Arduino: An Open Electronics Prototyping Platform

David A. Mellis
Copenhagen Institute of
Interaction Design
Njalsgade 88
2300 Copenhagen S, Denmark
dam@mellis.org

Massimo Banzi
Tinker.it!
Via Amendola, 2
20052 Monza (MI) Italy
m.banzi@tinker.it

David Cuartielles
School of Arts and Communication
Malmö University
20506 Malmö Sweden
david.cuartielles@k3.mah.se

Tom Igoe
ITP, Tisch School of the Arts, NYU
721 Broadway, 4th floor
New York, NY 10003 USA
tom.igoe@nyu.edu

CHI 2007, April 28 – May 3, 2007, San Jose, USA
ACM 1-xxxxxxxxxxxxxxxxx.

Abstract
Arduino is a platform for prototyping interactive objects using electronics. It consists of both hardware and software: a circuit board that can be purchased at low cost or assembled from freely-available plans; and an open-source development environment and library for writing code to control the board. Arduino comes from a philosophy of learning by doing and strives to make it easy to work directly with the medium of interactivity. It extends the principles of open source to the realm of hardware, supporting a community of people working with and extending the platform. It has been used in universities around the world and in numerous works of interactive art.

Keywords
open hardware, prototyping, microcontrollers, open-source, education, interactive art

ACM Classification Keywords
H.5.2 [Information Interfaces and Presentation]: User Interfaces—prototyping; C.5.3 [Computer System Implementation]: Microcomputers—microprocessors; K.5.1 [Legal Aspects of Computing]: Hardware/Software Protection—open source; J.5 [Art and Humanities]—design

Introduction

There are many tools for prototyping with electronics, used for everything from new musical instruments to intelligent rooms, custom input devices and interactive art pieces. These tools attempt to reduce the difficulty of working with electronics and expand the number of people who can experiment with the medium. Many of them, however, are either commercial products – expensive and closed – or research projects unavailable for use by most people. Others consist only of circuit boards, providing no tools to simplify their programming.

The open source movement, meanwhile, has shown that useful and robust software can be created by a distributed team of volunteers freely sharing the results of their efforts. Open source projects often gather strong communities of people working at many levels: some work on the core code, others contribute small extensions, still others write documentation or offer support, with the majority simply making use of a quality product.

Can we apply the principles of open source to hardware and electronics? What does it mean to make a circuit board which is open and extensible, but still usable with little effort? How can we make working with electronics easy, cheap, and quick? These are some of the questions that led to the creation the Arduino prototyping platform.

This paper discusses related work, the educational and design context within which Arduino was developed, the philosophy behind it, the platform itself, both hardware and software, and the community that has formed around Arduino.

Related Work

There are many different microcontroller development tools available for use in teaching and prototyping. Those that are most popular outside the electrical engineering community work to offer some balance between cost, expandability, and ease of use. Arduino also seeks to balance these factors, while making up for some of the shortcomings of existing platforms[1]. This section presents a survey of a few of the more popular tools on the market, followed by an analysis of how their strengths and weaknesses affect on the design choices behind Arduino.

At the highest level of abstraction are microcontroller tools such as Infusion Systems' MicroDig, Phidgets, and Stanford's d.tools. Modules at this level are generally not programmable by the end user. Instead, they are configured using a desktop tool. These tools are generally not standalone devices, but must be connected to a personal computer in order to be useful.

Infusion Systems' MicroDig [5] is a sensor interface box with a MIDI interface. Its hardware interface consists of an analog-to-MIDI controller with 8 analog inputs, and various sensor modules that mate with the controller. Users attach pre-packaged sensors to the inputs, and connect the controller to a MIDI output device. The values of the sensors are output as MIDI values. The MicroDig is handy for teaching students with some knowledge of MIDI but little programming or electronics knowledge how to design hardware interfaces, because it requires little new knowledge. It is an expensive platform, however, with the basic kit costing $399, and

[1] For a more in-depth comparison, see [2].

requires that the connecting equipment be MIDI compatible.

Phidgets [9] is a modular system of sensor controllers, motor controllers, RFID readers, and other special function devices, all united by a common USB interface and a set of desktop software APIs. Each Phidget device is a self-contained electronic device, whether it's a sensor, motor or LED controller, or a more complex device like an LCD display. The user needs almost no electronics knowledge to use Phidgets. Each device is connected to a desktop computer in order to access its sensor data or to control it. The development team has released application programming interfaces for the system in several languages, including Visual Basic, VBA (Microsoft Access and Excel), LabView, Java, Delphi, C , C++, and Max/MSP. The modules are relatively inexpensive, ranging from $10 to $100. The devices cannot be used as standalone units, however and must be interfaced to a personal computer to use. The learning curve for Phidgets is somewhat steeper than for the MicroDig, but it's useful for those familiar with software development who want to begin making hardware interfaces.

d.tools [11] is a high-level hardware and software tool developed at Stanford University's HCI group that addresses some of the shortcomings of others in this class. First, d.tools is a more flexible system. The d.tools software can be used with other hardware platforms, as long as that hardware is running a firmware that can communicate in the d.tools protocol. Wiring, Arduino and Phidgets hardware have been used with d.tools. The software is is written in Java as a plugin for the Eclipse universal tool platform, and can theoretically run on any Java-capable operating system. Like Phidgets, the hardware is made up of a series of plug-and-play USB modules, each of which communicates with the d.tools software. d.tools also offers a suite of analysis tools which allow users to see the results of their devices graphed on screen, and time-indexed against a video of the person using the device. d.tools is an open source platform, and as of this writing, the hardware is not commercially available.

Moving down a level of abstraction, there are a number of mid-level microcontrollers. Controllers in this range feature a microcontroller with its necessary support electronics (crystal, power regulator, etc.) on a small module. These modules assume users can build input and output circuits to attach to the module. They're usually programmed in BASIC, or some variation of C, and attach to the programming environment on a personal computer using a serial or USB connection. Parallax' BASIC stamp [8] is the most well-known of these modules. Also in this family are NetMedia's BasicX [7], BASIC Micro's Basic ATOM [4] processors. Arduino most resembles these.

The main advantage offered by the mid-level controllers is programmability in a high-level language, with a simple programming interface. The disadvantage is generally that the programming languages are very limited, and the lower levels of the controller itself are not accessible to the user at all. Students taught to use these controllers generally reach a problem beyond the module's capability within their first semester. The programming environments are almost all available for Windows operating systems only, though there are some exceptions.

The processors themselves are usually priced around $50, and experimenter's kits, including a processor, power supply and prototyping board usually cost between $75 and $100. Since they're aimed at beginners, this price, while less than the high-level controllers, is still high. Users don't often think about using multiple modules in a project because the price of multiples is prohibitive. It's inevitable when learning that a person will make mistakes. With mid-level microcontrollers, the mistake of mis-wiring a circuit and destroying the processor is high.

At the lowest level of abstraction are the microcontrollers themselves. Two of the most popular are Microchip's PIC family of processors [6], and Atmel's AVR processors [1]. These are programmed in C or Assembly or BASIC, and much of the programming involves direct access to the processor's registers. A separate hardware programmer is usually needed to communicate between the programming environment and the processor. The developer must build the necessary support circuitry for the processor in addition to any sensor or actuator circuits. The learning curve for this class of controllers is the steepest of any mentioned here.

The advantage offered by lower level controllers is cost. At $1 - $15 apiece, they can be used in multiples easily. There is often some initial setup cost for the development environment, however. Programmers range from $30 - $100, and more fully-featured programming environments can be in excess of $300. This cost is amortized by continued use: the more processors you use, the cheaper it gets. There are some good open source development environments, particularly for the AVR controllers, but they are not designed for beginning users. The interfaces are usually complex, or command-line interfaces, and the code libraries require a working knowledge of C or C++.

Finally there are Arduino's predecessors, Programma 2003 and Wiring. One of the authors (Massimo) developed in 2003 a simple micro-controller platform called "Programma 2003" based on a PIC chip and the open source language Jal. The design goal for this platform was to have something as cheap as possible which would be open source and run on Windows, Mac OS X, and Linux. Programma 2003, however, lacked good documentation, a wide community, and used a relatively unknown programming language that lacked certain key features.

Wiring [3] is a mid-level module, based on one of the AVR microcontrollers. Wiring attempts to address the programming interface limitations of the families of processors above.

The programming environment for Wiring, on which the Arduino environment is based, has its origins in Processing [10], a multimedia programming environment. Wiring, like Processing, was made to teach design students about programming. The environment itself is spare and simple to understand. The language uses clear terms for command names like analogRead() and digitalWrite() rather than the more terse style usually associated with microcontroller flavors of C. The interface for uploading programs to the microcontroller is minimal, allowing users to focus on the task of programming. Menu item names are unambiguous, and kept to a minimum. The environment is written in Java, and is available for

Windows, OSX, and Linux, unlike most other microcontroller development environments. Its availability for OSX alone has led to its increased use by designers and artists who prefer that platform.

The Wiring module is a powerful tool, but it is limited in that it cannot be constructed by a beginner. The module as sold by the developers is priced in the range of other mid-level modules, around $60 -$80, too expensive to allow for easy experimentation or use of many boards.

Context

Arduino was born out of the combination of many prototyping cultures. It was developed at the Interaction Design Institute Ivrea, a small school in northern Italy offering a Masters degree in interaction design. Interaction-Ivrea's curriculum focused on screen interfaces, physical objects, and services, emphasized hands-on work, encouraging designers to create rapid prototypes of their ideas, then repeatedly test and refine them.

The institute also supported the creation of tools for use in the prototyping process. It hosted for a number of years Casey Reas, who worked on the development of Processing. Courses at Interaction-Ivrea used Processing for the creation of prototypes of software interfaces. Programma 2003 was created while Massimo was at Interaction-Ivrea and used in physical interaction design courses. Wiring was developed as a thesis project at the institute and also used in the education program.

Interaction-Ivrea had a philosophy of learning by doing, believing that skills such as programming or electronics are best acquired in the course of more general projects. Students are motivated to pick up the practical knowledge they need to express their design ideas. This, in turn, emphasizes tools that allow for quick experimentation and rapid iteration. Although they were created in educational environments, these platforms were intended primarily to help people build software or hardware prototypes, not as experimental aids in a research investigation.

Philosophy

From the beginning we have been interested in the dissemination of prototyping techniques within design educations as a way of communicating the values of new interactive devices. It has always been our aim to encourage the traditional design disciplines to go beyond the screen, trying to give meaning to human-machine interactivity through actually designing the machine itself.

It is unrealistic to expect designers to become engineers. Instead we believe in prototyping, with which designers can themselves build an object which expresses the desired design intentions. This gives them a valuable tool in communicating with engineers in the further development and realization of an idea. We seek to provide designers with the tools they need to prototype interactions not just forms or materials – turning interactivity itself into a medium for expression.

Dealing with education means making compromises concerning the language to use when communicating, the depth of the contents, etc. Arduino's philosophy avoids removing all the complexity behind electronics as some other platforms do. Electronics are made of components, which are physical devices representing

logical functions, and we believe that we should keep that as part of the education process. Therefore we like to speak about making things "easy enough" for students to get an understanding of how things work by trying them. Rather than trying to hide complexity from users, we prefer to simplify it to the point where they can deal with it directly. Later, they will have the chance to deepen their knowledge by accessing existing resources from disciplines like electric engineering.

Cost was an important consideration when designing Arduino. If a hardware tool is not cheap, people are hesitant to purchase it, slowing distribution and keeping it inaccessible to many people. Further, if the board is expensive, people will not use many of them, meaning they may have to disassemble one work to build the next. One unexpected factor is shipping costs, as a piece of hardware that is cheap in the US may double in price if it must be shipped to, say, China.

From the beginning, Arduino was born as a collaborative project between different universities and individuals. It has always been our goal to maximize the impact within the academic world, trying to raise questions about how we design interactive artifacts. One of the main issues to address is the one related to intellectual property. Physical interaction design is a young discipline. We believe that a good way of making it grow to accommodate society's needs is to look for a way of licensing our results that makes them available for other people to use. We chose to make the whole platform part of the free culture movement, release them under permissive licenses.

Arduino has been left open in many different ways. First, the system is ready to be hacked in different ways both software and hardware-wise. Second the licenses allow reusing the design in other contexts. Third, the education process is not closed, people generate their own materials, workshops, examples and tutorials.

The Arduino Prototyping Platform

Arduino allows users to create working electronic prototypes, either stand-alone objects or devices tethered to a computer. It can read from a wide range of sensors, control a broad spectrum of output devices, and communicate with software running on a computer or talking over a network.

There are many steps required to perform the most basic of tasks with a microcontroller: picking a particular microcontroller, figuring out the circuit needed to use it, ordering the necessary parts, assembling them, downloading the software needed to program the microcontroller, figuring out a way for the microcontroller to talk the computer, installing any necessary drivers, buying or building an external device to program the microcontroller, learning how to write code for the microcontroller (which may require reading a datasheet that is hundreds of pages long), writing the code, working out the command line arguments needed to compile and upload the code, etc.

Arduino attempts to eliminate or ease as many of the steps as possible with a combination of hardware and software.

Hardware

The Arduino board is a printed circuit board containing a microcontroller (basically a low-power computer squeezed into a single chip), and the components

needed to provide it with a stable power supply, to connect it to other components, and to enable it to communicate with the computer.

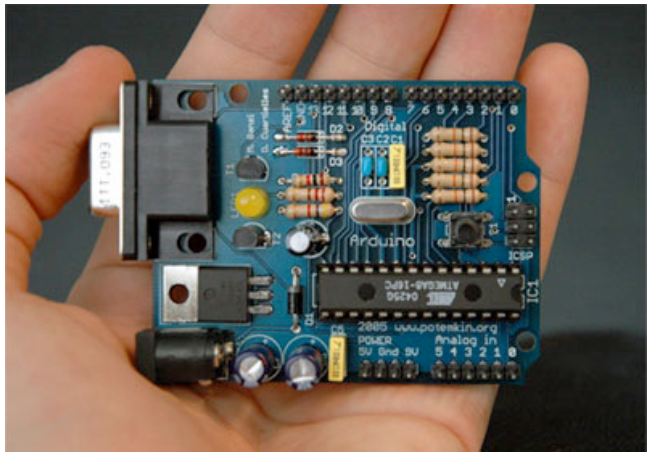

Figure 1. A serial Arduino board with removable Atmega8 microcontroller, one of the original versions of the Arduino hardware.

There are many versions of the board, but the main one (whose most recent version is entitled the Arduino NG) is about the size of a playing card and costs €23 or $32. It contains a removable ATmega8 microcontroller, which can be easily replaced if broken or removed from the board for use in a custom circuit. It includes extra components that enable it to communicate with the computer via USB, as the alternative interface, a serial port, is no longer available on most machines. The board can be powered directly from the USB connection or using an external power supply.

This version of the board is manufactured in quantity to keep costs low and allow people to get started without needing to assemble their own board. Each board is individually loaded with a special piece of software (called a "bootloader") that allows users to program it without any extra devices, further lowering the total cost to the user. By including the manufacturer in the team we were able to engineer the board to higher standards, adding functionalities but keeping the price essentially unchanged. At the moment, assembled boards can be purchased online from the manufacturer, SmartProjects Snc of Chivasso, Italy and through a distributor in the US (SparkFun Electronics). More distributors are planned to provide easier access for users across the world.

Other, compatible, versions of the board are designed to be assembled by hand. They use components large and simple enough to be soldered by beginners and keep the number of parts to a minimum. Interested users can purchase a blank PCB or etch their own from a provided image of the layout, which can be printed and used as a stencil. This means that Arduino boards can be made by anyone with access to basic electronic components, without relying on our manufacturer. Making the hardware out of parts a beginner can use serves another purpose as well: it means that users can build the module on a prototyping board if they wish. This can speed development and reduce the costs of projects that use multiple processors. The schematics and circuit diagrams are licensed under a Creative Commons Attribution, Share-Alike license, meaning that anyone can use them provided they share their derivates with the world.

All versions of the Arduino board are designed to work with standard electronic components. The board provides a base platform but does not limit users to

pre-packaged sensors or actuators. This means that users can use new or unusual components without needing to wait for special Arduino versions of them.

Now that the basic design of the board has been in use for nearly two years, we have begun to create extensions and variations. There are extension "shields," other circuit boards that can be snapped on top of the Arduino board to provide extra functionality like circuitry for handling high power devices like motors, an RFID reader, or a small breadboard that can be used to mount circuits in a compact form. We have also developed a board with a Bluetooth interface and a mini-version that with the same footprint as the BasicStamp. We plan to continue designing and producing hardware as desired by our users.

Software

The Arduino software is an attempt to simplify the process of writing code without unduly limiting the user's flexibility. It builds on many other open-source projects, adapting them to the Arduino hardware and hiding their unneeded complexities. The Arduino software consists of two main parts: the development environment and a core library, both open-source.

The Arduino development environment is a minimal but complete source code editor. It is a cross-platform application written in Java and usable under Windows, Mac OS X, and Linux. In it, users can manage, edit, compile, and upload their programs (called sketches). All functions can be accessed from a set of seven toolbar buttons or a few drop-down menus. The user need not fiddle with makefiles or command line arguments, which can pose significant obstacles for the beginner. The environment includes a serial monitor, allowing the user to send data to and receive data from the board, easing debugging without requiring additional software. In fact, all programs required for Arduino development are included in a single archive downloaded from the Arduino website (except for Linux users, who must install some packages with their distribution's package management tool). The GUI itself is based on the Processing development environment, while sketches are compiled by avr-gcc, and uploaded with uisp. The source code is distributed under the GNU Public License (GPL).

The Arduino core library consists of AVR C/C++ functions that are compiled along with the user's sketch. The combined binary file can then be uploaded to the Arduino board. Using an API compatible with Wiring, the Arduino core encapsulates low-level aspects of microcontroller programming (e.g. register manipulation), allowing users to concentrate on their particular task. In particular, this saves users from having to read the 300 page data-sheet for the microcontroller, the only reliable source for information on its low-level functionality. Users are still programming in standard C/C++, however, so the programming knowledge they acquire can be transferred to many other situations. In fact, the full source code to the Arduino core (licensed under the LGPL) is included in the distribution, so that curious users can learn how it works and modify it. Because of the limited capacity of the microcontroller, some code is split into separate libraries which can be specifically included when required for a particular sketch. Anyone can write an additional library, which can be installed by simply moving it to the correct directory.

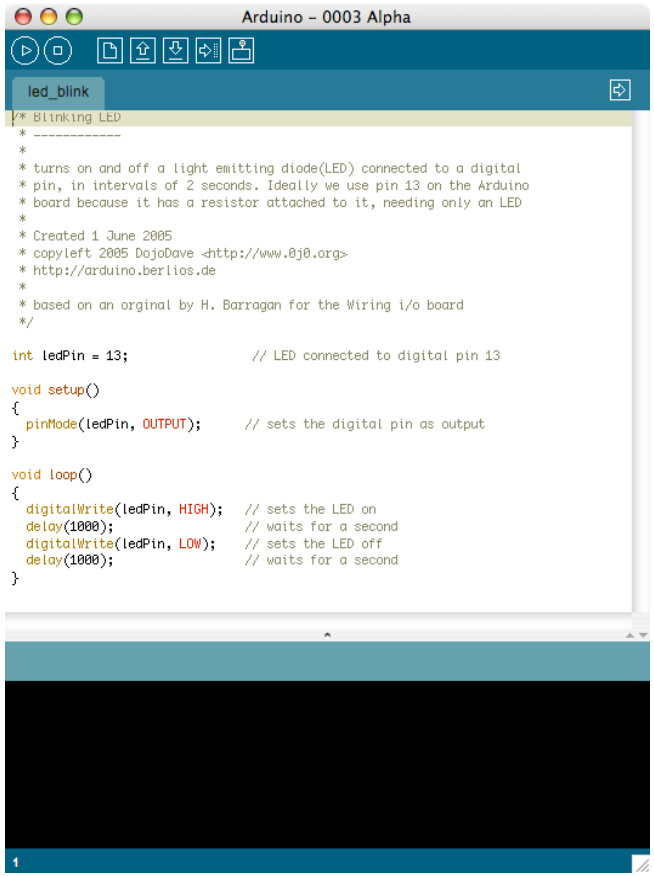

Figure 2. The Arduino environment, showing a simple example of code designed to blink an LED.

The environment and core are closely integrated, so that a single button press compiles a sketch, and another uploads it to the board. The environment performs some basic preprocessing on the user's code, hiding some unimportant syntactical nuisances. The core includes a list of keywords to be highlighted in the environment. Available libraries are listed in a menu within the environment, from which they can be selected for inclusion in a particular sketch. The environment comes with example sketches for basic tasks, allowing users to try some things without writing any code, but also providing simple and clear references of the Arduino language and functions.

The hardware and software, too, are designed to work together. The software need only support the few possible hardware configurations and work only with the bootloader that is pre-installed on the boards, limiting the number of configurations options required.

Community

Arduino is about more than hardware and software. From the beginning we have tried to encourage a community to form around the project – we gave boards to interested parties and taught workshops to show people how to use them. Schools like ITP at New York University, Goldsmiths in London, Fabrica in Italy, and Universidad Politécnica de Catalunya in Spain saw the potential of Arduino and committed to buying boards, providing security and motivation for expanded production. Arduino has since been used at schools over the world, including the MIT Media Lab, Parsons School of Design, the University of California Irvine, Malmö University in Sweden, Keio University in Japan, the Taipei National University of the Arts in Taiwan, and more. We've discovered many of these cases only by stumbling across them online, showing us that Arduino is open and easy enough for anyone to start using, without needing instruction from us.

Arduino has also proven popular among artists seeking to incorporate electronics and interactivity into their

works. We started with a one-week workshop for artists in October 2005 in Madrid, Spain. Since then we have taught in Canada, UK, USA, Germany, Switzerland, Netherlands, Norway, Italy, Austria, Sweden, Slovenia, etc. Others have used our model and given workshops in places such as Taiwan, China, Japan, Finland, Mexico, Chile, Colombia, and Turkey. Art pieces using Arduino have appeared in the Centre Pompidou in Paris, France, at the Ars Electronica Festival in Linz, Austria, at the Salone del Mobile in Milan, Italy.

The main dissemination tool for Arduino is a website, collaboratively edited by fifteen people from various countries. A publicly-editable wiki provides a space for Arduino users to write tutorials, provide example code, upload circuit designs, and share what they've made with Arduino. In the forum, users can ask for help with a variety of issues: getting the board up-and-running, debugging their code, figuring out which components to use for a particular task, etc. The project is not limited to English either. Twelve editors have worked on creating a Spanish version of the Arduino site, and one has written a long guide for teachers who wish to use Arduino in their classes. There are Spanish, French and Italian boards in the forum, and people have also written tutorials in German, Japanese, Chinese, and many other languages.

Perhaps most satisfying are the many contributions that people have given back to the project. We've already mentioned the publicly-written documentation on the Arduino wiki and the many workshops run by people unaffiliated with the project. Other contributions include libraries of code for specific hardware, custom shields for various circuits, patches to the Arduino environment, even icons and color schemes. While this sort of international collaboration is common in the open-source world, Arduino's dual approach of cheap manufactured boards and open plans have allowed it to flourish in the realm of hardware.

Conclusions

We have presented Arduino, an open platform for electronics prototyping. We hope that Arduino demonstrates the potential of the open-source model to apply to hardware as well as software and shows the value of manufacturing a tool to be used by a wide range of people, not confined to a research lab or those who can afford expensive equipment.

Acknowledgements

Gianluca Martino is the remaining member of the Arduino team, responsible for engineering and manufacturing. Nicholas Zambetti has contributed to Arduino since the beginning.

Citations

[1] Atmel Inc., "Atmel." Atmel. unknown. Atmel, Inc.. 25 Jan 2007 <http://www.atmel.com/>.

[2] Barragán, Hernando. "Wiring: Prototyping Physical Interaction Design." IDI Ivrea People. June 2004. IDI Ivrea. 25 Jan 2007 <people.interaction-ivrea.it/h.barragan/thesis/>.

[3] Barragán, Hernando. "Wiring." Wiring. unknown. University de los Andes. 25 Jan 2007 <http://wiring.org.co/>.

[4] Basic Micro, Inc., "BASIC Micro." BASIC Micro Home. unknown. Basic Micro, Inc.. 25 Jan 2007 <http://www.basicmicro.com/>.

[5] Infusion Systems, "MicroSystem." Infusion Systems. Infusion Systems. 25 Jan 2007

<http://infusionsystems.com/catalog/product_info.php/cPath/21/products_id/91>.

[6] Microchip, Inc., "Microchip." Microchip. unknown. Microchip, Inc.. 25 Jan 2007 <http://www.microchip.com>.

[7] NetMedia, Inc., "The BasicX Family of Rapid Development Microcontrollers." Basicx by NetMedia. unknown. NetMedia, Inc.. 25 Jan 2007 <http://www.basicx.com/>.

[8] Parallax, Inc., "BASIC Stamps." Parallax. unknown. Parallax, Inc.. 25 Jan 2007 <http://www.parallax.com/html_pages/products/basicstamps/basic_stamps.asp>.

[9] Phidgets, "Phidgets - Unique and Easy to Use USB INterfaces." Phidgets. unknown. Phidgets. 25 Jan 2007 <http://www.phidgets.com/>.

[10] Reas, Casey and Fry, Ben, "Processing.org: a networked context for learning computer programming." ACM SIGGRAPH, 2005.

[11] Stanford HCI group, "d.tools: Enabling rapid prototyping for physical interaction design." HCI at Stanford University: d.tools. unknown. Stanford University. 25 Jan 2007 <http://hci.stanford.edu/dtools/>.