QWERTY and The Art of Designing Microcontrollers for Children

Paulo Blikstein
School of Education and (by courtesy) Computer Science Department, Stanford University
520 Galvez Mall, Stanford, CA, USA
paulob@stanford.edu

Arnan Sipitakiat
Dept. of Computer Engineering, Faculty of Engineering, Chiang Mai University
239 Huaykaew Rd., Muang, Chiang Mai, Thailand
arnans@eng.cmu.ac.th

ABSTRACT

Microcontroller-based or physical computing devices have been used in educational settings for many years for robotics, environmental sensing, scientific experimentation, and interactive art. In this paper, we discuss design principles underlying the several available platforms for physical computing, based on a historical analysis of the development of these devices, and data from workshops conducted with students. We evaluate two of the main frameworks for physical computing ("Cricket" model and "Breakout" model), discuss affordances of each platform, and propose a new software and hardware design for microcontroller - based platforms.

Categories and Subject Descriptors
K.3.1 [Computers and Education]: Computers Uses in Education

General Terms
Design, Experimentation

Keywords
Education, interaction design, physical computing, robotics.

1. INTRODUCTION

Seymour Papert famously compared the use of the BASIC programming language for kids and the "qwerty" keyboard layout [5]. Designed to slow down typers in the early days of mechanical typewriters, the "qwerty" keyboard became too popular to be replaced by a better design, even when the original design rationale was obsolete. In a similar vein, Papert also criticizes the techno-centric approach in educational technologies which are often designed from needs and constraints of technologists instead of students.

This paper initiates a discussion about the design of physical computing platforms for children. Surprisingly, despite the popularity of these platforms, there is little research examining this topic from a usability perspective, taking into account the history of the development of these platforms. Could the field of interaction design for children be suffering from a version of the "qwerty" phenomena – importing popular technologies from other professional fields without the necessary adaptations for their use by children? In this paper, we will discuss two prevalent designs for microcontroller-based devices for children, and argue for a more careful consideration of their affordances and usability by young audiences. We will also introduce the idea of Spatial Computing [9] and how it could constitute an alternative programming paradigm for physical computing, which could be a better fit for the projects typically done by students.

The history of the use of microcontrollers in education begins with probeware (i.e., data acquisition devices), since Bill Walton's Laboratory Calculus at EDC in the early 70s [11]. Soon after, Papert and collaborators first envisioned the use of programmable robotics with children [4]. However, it was not until the late 80s and early 90s, when designers started to create products especially tailored for use in education that educational robotics became popular (for example, the Handy Board, the Handy Cricket and the Lego Mindstorms kit, [3, 6, 7]). More recently, the Lego NXT kit, the Pico Cricket, the GoGo Board [8], and the MIT Tower [2] have incorporated new features based on advanced sensing and modular design. Other platforms such as the Phidgets, are also part of this group, and cutting edge frameworks such as the Handy Board Blackfin are practically a full-blown computer.

However, another design tradition developed in the early 2000s, more focused on expandability and exposing the microcontroller's connection for use by other circuits, such as Wiring, the CREATE USB interface (later CUI32), and Basic Stamp, and PICAXE. Launched in 2005, as a fork of the Wiring platform, the Arduino design was the culmination of this design, attracting users in massive numbers, and becoming the de-facto standard in physical computing. Several platforms have used arduino-inspired designs, such as the Freeduino, Cortino, Netduino, and Chipino. The popularization of physical computing brought about by Arduino-inspired designs has been very positive for the IDC community. Microcontroller programming has been made affordable, compatible, multiplatform, and approachable to an unprecedented number of people, and the synergy in the community has produced more than 100 add-ons board (shields) and numerous software libraries.

Despite all the positive aspects of the advent of these platforms, one aspect needs to be reconsidered by researchers – how much do the hardware design principles impact how approachable and usable a platform is for young audiences with little technical interest or background? What are the consequences for educational use of the design choices made for the Arduino/Wiring-inspired platforms, compared to Cricket-like platforms? And finally, how can we rethink programming for physical computing, and how can we better integrate software and hardware to make it more approachable for children?

Constructionist researchers have shown since the sixties that not all programming languages are made equal [4, 5]. Logo has had a significant impact in K12 education because it was easy to learn and use by the average child, and designed based on carefully-crafted, theory-inspired design principles (low threshold, high ceiling, body sintonicity, etc.). Based on them, Papert and collaborators made a strong case for why LISP and BASIC were not good choices, and why there was a need to have a special language for children. They made the case that *media matters*, in other words, the particular affordances and properties of the constructive building blocks offered to children are determinant for what they can build, create, and learn. In particular, they made an important distinction between *understanding the inner workings* of a technology and the *content* that we want children to learn using them. For example, Papert was interested in Logo as a way to for children to learn the powerful ideas in mathematics and computer science (i.e., differential geometry, recursion, etc.) and not how the computer's memory was being managed by the operating system or the transistors were wired inside the microprocessor. The history of Logo, and more recently of the Scratch programming environment, shows how this design principle is crucial and powerful to engage children with technological tools for learning.

2. HARDWARE MODELS

2.1 The "Cricket" model

One of the first widely used microcontroller-based devices for children, the MIT Cricket [3], was also the origin of the LEGO Mindstorms kit and inspired an entire generation of devices. The Crickets were designed for autonomous use – it executed programs stored in its internal memory, but it was not designed to interact with software being used on the computer. Therefore, it was optimized for autonomy. The Cricket was compact, light, used infrared to connect with the computer (to enable quick, wireless reprogramming and resetting), and had a battery pack. In addition, because of its use in robotics, it had independent polarized connectors for motors and sensors, motor driver chips, and resistors for sensors inputs. The LEGO Mindstorms kit, NXT and WeDo kits, the GoGo Board, and the PicoCricket also incorporated this same design. In this design, each port has independent pins for ground, power, and signal, which enables children to connect devices without any additional circuitry (breadboard, resistors, etc.), and the external components to be "off the shelf" sensors, light bulbs, or LEDs without any additional components. Plugging a sensor or a motor to such platforms is as simple as plugging a device to an electrical outlet, and builds on this familiar practice. The battery pack, in most of these designs, serves as both autonomous power, physical support, and protection for the exposed circuitry underneath the board. In short, they were designed to be low-threshold, mobile, autonomous, sturdy devices, able to survive daily use by children.

The Cricket and the GoGo Board platforms also included support for add-on boards, using a custom bus (Cricket) or the industry-standard I^2C architecture (GoGo Board). Finally, the GoGo Board brought some other unique contributions. The first was the tethered mode, with which users could leave the board connected to the computer through a USB cable, and bring the sensor data in real time to the computer, communicating with several software packages such as Microworlds, Scratch, Microsoft Visual Studio, Adobe Flash, and NetLogo. The second was the fact that it was open-source, easy to build with simple tools, and low-cost. Cricket-inspired designs have been refined for more than a decade to enable children to quickly build robotics and physical computing projects – the latest generation of devices, such as PicoCrickets, offers a true plug-and-play experience.

2.2 The "Breakout" model

Arduino-like platforms were originally created by professional interaction designers for college-level physical computing projects. Born at the Ivrea Institute in Italy in 2005, as a fork of the Wiring and the Processing projects, the design principles were to have a simple, low-cost and extensible platform. The Arduino architecture achieves its goal of expandability and flexibility by exposing the microcontrollers' pins and capabilities to the user. This "breakout" design, also present in architectures such as the Propeller and Basic Stamp, eliminates one level of abstraction by foregrounding the microcontroller's internal architecture. The pins are reconfigurable and not assigned to particular functions. The advantages of such design decision are dramatic: for example, it's easy to move from an Arduino prototype to a custom printed circuit board by adding the microcontroller and some components to a new board, keeping the pin assignments and code. It is also a very flexible environment for programming, since pins can be dynamically allocated to different purposes. Also, many components are off loaded to the breadboard, such as resistors and motor drivers, reducing cost and component count.

2.3 The Cricket vs. Breakout design

Both platforms were designed for different audiences and use cases, but the literature is scarce in contrasting them. The comparison is important in the context of optimizing design decisions for use with young audiences, especially in middle school, when children have a limited understanding of basic concepts in physics and electronics. We present relative advantages and shortcomings in Table 1. With **[Ed]**, we indicated items that are commonly needed in educational environments, and with **[Ex]**, items commonly needed by experts. This classification is based on our experience working with both groups in robotics and physical computing workshops for novices [8, 9, 10].

Table 1. Comparison between Cricket vs. Breakout designs

Item	Cricket design	Breakout designs
Learning curve	Easy	More difficult
Programming [Ed]	Logo-like	C-like
External components [Ed]	None, mostly	Many (resistors, etc.)
Autonomy [Ed]	Out of the box	No, need shields
Robustness of connections [Ed]	High (polarized)	Low (can easily come apart)
Auto-ID of sensors	Some designs	No
Motor drivers [Ed]	Yes	No, need shields
Batteries [Ed]	Yes	No, need shields
Independent ports [Ed]	Yes	No, need shields
Expandability [Ex]	Low	High
Number of ports [Ex]	Low	High
Reusable MCU [Ex]	No	Yes
Complex programs [Ex]	Mostly no	Yes
Libraries [Ex]	Some	Many

One conclusion of this comparison is that some of the crucial requirements for educational use are not present in the breakout-inspired boards, conversely, many advanced features are indeed present, but absent from most of the Cricket-inspired platforms. Some successful and ingenious cross-over designs exist, such as the LilyPad Arduino [1], which has custom circuit boards for LEDs, sensors, and motors already supplied with soldered resistors and extra components, making its use much more approachable by novices.

We have run robotics workshops for several years in a variety of countries and socioeconomic settings. More recently, we started to run studies in which students use both "cricket" and "breakout" designs, in the context of physical computing classes for middle/high school, undergraduates, and non-technical graduate students. One consistent observation when introducing robotics to young audiences is that their knowledge of electricity and physics is extremely basic. We repeatedly found that high-school and non-technical undergraduate students could not understand several basic concepts, for example: (1) the difference between analog, PWM, and digital pins, (2) what abbreviations such as "GND," "PWM" and "V_{cc}" meant, (3) the difference between 5V and 3.3V, (4) the need for of "pull-up" and "pull-down," resistors (5) voltage/current dividers. Understanding the architecture of a breadboard was also challenging. The overhead involved in explaining those technical details was significant, and generated considerable anxiety in novice users, who reported having used the breakout-based boards without understanding the rationale of the physical connections. Indeed, just like writing a BASIC program, many of those technical issues are not intrinsically important for the task of building physical computing devices.

A second problem was that more often than not students wanted to build autonomous robots and cars, and therefore needed a power source. Usually they would connect a 9V battery to a power jack and tape it to the board, which would often come apart. Children often want to drive and reverse motors using their boards, which required a separate shield. Finally, we found that offering simple autonomous sensing without programming is a very effective tool to introduce robotics and sensors. In breakout-inspired platforms, that cannot be accomplished without programming and extra shields, but many of Cricket designs have this feature out of the box.

Cricket-inspired platforms also have their limitations, such as the lack of add-on devices. For advanced students, the possibility of adding Ethernet connectivity, wireless communication, playing music files, and using SD cards to record information was appealing. Arduino/Wiring-inspired designs make it convenient to add such functionality, conversely, in Cricket-inspired platforms the pins of the microcontroller are pre-assigned for digital/analog sensing or outputs, taking considerable more board real estate. A project requiring many inputs or outputs will run into limitations.

2.4 Towards age-appropriate physical computing

This preliminary data and analysis point to a curious conclusion. Educators and interaction designers are realizing that the breakout hardware model, *without* add-on boards (motor shield, LCD shield) or specialized daughter boards (à la LilyPad), is not as effective a design for children. A clear evidence of this argument can be seen from the fact that commercial vendors are starting to market shields which essentially make an arduino-compatible board look exactly like a Cricket board, adding motor drivers and independent sensor ports. Some examples are the Babuino board, the Robotuino, and the ProtoShield. If this trend truly represents recognition that the uses of physical computing are different across age groups, we are left with a difficult design conundrum. One of the key features of Arduino-inspired designs is their low cost – but if we add a shield with motor drivers, a display, and a battery board, the cost is considerable, and we end up with a stack of 3 or 4 boards, which could be hard to take apart, and prone to bad contacts. But, if we were to adopt a Cricket-inspired board, advanced students would be limited, and some projects – especially with tens of sensors or multiple outputs – would be difficult. In addition, students would not have available the extensive online support that exists for Arduinos.

3. GoGo Board, GoGo Shield, GoGoino

After years working in tens of schools across the world, it became clear that, when it comes to introducing students to unfamiliar technologies, age-appropriate design matters. Based on this principle, we designed a family of devices that could cater to different audiences and offer scalability. We hope to generate a solution that offers the best of both worlds – the scalability of arduino designs with the ease of use of Cricket-based devices.

Our family of boards begins with the GoGo "Classic," a low-cost PIC-based microcontroller board (Figure 1, left). The board has motor drivers, independent sensor connectors, batteries, and an I^2C bus for add-on devices. We have developed add-ons devices for GPS, home automation, PWM, LCD display, sound, and wireless communication.

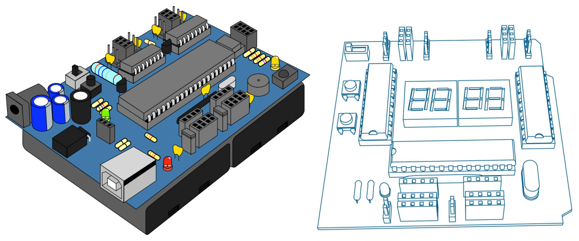

Figure 1. The GoGo Board Classic, and the GoGo Shield.

The second device is the GoGo Shield (see Figure 1, right). The GoGo Shield comprises the most used hardware features in educational settings in one single shield – independent motor and sensor ports, display, buttons for programming-less mode, and easy connection to external autonomous power. We employ the widely tested and effective GoGo Board pin standard and make them addressable in both an "arduino mode" (setting the pins individually using normal arduino code) or a "GoGo Board" mode (referring to them as "sensor 1," "sensor 2," "motor A," "motor B"). The third component is the GoGoino, a software platform which connects Arduinos and GoGo Boards through the I^2C interface, making the GoGo a controllable shield, or the Arduino programmable with Logo. Therefore, young GoGo Board users would have a smooth path towards other platforms, and Arduino users would also have an alternative when working with younger audiences.

4. TOWARDS SPATIAL COMPUTING

We strongly believe that the medium for which children express their ideas is an essential component that determines the possibilities of what they can learn. Even though we believe children do learn a great deal about logical thinking and problem solving (i.e. debugging) by programming in C or BASIC, Papert has shown that there are tremendous opportunities for children to explore knowledge domains that were previously thought too difficult for children. He not only created a new programming language but also a new kind of relationship between the child and the knowledge domain governing the activity being pursued. This kind of design still largely missing when using physical computing with children, and will show some examples based on a new programming environment that we are developing.

The design of Spatial Computing (SC) originated from our observation that children can easily become confused when dealing with multiple conditions – visual representations would be extremely useful. Consider that we wish to control the direction of a motor using two switches: (1) when only **switch1** or **switch2** are pressed the motor turns "this way" or "that way" depending on which switch was pressed; (2) when **both switches** are released, the motor should stop; (3) when **both switches** are pressed, make a beep telling the user it does not know what to do. Though this may seem simple, writing a computer program to follow these conditions is conceptually non-trivial for children regardless of the language used. A typical "buggy" program is shown below.

```
if switch1 then turn-motor-thisway
if switch2 then turn-motor-thatway
if switch1 and switch2 beep-and-stop-motor
if not switch1 and not switch2 stop-motor
```

Many novices would not see the mistake in the above code: multiple rules could fire at the same time. When **switch1** and **switch2** are pressed, all the conditions of the first three rules would be satisfied, resulting in erratic behavior. SC allows students to visually see sensor behaviors and determine the appropriate actions. Figure 3 shows how children can draw "hot spots" as rectangles right inside a 2D graph. This allows one to use his or her understanding of space to define areas of interest and associate actions to them. This is a shift in representation that could lead to children to do kinds of computation previously considered too difficult or abstract. Sipitakiat have used this technique to show how secondary school students could work with robot balance control, a topic typically considered challenging even for engineering students [9].

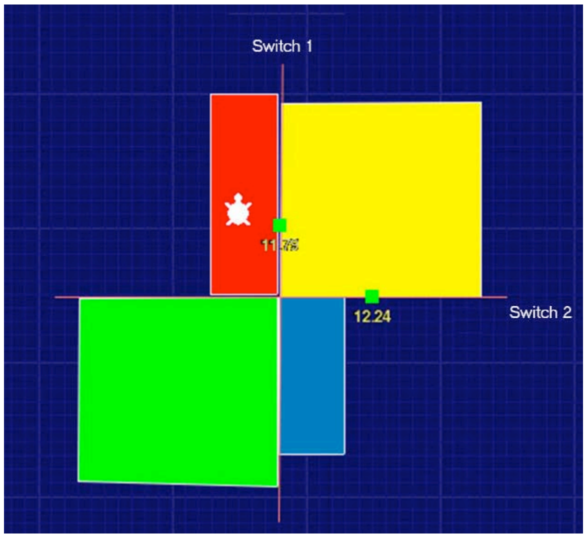

Figure 3. Spatial Computing allows children to observe multiple sensors and define hot spots for conditions.

5. CONCLUSION

The two components of physical computing – hardware and software – deserve a greater attention from interaction designers, so as to prevent a new 'qwerty' phenomenon. We need to systematically consider the differences in the existing platforms and evaluate their age-appropriateness. The Logo tradition offers a useful framework by pointing out how the definition of the level of transparency of a technological tool is crucial. Both arduino-inspired and cricket-inspired platforms are popular and effective for different use cases and audiences, we need to keep in mind that they were designed for a different level of transparency. Students should not be exposed to unnecessarily complex electronics concepts unless they are achievable learning goals. The same is valid for software. By simply transposing the structure of "on-screen" programming to physical computing, we might miss an opportunity to make the physicality of sensors and actuators an aid for children to understand programming. With Spatial Computing, we have observed students' discovery of new ways to program and interact with the physical world.

6. REFERENCES

[1] Buechley, L. 2008. The LilyPad Arduino: Toward wearable engineering for everyone. In *Proc. Pervasive Computing*, IEEE.

[2] Lyon, C. 2003. *Encouraging Innovation by Engineering the Learning Curve.* Cambridge, MA: Master's Thesis, MIT.

[3] Martin F., and Resnick M. 1993. LEGO/Logo and Electronic Bricks: Creating a Scienceland for Children, in *Advanced Educational Technologies for Mathematics and Science.* Springer.

[4] Papert, S. 1971. Teaching children thinking. MIT Artificial Laboratory Memo #247, MIT, Cambridge.

[5] Papert, S. 1980. *Mindstorms: children, computers, and powerful ideas.* New York: Basic Books.

[6] Resnick, M., Berg, R., & Eisenberg, M. 2000. Beyond Black Boxes: Bringing Transparency and Aesthetics Back to Scientific Investigation, *Journal of the Learning Sciences*, 9 (1), pp. 7-30.

[7] Sargent, R. 1995. *The Programmable LEGO Brick: Ubiquitous Computing for Kids.* Cambridge, MA: Media Laboratory Master's Thesis, MIT.

[8] Sipitakiat, A., Blikstein, P., & Cavallo, D. 2004. Moving towards highly available computational tools in learning environments. In *Proceedings of the International Conference of the Learning Sciences*, Los Angeles, CA, USA.

[9] Sipitakiat, A., Cavallo, D. 2008. Giving the Head a Hand: Constructing a Microworld to Build Relationships with Ideas in Balance Control, In *Proceedings of the International Conference of the Learning Sciences*, Netherlands.

[10] Sipitakiat, A., & Blikstein, P. 2010. Think globally, build locally: a technological platform for low-cost, open-source, locally-assembled programmable bricks for education. In *Proceedings of the Conference on Tangible, Embedded, and Embodied Interaction*, Cambridge, USA.

[11] Tinker, R. 2000. *A History of Probeware.* Retrieved from http://www.concord.org/publications/detail.