web-development to utilize familiar technologies (HTML/CSS/JS) to create integrated control systems and interfaces for cyber-physical devices.

Our team consists of a researcher of accessible modular machine building at MIT's Center for Bits and Atoms, an open-source developer who runs educational open-source projects at Hack Club (a global network of thousands of teenage technologists), and a prominent community member from the Fab Lab network (a global network of makerspaces). Collectively we have created open-source tools for learning programming and digital fabrication used by tens of thousands of people around the world. These tools span low-level hardware design and fabrication to high-level application architecting and interfaces.

2 RELATED WORK

Modularity, packaging and ease-of-sharing are common themes in the open-source software ecosystem [6], enabling an exponential growth of collaborative and ambitious projects. We have not seen the same revolution happen with hardware. While accessible physical computing has been enabled by projects like the Arduino [9] or the Raspberry Pi [12], the integration of additional physical modules still often involves writing code at the firmware/OS level, with little to no concept of discoverability or object-orientedness between the controller and the modules. Talkoo [8] introduced plug-and-play modules with a visual IDE to Arduino, though Talkoo required interfacing with modules through a dedicated hub.

In the last decade, there has been a push toward modular sensors by introducing connector standards such as Sparkfun's Qwiic or Adadfruit's STEMMA QT [1], both relying on the I2C protocol for data transmission [4]. While this allowed manufacturers to produce a variety of compatible sensors, the main issue is the absence of discoverability of the added devices. Users must know beforehand which address and sensor type they are introducing to the system, then include the appropriate code snippet to communicate with it. Moreover, there is no consensus on the packet format, often requiring an ad hoc library for each sensor.

More recently, Microsoft research proposed Jacdac [5], a modular hardware system that emphasizes the plug-and-play aspect at the hardware and software level simultaneously, by offering playful connectors and a discoverability system. Adapters also let the user integrate Jacdac with other ecosystems (e.g. micro:bit, Raspberry Pi). While we share similar design goals with Jacdac, their devices are not fully self-documenting. Jacdac's system also relies on a custom bus, whereas our work extends across arbitrary link layers. We want to make it easier for users to create their own modules by not relying on a specific connector or link layer, and giving the users the tools to write their own firmware for novel devices.

3 OUR APPROACH

In our work, we rely on the decreasing cost and increasing performance of microcontrollers and a scalable networking framework to develop single-purpose hardware modules that can be intelligently assembled into application-scale systems in a single programming context. For example in "Modular-Things", Figure 1, single-purpose boards can each be connected to a host computer where they are automatically discovered by a Web IDE. The IDE then queries the boards for their device type, and pairs their firmwares with a matching JavaScript object whose API is presented to the user-programmer. To build their system, the user simply calls methods on the object.

This has the effect of *collapsing* the many-layered systems integration required in most mechatronic systems into a simple programming task. Unless their application requires some fundamentally new module or functionality, it is likely that a modular approach like ours would enable the development of many applications [3]. It also allows for live programming of new electronics systems, since new devices can be added on-the-fly to existing programs. In the cases where new devices need to be incorporated into the system, we have also endeavored to make their authorship easy. Using an accessible library, any embedded project written in Arduino can be quickly added to "Modular-Things". Additionally high level application interfaces such as the sequencer pictured in Figure 2 can be easily built alongside low level machine control.

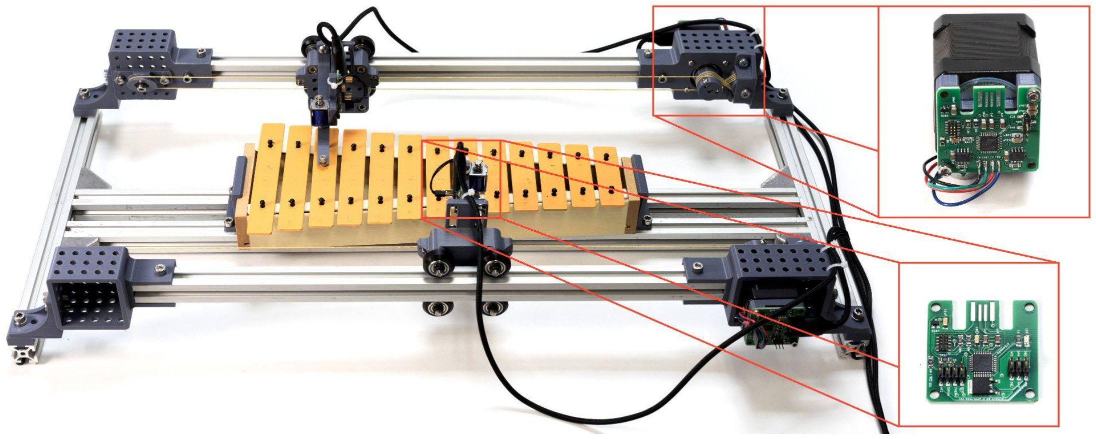
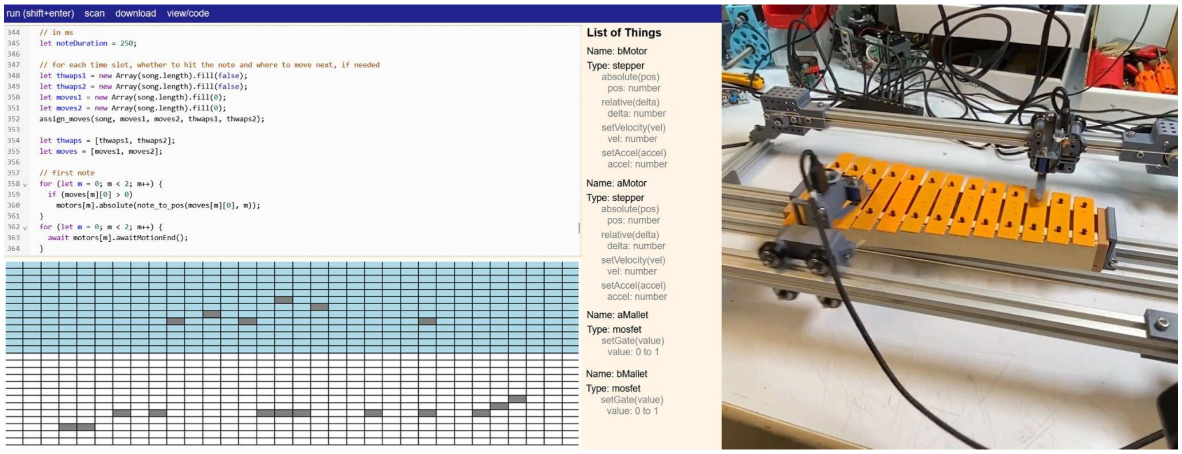


Figure 1. A modular xylophone (above) composed of four modules: two motors and two low-side mosfets (driving solenoid mallets).



```
// in ms
let noteDuration = 250;

// for each time slot, whether to hit the note and where to move next, if needed
let thwaps1 = new Array(song.length).fill(false);
let thwaps2 = new Array(song.length).fill(false);
let moves1 = new Array(song.length).fill(0);
let moves2 = new Array(song.length).fill(0);
assign_moves(song, moves1, moves2, thwaps1, thwaps2);

let thwaps = [thwaps1, thwaps2];
let moves = [moves1, moves2];

// first note
for (let m = 0; m < 2; m++) {
  if (moves[m][0] > 0)
    motors[m].absolute(note_to_pos(moves[m][0], m));
}
for (let m = 0; m < 2; m++) {
  await motors[m].awaitMotionEnd();
}
```

List of Things

Name: bMotor
Type: stepper
  absolute(pos)
    pos: number
  relative(delta)
    delta: number
  setVelocity(vel)
    vel: number
  setAccel(accel)
    accel: number

Name: aMotor
Type: stepper
  absolute(pos)
    pos: number
  relative(delta)
    delta: number
  setVelocity(vel)
    vel: number
  setAccel(accel)
    accel: number

Name: aMallet
Type: mosfet
  setGate(value)
    value: 0 to 1

Name: bMallet
Type: mosfet
  setGate(value)
    value: 0 to 1

Figure 2: Application code for the xylophone is composed in a Web IDE (middle, left) that additionally serves a sequencer UI (bottom left). Module APIs are presented on-the-fly to the programmer (bottom picture middle column).

4 FUTURE WORK

Our work to date uses only a small set of link layers to connect modules to high-level systems, namely USB and UART. However, we have developed a routing layer that is agnostic towards link layers, meaning that many new links could be quickly integrated using SPI, Bluetooth, LoRA, I2C, CANBus, etc. Building these layers is important future work for us. It will allow user-programmers to build a breadth of new types of system - scaling into larger IoT networks using wireless links, and smaller grained, cheaper systems using lighter weight wired links, as well as systems that mix both. We also aim to shrink the size of our core codes to fit onto smaller, lower-cost microcontrollers.

We have begun developing a typed serialization layer. This allows us to automatically build descriptors for typed software objects (like remote procedure calls, or dataflow inputs and outputs), and to packetize their inputs and outputs in a manner that is consistent across applications.

The tools we have assembled so far are successful because they *collapse* layered, networked systems into single-layer assembly tools. We are interested in developing each of our systems' layers as their own independent open-source systems-assembly components, roughly following the principles of "end-to-end" design in our applications [11] and OSI layering in our networks [7].

The serialization layer above will allow us to continue improving what we call "embeddable APIs" - these are discoverability systems that will allow us to embed descriptions of modular objects' functions into their firmwares. These are important because they allow a new kind of collaborative systems assembly; for example, allowing authors of new hardware modules to embed into their firmware all of the information required to integrate that firmware into a system. The user of that module will not need to download any libraries, read any online documentation, or understand any of the module authors' circuit design or pinout. Instead, they can simply plug the new module into their system, read an API descriptor from the device's memory, and begin to experiment with it using live programming. We believe that this amounts to a lightening of the intellectual burden for the module's user, which is a commonly cited factor in the success and rapid growth of open-source development [2] [6] [10].

Finally, we are interested in exploring both language- and dataflow- based high level programming interfaces. We have built versions of both, and can imagine contexts for both. As an overarching theme, we hope that our work across all of these layers will continue to make it easier not just to make modular devices for existing systems, but to make new modular hardware frameworks themselves.

REFERENCES

[1] Bell, Charles. "Introducing Qwiic and STEMMA QT." *Beginning IoT Projects: Breadboard-less Electronic Projects* (2021): 217-258.

[2] Benkler, Yochai. "Coase's penguin, or, linux and the nature of the firm." *Yale law journal* (2002): 369-446.

[3] Blikstein, Paulo. "Gears of our childhood: constructionist toolkits, robotics, and physical computing, past and future." *Proceedings of the 12th international conference on interaction design and children*. 2013.

[4] Corcoran, Peter. "Two wires and 30 years: A tribute and introductory tutorial to the I2C two-wire bus." *IEEE Consumer Electronics Magazine* 2.3 (2013): 30-36.

[5] Devine, James, et al. "Plug-and-play physical computing with Jacdac." *Proceedings of the ACM on Interactive, 5obile, Wearable and Ubiquitous Technologies* 6.3 (2022): 1-30.

[6] Eghbal, Nadia. *Working in public: the making and maintenance of open source software*. San Francisco: Stripe Press, 2020.

[7] ISO/IEC 7498-1:1994 Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model. June 1999. Introduction. Retrieved 26 August 2022.

[8] Katterfeldt, Eva-Sophie, David Cuartielles, Daniel Spikol, and Nils Ehrenberg. "Talkoo: A new paradigm for physical computing at school." *Proceedings of the The 15th International Conference on Interaction Design and Children* (2016): 512-517.

[9] Kushner, David. "The making of arduino." *IEEE spectrum* 26 (2011): 1-7.

[10] Lerner, Josh, and Jean Tirole. "Some simple economics of open source." *The journal of industrial economics* 50.2 (2002): 197-234.

[11] Saltzer, Jerome H., David P. Reed, and David D. Clark. "End-to-end arguments in system design." *ACM Transactions on Computer Systems (TOCS)* 2.4 (1984): 277-288.

[12] Upton, Eben, and Gareth Halfacree. *Raspberry Pi user guide*. John Wiley & Sons, 2014.