University of Alberta

Open Source Hardware: The history, issues, and impact on digital humanities

by

Garry Chun Yang Wong

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Arts

Humanities Computing

Fall 2011
Edmonton, Alberta

Abstract

The history of free and open source software (FOSS) spans the better part of 20 years. We are now seeing the principles of FOSS spread to different media - including, notably, to hardware and its distribution models. Recently, the term open source hardware (OSHW) was defined at the 2011 open hardware summit.

OSHW has the potential to redefine the way that goods are designed, transported, and consumed. Accordingly, researchers in the humanities and digital humanities in particular should pay attention and take advantage of this potential area of research.

This thesis first provides a basic understanding of open source software including its history, definition, and business models. Next, it explores how open source ideas are applied to hardware as well as the history, critical issues, and future of OSHW. Finally, it presents a research project undertaken as a case study for how the digital humanities may use OSHW in research.

Acknowledgements

I would like to thank my co-supervisors Dr. Geoffrey Rockwell and Dr. Scott Smallwood, as well as my third examiner, Dr. Sean Gouglas. Without their constant guidance, supervision, and time this thesis would not have been completed.

In addition, I would like to thank Huiwen Ji, Calen Henry, and Patrick Von Hauff for their amazing work on the AXCase.

The development of the AXCase began in a Humanities Computing and Industrial Design graduate class. Calen Henry, Huiwen Ji, Patrick Von Hauff, and I continued to work on the project after the completion of the class under the supervision of Dr. Geoffrey Rockwell.

The design of the case was carried out by Huiwen Ji. Huiwen was responsible for almost all of the designs, sketches, and drawings of the case. Huiwen was supported by the rest of the group through discussions, defining goals, and suggesting solutions to problems. In addition, Calen was responsible for much of the documentation and research while Patrick was in charge of programming an example game for the device. My contribution mainly involved building the electronic hardware.

This project spurred my interest in open source hardware and served as the inspiration for this thesis.

Open Source Hardware: The history, issues, and impact on digital humanities

Table of Figures

Introduction

"Open-Source hardware" (OSHW) is a relatively new term that has begun to gain attention. Most are familiar with the open-source software movement but have never heard of the former. The first time a person hears the term, it may seem counterintuitive to think of hardware as possibly being "open-source". However, upon further contemplation, the concept begins to make more sense. It is easier to understand when objects are thought of not as physical artifacts but blueprints, plans, or recipes. These types of documents, once digitized, can be easily modified and then transmitted from one person to another over the Internet just as software is. However, there are many questions that remain unresolved and many obstacles to overcome before this movement gains widespread adoption.

This thesis will serve to answer some of the interesting questions regarding OSHW which have presented themselves:

- What do the terms "free" and "open-source" mean?
- How can these ideas be applied to physical objects?
- Why is OSHW interesting to the digital humanities?
- What will the OSHW movement mean for the humanities and for society as a whole?

First, this thesis explores the meaning of "free" and "open-source" within the context of software. In order to establish a baseline understanding of the terms, it examines their similarities and differences. There is also an exploration of key aspects of free and open-source software (FOSS) such as the roots of the philosophy, its definitions, its impact on computing and the web, and successful business models centered on open-source software.

Second, this thesis explores the application of FOSS principles to hardware and contrasts OSHW with FOSS. A part of this exploration involves an investigation as to where these principles fit into the manufacturing and dissemination of physical objects. Furthermore, a discussion of where OSHW came from, its current state, and future challenges facing the movement takes place.

Finally, this thesis will discuss, as a case study, an open-source hardware research project completed at the University of Alberta known as the AXCase. Through discussion of this project, this thesis makes the argument that OSHW and rapid fabrication can be used to enable research in the digital humanities that was previously very difficult or not possible at all.

Chapter 1: Free and Open-source Software

History of "Open-source"

"Open-source" is the philosophy that promotes access to a finished product's source materials.[1] In addition, the term "open-source" goes beyond this basic definition in that it requires that users should be allowed to use, improve, and otherwise modify an end product freely and without restriction. This philosophy, while most often associated with computer software, is not a purely modern idea.

Sherman, B., & Bently, L. (1999) argue that it was not until the 19th Century that society developed the modern idea of intellectual property rights. It was thought that in order to protect the individual who created a new and wonderful idea, nobody else must be allowed to copy it without paying the creator money. It was argued that without this protection, people would not be motivated to innovate and create.

This argument is contested, however, by the story of Linus Pauling, Francis Crick, and James Watson regarding the development of the double-helix model of DNA.[2] DiBona and Ockman (1999), write that the competition between Watson's team and Pauling's team resulted in keeping both "parties from disclosing all they knew, and that the progress of science may have been delayed, if ever so slightly, by that secrecy." (Chapter 1, para. 1) This possibility for closed-source principles to negatively impact progress is exactly the reason why open-source philosophies are so important. "Open-source" is a counter to the idea that sharing and the freedom of ownership will erode the motivations for innovators to create.

---

[1] See DiBona and Ockman (1999) for an excellent book-length discussion of the open source movement, its history, and future.

[2] See DiBona and Ockman (1999) Ch. 1. for the complete retelling of the story.

The idea of "open-source" played a key role in early computing. Stallman (2010) proposes that, in the early 70s, the status quo was that software was shared for free between creators and other users. As Stallman (2010) notes in his essay regarding the GNU (GNU's Not Unix) project, the early 80s brought about a rapid change in which the computing community eventually adopted a closed-source software model. This closed-source software model was pioneered by Bill Gates in a famous open letter sent to computer hobbyists (Gates, 1976).

Furthermore, Unix, a closed-source operating system, achieved widespread adoption. Unix was developed in 1969 primarily by Ken Thompson (Bell Labs, n.d.). According to Bell Labs (2002), "this operating system was designed to let a number of programmers access the computer at the same time and share its resources."(Overview section, para. 1) As a result of the widespread adoption of this operating system, Richard Stallman began the GNU project. GNU was to be a clone of Unix to encourage adoption with the main difference being that its users were free to run, edit, fix, and redistribute the software as they saw fit. At the same time, Richard Stallman also coined the term "free" to describe the GNU software and founded the Free Software Foundation to manage this term.

Despite being younger, the term "open-source" has gained more mainstream acceptance than the term "free." The meaning of each term is expanded upon in the following section.

Definition of Open-Source Software

Open-source software is now quite common due to highly successful projects such as Open Office, Linux, and arguably best-known, Firefox. Open-source software is a piece of software released to the public under one of many different open-source licenses. The clauses that make up each license are quite complex and unique yet must follow a common guideline

in order for the entire license to qualify as "open-source." This guideline is known as the open-source definition.

The best-known criterion of the open-source definition is that the source-code of the software must be published and available to the end user. As a result, most people know FOSS as this one single criterion. While it is indeed a part of the open-source definition, it is only one of a large number of criteria that must be fulfilled before a license can be called "open-source."

Example criteria include free redistribution, availability of the source code, allowance of derived works, protection of the author's source code, and non-discrimination. The Open-Source Initiative publishes the entire open-source definition online;[3] I have attempted to summarize it here:

Free redistribution:
This criterion requires that the license must not restrict anyone from selling or giving away the software for free or for a fee. The original licensee cannot demand a fee for the selling of derivative software covered by the license. This means that while open-source software can often be acquired for no cost, the source code can be used in software that costs money. Furthermore, the original developer of the software cannot claim royalties on the derivative works using the original source code.

Available source code:
This criterion is fairly straight forward; it simply guarantees the availability of the source code of the software. Without this clause, the software would not really be open-source. It is the most basic and well-known clause of the open-source definition

[3] For the entire definition, see Open-Source Initiative (n.d.e).

Derived works:

The open-source definition dictates that the licensed software must allow for others to create new works from itself. This is one of the most important criteria of the open-source definition; this clause ensures that growth and creativity are encouraged. One of the strengths of "open-source" is that there is a large pool of individuals who are capable and willing to improve the piece of software by redesigning, rewriting, or otherwise creating derived works; without this clause, no improvements would be allowed.

Integrity of the author's source code

The license is allowed to stipulate that the original code may not be modified provided the author allows patch files to be included along with the original source. The license may also request that derivations carry a different name or version number from the original software.

This is in place in order to properly attribute the original author's work. Furthermore, this protects the original author from being associated with anything that he or she may not wish to be.

For example, if an organization whose values conflict with those of the author wishes to create a derivative work, the author will be protected from being associated with that organization since his source code could only be released in a form where new modifications were distinguishable from the original work.

No discrimination against persons or groups or fields of endeavor

This clause is quite self-explanatory. The license cannot discriminate against any person, groups of persons, or specific field, be it genetic engineering, abortions, or for-profit business.

Distribution of license

The license is automatically applied to all users to whom the program is redistributed. This prevents the source code of the software from being concealed through other means such as a non-disclosure agreement. In other words, a non-disclosure agreement would be super-ceded by the open-source license as anyone using or modifying the software has already accepted the open-source license.

The license must not be specific to a product

This clause prevents the concealment of a portion of a program by distributing it separately from the rest of the program to which the original license applies.

The license must not restrict other software

This clause is simple and straightforward. The license must not apply only to one specific product nor restrict other software. Restricting other software in this case means dictating how the user might use other pieces of software. For example, by using Mozilla Firefox as my primary browser, I am not forced to use Mozilla Thunderbird as my email client as well.

Examples of Open-Source Licenses

Example licenses that adhere to these criteria include the GNU General Public License, Mozilla Public License, and The BSD License. These licenses are all considered open-source. However, each one has its subtle and unique attributes that make it better suited for one purpose than for another. The GNU General Public License, which is not tailored to any one piece of software, is the most common. For example, the Mozilla Public License allows the use of closed-source software within the larger body of work while the GNU General Public License strictly forbids this.

Definition of Free Software

The definition of free software is much more succinct than that of open-source software. It is provided by The Free Software Foundation (2010). All free software licenses must protect the following freedoms:

- The freedom to run the program for any purpose.
- The freedom to study how the program works and change it to make it do what you wish.
- The freedom to redistribute copies for free or for a charge.
- The freedom to distribute copies of your modified versions.

Some of these freedoms, such as "the freedom to study how the program works," are the same as those articulated in the open-source definition. However, there are differences between the two factions, which may be contradictory statements.

For example, the freedom to distribute copies of your modifications possibly conflicts with the open-source clause allowing authors to require modifications to be placed in patch files.

Examples of Free Software Licenses

These licenses follow the free software guidelines: Apache License, Version 2.0, The GNU General Public License, and the WebM License.[4] The different licenses differ from each other the same way that open-source licenses differ from one another. Each license serves its own particular type of software best. Also, the GNU General Public License qualifies as both open-source and free software.

[4] See See Apache Software Foundation (2004), Open-Source Initiative (n.d.b), and WebM Project. (n.d.) for the full text of each respective license.

Free Software vs. Open-Source Software

As the GNU project, and other freely shared software, gained a larger and larger following there came to be an increasing need to define the term "free." However, rather than settling on a single unified definition, the discussion among the community that followed resulted in two: "open-source," and "free." Both terms were put forward by various individuals as the definition of "that which is freely used, shared, and modified." The reality is that neither term is perfect; however, both are commonly used terms today. Unfortunately, each term is also misleading. For example, "open-source" implies that the object simply needs to reveal its source materials to the end user to qualify. In the same fashion, "free" implies that the object is without monetary cost. These misunderstandings highlight the fact that neither term is perfect.

The definition of "open-source" is managed by The Open-Source Initiative (OSI), which was founded in 1998 in California (Open-Source Initiative, n.d.a). The definition of "free" is managed by The Free Software Foundation (FSF), founded by Richard Stallman in October of 1985 (Free Software Foundation, 2010).

The relationship of free software and open-source software is that free software includes all open-source software but open-source software is not necessarily free. For example, the GNU General Public License - while accepted under the definition of "open-source" - goes above and beyond the goals of OSI and meets the standards of the FSF. In contrast, the JSON License qualifies as "open-source;" however, it is not free since it dictates that "The software shall be used for good, not evil" (JSON, 2002, para. 3). In this example, the FSF is not satisfied by the criteria laid out by the OSI.

The FSF states that, in addition to considering the points laid out by the open-source definition, it is important to bring attention to the ethical

considerations of freedom. The primary difference according to the FSF is that "open-source" merely considers issues of improving software while "free software respects the users' essential freedoms" (Stallman, 2011, para. 1). The FSF makes clear the idea that a particular piece of free software may very well be of lower quality than closed-source software. Regardless, despite its inferior quality, it is our moral responsibility to use and improve the free software rather than use the better closed-source software. The OSI is of the opinion that the collaboration of so many different developers will inevitably result in software that is better than software of a closed-source nature.

Another issue that Stallman (2011) has with the term "open-source" is that its obvious meaning, that one may get copies of the source code, is misleading. Stallman (2011) argues that the term free software needs to be clarified only once with the explanation "free speech, not free beer" (Common Misunderstandings section, para. 1). Stallman contrasts this with the much longer explanation required for "open-source" which must explain that an accessible source code is only part of the definition.

One might question why the FSF emphasizes wording when Stallman (2010) himself admits that much of the software that is "open-source" is also "free." However, when one considers the fact that some software, such as digital rights management (DRM) software, is specifically made to restrict the users' freedom, it becomes a pressing issue. For example, it is quite common for digital media to include digital rights management (DRM) software which restricts the ability of the end user to use the product legally. Stallman (2011) goes so far as to give the example of "open-source DRM" as software that would be perfectly acceptable under the philosophy of the OSI yet unacceptable to the FSF.

While the philosophy of the FSF is a legitimate concern, for the remainder of this thesis, I will use the term FOSS to refer to both "open-source" and "free" software. Further on, this thesis will discuss the applicability of the idea of free software to hardware. For example, is it possible to apply the term "free hardware" to guarantee basic rights for the end user of a physical object? The idea of free hardware is relevant in the context of repairing electronic devices. Currently, many hardware manufacturers do not allow users to open and tinker with the objects they purchase without voiding a warranty. Another situation relevant to free hardware is where a car dealership requires the user to return to the dealership for service rather than going to any repair shop they wish. However, the freedom to use a piece of hardware might imply too much, such as the freedom to do harm with the object. A license that guaranteed the basic freedoms of an end user and their hardware might be beneficial; however, it might simply be too difficult to apply.

FOSS Development Methods

In the early stages of the open-source community, the development of a project involved a select group of highly skilled developers while the remaining people were often regarded as peripheral and relegated to bug-testing. For example, the source code would be released to the public in regular intervals; however, development and testing that was taking place between each release remained within the control of the core development team. Each release had to be planned, developed, tested, and fixed by a small group of individuals. In essence, this was a top-down development model similar in ways to the development model used by closed-source software developers. This model of development is functional, as evidenced by the development of great pieces of software such as Emacs; however, it does not fully leverage the large numbers of people interested

in any given project. Referred to by Eric Raymond as the "cathedral method," it was the predominant development model of early open-source software. The cathedral method was thought to be necessary when managing highly complex projects. Furthermore, early open-source developers were hesitant to release early code, fearing that their user base would be dissatisfied with code that was buggy and untested. Eric Raymond (2002), in the same essay, credits Linus Torvalds with turning his thinking upside down.

The development of the Linux kernel established a new development model that takes advantage of the large and highly skilled user base that open-source software attracts. Rather than having a small group responsible for the main development of a complex piece of software and releasing after long intervals of development, Torvalds often had multiple releases in one day with many, many users contributing to each release. The software users undertook a much larger portion of development and testing than usual while Torvalds carried out the tasks of planning and gate-keeping. Raymond dubbed this new approach the "bazaar method," in comparison to the common "cathedral method." It capitalizes on the characteristics of the users of open-source software in order succeed. Specifically, open-source software users tend to be tinkerers; they are eager to contribute.

According to Raymond (2002), there are a few requirements for the bazaar model to be successfully used. First, a project cannot start by using the bazaar method; users must be given something that is running in order to be able to test it. Second, there must be a critical mass of skilled users who are willing to contribute to a project. Third, communication channels must be incredibly efficient in providing contributors clear terms of reference in order to minimize duplication of effort among the user base. In addition,

as with all open-source projects, projects utilizing the bazaar method must keep excellent documentation and follow coding conventions.

If successful, the bazaar method provides a number of advantages over closed-source software and even FOSS using the cathedral method. For example, the software developed under this model tends to be more robust and stable. When it comes to fixing bugs, the amount of person-hours available to solve a problem using the bazaar method is far greater than when using a different model. With this increased number of person-hours, the chances are high that someone with the proper experience will solve the bug with little effort. This is summarized by Raymond (2002) as Linus' Law: "given enough eyeballs, all bugs are shallow" (Release Early, Release Often section, para. 12).

Furthermore, the projects that come out of the bazaar model have lasting power. A bazaar project can carry on, whereas a cathedral project has a high chance of fading away when the core developers move on to other projects. Since the bazaar method of development decentralizes development, even if the developers move on, there will ideally still be a large group of people ready to fill their shoes. Granted, there may come a day for any project when nobody wishes to work on it, at which point it will die. Nevertheless, the point is that the bazaar method reduces the likelihood that a project is abandoned.

Finally, the bazaar model leverages a volunteer workforce. The individuals working on bazaar projects will be doing so for some form of non-monetary gratification. Thus, projects do not need a project manager who pokes and prods to motivate workers. These individuals are motivated because they are at play. Also, these projects are able to draw on the whole world as its talent pool by leveraging the Internet. Furthermore, those who produce sub-par work are overlooked in favor of

those who produce better work. This phenomenon results in bazaar projects having a diverse, highly talented, motivated, and skilled set of contributors.

One of the most common criticisms of bazaar-style development is the notion that only the sexiest or most attractive tasks will attract participants. However, this has proven to be the incorrect. According to Eric Raymond (2002), the open-source community has produced wonderful documentation despite documentation being an almost universally abhorred task among programmers. Raymond suggests that this is due to the fact that open-source communities function because every participant wants recognition. This recognition, called "egoboo" (ego-boosting) by the community, is the currency of the open-source development community. In a capitalistic fashion, the selfish desire for egoboo in each and every contributor is the utility function that fuels the order, progress, and self-correction of an open-source project. It is also egoboo which motivates contributors to complete the less enjoyable tasks associated with software development such as documentation.

Beyond increasing ego, two other general categories of motivators exist for open-source software programmers. First, Hars & Ou (2001) identify internal motivators such as altruism, community identification, and joy derived from the activity. Second, they also mention external motivators such as gaining future rewards by enhancing one's own skill levels, increasing revenue from related projects, marketing oneself, and gaining peer recognition.

While very successful for Linux and Fetchmail, the bazaar method is not applicable to every project. For example, a project that is mundane to most users will not be as successful at gathering participants. For example, Lindal (1999) proposes that those who are capable of creating personal

finance software would likely find the end product useless, while those who would need such an application would not have the expertise to develop one. Furthermore, relative to the cathedral method, more effort is required in order to organize bazaar development. Without proper organization, there will be much duplicated effort among the user base, reducing the overall efficiency of the system.

"Open-source" as Business

While the success of "open-source" was at one point heralded as the future of computing,[5] the reality is that it is not simple to make money producing open-source software. However, it is not impossible and there are many commercially viable open-source projects that thrive.

The open-source definition allows a number of ways for companies to make money producing FOSS. There are many less inventive ways that a business might leverage "open-source" such as offering a limited piece of software under an open-source license while selling the full version for a fee. Other companies may also put advertising into their software. These models of business are often successful but somewhat uninteresting to discuss and straightforward to understand. What is more interesting are the following: hybrid business models, software with services, and partnerships.

Hybrid model

The hybrid business model is a model in which a piece of open-source software is sold for money after adding closed-source features. Normally, after a period of time, these propriety features are released back into the open-source community. For example, OpenOffice.org is the open-source version of Oracle Open Office, which is the closed-source version.

[5] See Shankland (n.d.), McCarthy (1999), and Moore (2001) for more commentary on the attitudes towards open-source software in the 90s.

Traditionally, the piece of software that is sold for a monetary value is sold as an enterprise version.

Example - Apple and Darwin

An example of this hybrid system working exceptionally well is Apple's OS X operating system. In 2001, Apple released an open-source version of the operating system called Darwin at the same time as the company publicly released Mac OS X Cheetah. Along with every new release of the Mac OS X, there has been a corresponding release of Darwin. The effect is somewhat cyclical: improvements made to Darwin by the community are implemented in Mac OS X, while new improvements to Mac OS X are eventually given to the open-source community in Darwin as new versions are launched. Darwin includes core components of the operating system such as the kernel, file system, and basic I/O functions.[6] However, it does not include a GUI, Carbon or Cocoa development APIs, nor does it feature the Quartz windowing system. These features are only found in Mac OS X, and arguably, for the vast majority of the population, they are what the words "operating system" bring to mind.

Examining Apple's Mac OS X and Darwin project, it is possible to see the benefits of this business model:

- Apple has saved a considerable amount of time and money during the development of OS X incorporating elements of ready-made open-source software. The savings theoretically should be passed on to the consumer by a reduction in price for OS X.
- The end user benefits as UNIX operating systems are known for security and stability. However, they are not known for their ease of use, which Apple has improved.

[6] See Russell (2005) for a full description of the components of Darwin found in OS X.

- Some improvements that Apple has developed are released back to the community. Darwin, Bonjour, WebKit, and other projects are a significant contribution to the open-source community.[7]

While this sounds wonderfully symbiotic, it should be acknowledged that there has been a growing body of discontent with the level of contribution Apple has given back to the open-source community (Gruber, 2006). There is much disagreement, and as Jon Buys notes, the relationship between Apple and the open-source community is a complicated one (Buys, 2010).

Services with software model

This business model incorporates open-source software in a very easy-to-understand way. The software provided by the company is usually fully "open-source" and freely distributed. However, services that are tied to that product such as support, large-scale deployment, documentation, or maintenance are sold for profit.

Example – Canonical and Ubuntu

Ubuntu is a version of the popular open-source operating system, Linux. It was first released by Canonical in 2004 intending it to be a user friendly version of Linux (Canonical, n.d.a). It has since grown into one of the most successful distributions of Linux. Ubuntu as with many other versions of Linux has no cost for the majority of individuals. However, Canonical generates revenue by providing support and services for both enterprise and commercial Linux users.

Canonical, unlike most services with software companies, offers support not only to large enterprises but also to the individual user. The average individual is able to sign up for courses and training related to Ubuntu use through affiliated training companies.

[7] See Apple Inc. (n.d.a), Apple Inc. (n.d.b) for more of Apple's contributions to "open-source."

In addition, Ubuntu has begun partnering with computer manufacturers in order to create computers which have Unbuntu as the default operating system. Previously, Microsoft Windows was almost always the default operating system found on IBM PC computers. The only other option for computer manufacturers was to sell a computer with no default operating system installed. The other major operating system, Apple OS X, is only licensed for Apple computer hardware which means no other computer manufacturers can legally use OS X in their machines.

Nowadays, some niche market companies[8] exclusively provide systems which users can purchase with Ubuntu as the main operating system. Mainstream manufacturers such as Dell, HP, Acer, Lenovo, IBM, Asus, Toshiba, and Samsung have all produced computers that utilize Ubuntu as the default operating system (Canonical, n.d.b).

While Canonical is not a publicly traded company and does not release their financial results, one can only assume that they are successful based upon the adoption rate of their operating system which is now the most popular distribution of Linux (DistroWatch, n.d.).

Partnerships

Partnering with computer hardware manufacturers is not the only type of business model that can generate large amounts of revenue for open-source software companies. One company which follows this business model is Mozilla, which produces the Firefox web browser. Mozilla, developers of one of the most successful pieces of open-source software in the world, relies on search companies for the vast majority of its revenue.

[8] System76 (n.d.) is an example of one such company.

Example – Mozilla

While many believe that the main source of income for a company like Mozilla is contributions from donors, the majority of revenue for Mozilla is generated from the search functionality that is built into the browser (Mozilla Foundation, 2010). Mozilla is paid a certain amount of money by search providers and other online service providers every time a user enters a search query into the search box on the browser rather than navigating to the search website itself. For example, if you search eBay by using the search box found on the browser rather than the website proper, eBay will pay Mozilla a royalty.

The specific amount paid per search with each partner is undisclosed; however, from the annual audited financial report of the Mozilla Foundation, search providers accounted for approximately 86% of royalty revenue generated in 2009. The total amount of revenue generated through these royalties for 2009 was $101,537,000 (Hood & Strong LLP, 2010). Interestingly, the largest single source of royalty revenue for the Mozilla Foundation provided $81,229,600. In comparison, according to the same financial statement, in 2009 Mozilla received only $50,000 in donations.

This business model, while not unsuccessful, also carries with it a certain degree of risk. Mozilla's search contract with Google was started in 2004 and carries through until 2011 (Mozilla Foundation, 2010). Mozilla is almost completely reliant on this one source of funding. This is a precarious situation as Google has since launched its own web browser, Google Chrome. It is possible for companies such as Mozilla to use the same business model yet diversify the number of partners that they have. Despite this weakness in its funding source, Mozilla continues to be one of the most widely used and well-funded open-source projects to date.

While the three business models mentioned, partnerships with other corporations, selling services with software, and producing hybrid open-source/closed-source software, are not the only business models possible, they are often the most successful when compared with relying on donations. It is sufficient to say that, while it is not easy to utilize, a for-profit open-source business model is feasible.

"Open-source" Today

Due to the success of companies such as Mozilla, Canonical, and Apple, "open-source" has cemented itself as a viable method of producing, selling, and profiting from software. "Open-source" has matured from being an anti-establishment movement in the 80s, to the next big thing in the 90s, to being finally accepted as a business model with long-term viability.

Contrary to the optimistic beliefs of the early 90s, open-source software is far from taking over closed-source software in most areas of computing. However, while falling quite short of supplanting closed-source software, open-source technology is crucial in one of the most important areas of modern computing, the World Wide Web. In fact, open-source philosophies are widely considered the driving force that made the Internet and World Wide Web succeed. Many of the core technologies such as HTML, PHP, Apache, and MySQL are "open-source." These are some of the very building blocks that the web consists of; without them, the web would not exist in its current form.

To this day, there are constant struggles to keep the web free and open for everyone. Currently, the search for a suitably open video codec to include in HTML5, the latest revision of HTML, is a heated debate. Much like the format wars of physical media (for example Betamax vs. VHS), there are two potential digital formats that are competing for inclusion within

HTML5 standards: h.264 and Web-M. While h.264 is royalty free when used for noncommercial purposes, it is still closed-source. An open option, Web-M, is being championed by certain companies such as Google.

The status of “open-source” today is that it is widely accepted, and not only as a viable business model for software. The philosophies of “open-source” are now being transplanted to other fields. Most notably, the philosophy of sharing source, instructions, and materials is being adapted to the world of hardware.

Chapter 2: Open-Source Hardware

Concept of Open-Source Hardware

OSHW is the extension of the ideologies of FOSS to the world of physical artifacts. While the translation from software to hardware is at times a difficult and somewhat ill-fitting one, it has been mostly successful. Balka et. al (2009) found "that open design is already being implemented in a substantial variety of projects" (Summary and Conclusions section, para. 1).

Since the FOSS community is centered around computer software, it is natural to think of OSHW as computing hardware only. However, in the case of OSHW, the word "hardware" encompasses all physical objects including but not limited to furniture, tools, or computing hardware.

Much like FOSS, OSHW is arguably not a new concept. The designs of many objects are shared openly and freely on a daily basis. Unfortunately, more and more hardware (especially of the electronic variety) is encumbered with usage terms and restrictions. For example, although it is perfectly legal, a user cannot "jailbreak" their Apple iPhone in order to install software from third party vendors without violating an end-user license agreement and thus negating all warranties on the device (Apple Inc., 2010). Despite Apple's protests, there are many perfectly legal reasons for wanting to install software from a third party such as unlocking additional functionality that has not been implemented by the manufacturer or cellular carrier for whatever reason.

Another example is the fact that a manufacturer may dictate that the user is not allowed to open a digital music player to replace a dead battery.

Like the FOSS movement, OSHW is mainly concerned with these freedoms attached to the ownership or use of an object. FOSS is concerned

with software, while the OSHW community strives for the freedom of users to use, modify, and combine their physical objects as they see fit. The community fights against limitations placed by manufacturers which restrict how property owners use their hardware (McNamara, 2007).

Comparison of FOSS and OSHW

OSHW and FOSS share many similarities, such as the commitment to fostering innovation through sharing and collaboration. However, due to the innate differences between hardware and software, the practices of these two groups differ in a few very important ways.

Transmission

FOSS is most often transmitted through the Internet by either disseminating the source code online or distributing it via peer-to- peer networks. FOSS is also transmitted in a finished form; in other words, the source code is often compiled and packaged along with an installer before distribution. Conversely, transmitting an OSHW project from the developers to the users can be more difficult. There are a few possible ways that OSHW developers can get their products into users' hands: making available documentation only, selling a semi-completed kit, or selling the finished product.

Most commonly, developers simply provide the user with the documentation needed to make the project. This documentation may take many different forms: a step-by-step webpage, an online video, PCB design files, blueprints, or a 3D-model. The nature of the project often guides the direction of the documentation. Documentation for electronic projects will differ greatly from a crochet pattern. Each type of project often has a set standard way of transmitting documentation.

Other projects may be distributed as kits. These kits vary in terms of how complete they are. For example, Arduino clones may be purchased as PCBs only, with no components, or populated with a number of surface-mount components, leaving only the larger through-hole components for the end user to solder. These kits benefit the user as most or all of the required parts are available in one purchase, while the developers save time by not having to assemble the product.

Finally, OSHW projects are also sold as completed products. Monomes, Arduinos, RepRaps, and Makerbots are all available in a completed state. In addition to being sold by the original developer, others may also sell identical copies of the original or clones with different names and features.

Making OSHW

If FOSS is not acquired by the user in a convenient form such as an installer, the user will have to make the software themselves. A straightforward one-line command is all it takes for the end user to make software from source code. In other words, the "making" of FOSS is trivial. By comparison, making an OSHW project is nontrivial. The user must possess a certain set of skills that change from one project to another. Sometimes, this is quite obvious; for example, an open-source crochet pattern will require crocheting skills in order to be completed. If an OSHW project includes electronics, it is a relatively safe bet that the project will require basic soldering skills. Other commonly needed skills include basic woodworking, programming, and 3D design. While it is true that the OSHW community strives to make projects as easily understood as possible, a certain minimal level of proficiency in the relevant field is essential to successfully making many OSHW projects.

If someone has the skills to complete a project, he or she must then look at purchasing all of the required materials. Aside from re-purposing old

objects, this usually cannot be done for no cost. Furthermore, the tools required to complete OSHW projects can be extremely expensive. A quality soldering iron will cost the user a significant amount of money as can basic tools like hammers and screwdrivers.

While making software is a relatively quick process, usually taking from a few seconds to a few minutes to complete, finishing OSHW projects can take a few days to a few months. This is a significant time investment that requires much patience when something goes awry. Conversely, due to this huge investment of time, money, patience, and skill by the user, users also tend to learn more when making a piece of OSHW compared to FOSS. Furthermore, despite all the resources, time, and patience that an OSHW project might demand, it is an enjoyable process for many.

Communities

Community plays a large role in OSHW just as communities do in FOSS; however, the OSHW communities are different in at least one critical aspect. Most FOSS communities are almost exclusively organized online through discussion forums and chat rooms. In addition to this, OSHW tends to build a large portion of community offline in the form of hacker-spaces, clubs, dorkbots, or large conventions. In addition, these offline communities also communicate with each other via online mediums. For example, there are often hacker-space challenges where geographically separate hacker-spaces may compete at a specific task.

Forums and Wikis

Much like FOSS communities, OSHW discussion forums tend to be organized around a singular project. For example, Arduino has its own forum while RepRap has a separate forum.[9] These forums are most often

[9] See Arduino (n.d.d), and RepRap (n.d.a), each project has a separate community and thus seperate forums.

used to provide help to one another, to troubleshoot problems with the project, to distribute news and announcements regarding the project, general conversations, and further development and progress with upcoming changes to the project. In addition, OSHW forums tend to have a discussion board that facilitates the display of new projects. For example, both the RepRap and Arduino forums have sections for people to showcase their original work with each product. As with FOSS, forums are most often the main medium through which individuals participate in open-source hardware.

Blogs and Magazines

In addition to discussion forums, there are many blogs related to open-source hardware. While Linux has a large number of blogs that offer tips and hints related to the operating system, blogs related to OSHW are different.

New projects are often featured on these blogs, which include Hack-A-Day, Make.com, or LifeHacker. These blogs are a primary way for people to find out about new projects that they might like to attempt or take part in. Along the same vein, there are also print publications, such as *Make Magazine*, devoted to open-source hardware. This quarterly publication is dedicated to featuring new hardware projects including guides, instructions, and pictures. *Make Magazine*, published by O'Reilly, has been in existence since 2005 as a "hybrid magazine/book... or mook" (Make Magazine, n.d.).

These blogs and magazines do not focus exclusively upon OSHW; a project featured on any of them might not be licensed explicitly under an OSHW license. Furthermore, these publications also do not explicitly advertise themselves as OSHW blogs. However, they are devoted to the

idea of "hacking" and do-it-yourself (DIY) culture. The hacker and DIY demographic is a critical contributor to OSHW.

Hackerspaces and Dorkbots

Hacker-spaces are "community operated physical places, where people can meet and work on their projects" (Hackerspaces, n.d.). Though not necessarily associated with OSHW, hacker-spaces are often extremely supportive of the OSHW initiative as "open-source" legally enables hackers to play, make, build, edit, and hack. Members of hacker-spaces around the world have endorsed the OSHW Definition.[10]

The majority of these organizations are nonprofit co-operatives in which members pay dues in order to pay for the space rental, tools, and other expenses.

Most of these spaces strive to involve themselves in the communities that host them. For example, after the March 2011 earthquake and tsunami, the Tokyo Hackerspace had contributed to the relief efforts in their own way such as creating and giving away 150 highly affordable solar powered LED lanterns, providing Geiger counters, and organizing donations.[11] Hackerspaces have the potential to be a new form of the community center where people of all walks of life can come in and take part in classes or use available tools (Davis, 2011).

Dorkbot is another interesting loosely knit organization, the goal of the organization is:

> To create an informal, friendly environment in which people can talk about the work they're doing and to foster discussion about

[10] See Freedom Defined (n.d.a) for a full list of individuals and groups who have endorsed the definition.
[11] See Akiba (2011) and Baichtal (2011) for more about what the tokyo hackerspace did to help with the tsunami relief efforts.

> that work; to help bring together people from different backgrounds who are interested in similar things; to give us all an opportunity to see the strange things our neighbors are doing with electricity. dorkbot isn't really a forum for formal artist talks or lectures, but rather a chance for diverse people to have friendly conversations about interesting ideas. (Dorkbot, n.d., para. 2)

Again, while not necessarily affiliated with nor restricted to OSHW, the ideals of sharing, openness, and discussion are very central to the goals of Dorkbot. A Dorkbot is definitely one venue where an OSHW project could gain contributors.

History of OSHW

While at first glance the OSHW movement may seem to be a new phenomenon, the modern movement is actually the latest wave of a movement that traces its roots to the same period and location as FOSS: the mid-seventies, in the Silicon Valley-based Home-brew Computer Club (HCC).

Graham Seaman (n.d.) divides the history of OSHW into two separate waves followed by a dark age which ultimately led to the present period.

The First Wave and the Home-brew Computer Club

The HCC was an early computer hobbyist group that had many members who went on to establish highly profitable computer companies; for example, the founders of Apple Inc., Steve Jobs and Stephen Wozniak, were both members. The club was led by the designer of the first portable computer, Lee Felsenstein (Wozniak, n.d.). The Home-brew Computer Club first met March 5, 1975. This meeting, sparked by the release of the Altair 8800 computer kit, was meant to be a forum where people could

"exchange information, swap ideas, talk shop, help work on a project, whatever" (Homebrew Computer Club, 1975).

Eventually, the HCC became a breeding ground for publicly shared computer designs and schematics along with software. While the term "open-source" was not used, it was quite customary for computers to be designed collaboratively between individuals. For example, Wozniak (n.d.) states that "schematics of the Apple I were passed around freely, and [he'd] even go over to people's houses and help them build their own." Another example of a HCC member designing hardware according to open philosophies was Lee Felsenstein (Crosby, 1995). Felsenstein was heavily influenced by the ideas of Ivan Illich's Tools for Conviviality (Crosby, 1995).

The computer terminal designed by Felsenstein, known as the Tom Swift Terminal, was a computer terminal that allowed the user to maintain, modify, and experiment with the hardware in order to fit the user's individual needs. More importantly, the user could keep adding modular parts to customize the system to suit whatever need the user had.

Felsenstein believed that the Tom Swift Terminal and computing in general would benefit in a number of ways. First, computing hardware would be liberated from corporations that controlled computing hardware and software; people would be free to do what they wanted with their computers. Second, the user would be the expert. In other words, if a machine broke down, rather than having to find support from a company that may or may not be local, users could band together and fix each other's machines. Finally, the user could customize the hardware to perform whatever task they saw fit. Unfortunately, the Tom Swift Terminal was never built. However, the basic philosophy behind the system still lives on in OSHW today. With regard to the HCC, while it and

others like it flourished for close to a decade, as the complexity of the hardware increased with time it became more and more difficult for the home-brew communities to keep up. In the beginning, the home-brew system designers relied on incorporating commodity integrated circuits that were well documented into their designs; this could be done with relative ease by a single person hand-drawing schematics and designs. However, as time passed, it became the case that the integrated circuits that were available were becoming too complex. Designing systems around these integrated circuits became hard enough that the number of people who had the skills to design modern systems using the older methods of hand-drawing schematics was severely limited. The individuals with the expertise and software required to work with these newer integrated circuits were separated geographically and isolated from one another. This led to a slowdown in the OSHW movement.

The Second Wave

The solution to this issue resulted in the second wave of OSHW development. The universities and institutions began developing two key technologies that would re-enable users to share hardware designs. The first was ARPANET, a precursor to the Internet that allowed a large number of individuals to communicate, share, and interact. Graham Seaman (n.d.) argues that this is the first instance of a bazaar development method as it allowed the community to build up a pool of highly experienced hardware engineers.

The second technology consisted of simplified and open methods called the Multi-Project Chip Service (MPCS) (Conway, 1982). Essentially, MPCS allowed a large number of different chip designs to be produced on one expensive silicon wafer making the production of integrated circuits affordable once again. Combined, ARPANET and MPCS technologies

allowed users from different geographical locations to send designs to a central manufacturer and have them produced without having to pay for a full run of chips on the entire wafer.

As the MPCS method spread, universities also began to develop design automation software. These types of software automatically laid out circuit designs in the optimal configuration making chip design significantly easier. Berkeley and Stanford were two of the most notable schools which developed state-of-the-art automated chip design software which was also "free" and "open-source" (Seaman, n.d.). These technologies developed in academia allowed users to once again freely design and create their own integrated circuits. Unfortunately, this period would not last long, as beginning in the early 1990s, closed-source software began to be established as the standard development model.

Dark Ages

Seaman (n.d.) concludes the history of the OSHW movement with the early 1990s. He states that along with the rise of closed-source software, open-hardware also came to a stop. Rather than continuing to develop open tools for the creation of hardware designs, universities and corporations began to focus on closed-source software. Furthermore, the hardware itself again increased in complexity; this time it matured into multi-layered PCBs with increased miniaturization and vastly more complex designs. These advances, which led to a great boost in desktop computing power, also signaled the end of the beginning for OSHW.

Modern OSHW

From this point onward, the development of OSHW has taken two separate paths.

While large-scale OSHW development was no longer commonplace beginning in the 1990s, there have always been small pockets of do-it-yourself hackers who have pursued hardware design as a hobby. The first branch arose out of these remnants of the hackers from the 70s and 80s. The second branch has risen up due to further advancements in technology -- specifically, due to the rise of low-cost field-programmable gate arrays (FPGAs). These FPGAs are general purpose logic chips that may be programmed to perform different functions after physical production. In the case of OSHW, FPGAs are often used to prototype application-specific integrated circuits (ASIC).

While there is justification for discussing the OSHW movement as two separate branches, I must be clear that there is only one OSHW community. The definition of OSHW applies to both groups in that they are striving for the same goal: the freedom to use, modify, and distribute hardware. However, their participants, source materials, transmission mediums, and end results are different.

Hacker Branch

The first branch concerns itself mostly with home-made consumer level finished products. This is the modern hobbyist electronics market; quite naturally, this community seems to be embracing OSHW quite readily.

The hobbyist and DIY market seems to be concentrating on producing new types of finished products with relatively older and cheaper technology or with no electronics at all. These products rely on old technology, such as 8-bit micro-controller chips, to create “high tech” objects such as 3D printers, interactive kiosks, CNC routers, and robots. The components such as the integrated circuits, computing cores, or capacitors that make up the project may or may not be “open-source” themselves.

The information that is freely shared includes CAD files, PCB schematics, software, and written instructions. As the name suggests, the output of this community is most often finished products. However, some of these objects are then used to create other objects. For example, an Arduino is a finished product in and of itself; however, it is then used to create something like an energy use monitor by the end user.

The participants in this branch of OSHW often self-identify with the name "hacker." This term is used in the old sense of the word hacker as someone who explores "the limits of what is possible, in a spirit of playful cleverness" (Stallman, 2002). This sense of playfulness permeates the whole of OSHW. Often, the projects that arise out of this branch, such as the "exploding high-five glove," exemplify playfulness or silliness.[12]

An important characteristic of this branch of the OSHW community is the ethos of inclusion. The projects tend to be well documented, and step-by-step guides are often available. The community of this branch lowers the barrier to entry for as many people as possible. While this is not necessarily true for all projects, it has been my experience that it is more often than not the case. The participants attempt to do this through a number of methods including publishing detailed instructions, selling kits at different levels of completion, or even simply offering to sell the object as a finished object.

Example - Arduino

The Arduino, released in 2005, is a prime example of hacker OSHW (Osier-Mixon, 2010). It is an open-source, affordable micro-controller development board released under creative-commons. Simple, affordable, and easily understood technology is applied by the Arduino to solving

[12] See Skipp (2010) for details about this particular project.

modern problems. According to the Arduino website, "Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software" (Arduino, n.d.a). The Arduino revolutionized the micro-controller market, which has traditionally been dominated by more expensive and highly complex closed-source products. More specifically, the target market for these micro-controller development boards, which previously has been hardcore hardware and electrical engineers, has been expanded. The Arduino includes in its target market "artists, designers, hobbyists, and anyone interested in creating interactive objects or environments" (Arduino, n.d.a). The Arduino not only exemplifies the philosophy behind hacker OSHW, but has also become highly successful by focusing on basic and open hardware, a low monetary cost, and highly accessible software.

An interesting thing to note is that the name Arduino is trademarked while everything else related to the project is "open-source." This is to differentiate the original from derivatives.

Hardware

The core processor of the Arduino is an ATMEL microprocessor. At the time of this writing, the ATMEL ATMEGA328 microprocessor is in the reference configuration of the Arduino known as the Arduino Uno. The Arduino Uno features a 16MHz clock speed, 1KB of EEPROM storage, and 32KB of flash memory (Arduino, n.d.c). While this is much faster than most configurations of the original Altair 8800,[13] it is by no means a complicated piece of technology by modern computing standards which are a few orders of magnitude faster.

[13] See Klein (n.d.) for specs of the Altair 8800.

The "open-source" nature of the Arduino hardware plays a very significant role in its widespread adoption. More specifically, it has allowed the development of Arduino clones known as Arduino-compatibles.[14] These products are compatible to varying degrees. For example, some simply may be programmed with the Arduino development environment, while others may be compatible with daughter-boards known as "shields." Since the name "Arduino" is trademarked, other products are exact replicas of the Arduino itself that are simply re-named. These Arduino compatibles allow different niches to be filled very successfully. For example, the Ardupilot is for piloting autonomous vehicles and the Roboduino is for robotics.[15]

Furthermore, the "open-source" nature of the project allows others to integrate the Arduino design into their products. For example, the Arduino circuit is integrated completely into products like the Makerbot[16].

Finally, since the design of the Arduino hardware is "open-source," the creation of daughter-boards known as "shields" is encouraged and well supported. The shields provide a simple way to add functionality beyond basic computation. For example, shields exist for adding sensors, actuators, input/output, additional levels of logic, multiplexers, or other components. In this way, it is quite simple for a novice to allow an Arduino to control 64 LEDs using a multiplexer without having an in-depth knowledge of the multiplexer itself. All the novice user needs to know is that the multiplexer shield connects to the Arduino and the LEDs into the shield.

---

[14] See Arduino. (n.d.b) for a list of Arduino-compatible hardware.

[15] See Anderson (2009) and De Vinck (2009) for each details regarding each specific project.

[16] See MakerBot Industries. (n.d.) for more information about the MakerBot.

These shields also keep Arduino projects physically clean and organized. A novice will have very little ability to design an efficient circuit that takes up a minimal amount of space. The shield allows the user to design projects without having to worry about efficiently using space. For example, an example shield might take care of the routing between an Arduino and a GPS module. The user does not have to decide which Arduino pins to use, how to wire them, and where to place components in their devices. In other words, shields prevent novices from making their projects disorganized.

Pricing

The entire Arduino board (not just the microprocessor) can be purchased for approximately $30.00 CAD at this time. Competing products usually range from $40.00 to over $100.00 CAD. However, there has been one serious competitor targeted directly at the Arduino. The MSP430 Launchpad, released by Texas Instruments on June 22, 2010, costs under $5.00 CAD including shipping (Szczys, 2010). The Launchpad, while also an OSHW project to a certain extent, has not yet gained as large a following as the Arduino. The MSP430 Launchpad has as of yet seen limited success and serves as an excellent comparison to the Arduino for the purposes of this chapter.

Software

The Arduino development environment is FOSS and multi-platform. In fact, the development environment and language itself is based upon Processing. Processing is a different open-source programming language derived from Java (Processing, n.d.a). This means that all of the software related to the Arduino is free (monetarily and philosophically). In terms of being multi-platform, the Arduino development environment runs on Windows, Apple Mac OS X, and GNU/Linux. The Processing language

(and by extension the Arduino language) is purposely simplistic in order to teach the fundamentals of computer programming to non-programmers. Essentially, Processing hides away many pieces of Java code from the user thus requiring the user to understand and write only the basics. As a testament to the simplicity of the Arduino language, it has been referred to by old-school hardcore micro-controller programmers as "baby-talk programming for pothead[s]" (Provost, 2011).

In comparison, the LaunchPad development environment is closed source. However, a program-size limited version is provided for no cost. The limit on the size of programs that one can write with the free version is higher than what the basic LaunchPad can store. This development platform, known as Code Composer Studio, works by default only on Windows (Texas Instruments, n.d.).

The language that is used with the LaunchPad is standard C++. While powerful, standard, and versatile, this language is intimidating to those that have never programmed before. This is a principal reason why the LaunchPad has not made as large an impact on the market defined by the Arduino. Despite being incredibly affordable, the complicated software environment drives away programming novices such as artists, designers, and hobbyists.

While hardcore programmers may react in a condescending manner towards the simplified nature of Arduino programming, this simplicity has led to wide-spread adoption of the Arduino. Furthermore, the novitiate nature of many Arduino users gives rise to a community that is well accustomed to answering and asking questions, and to being exceedingly supportive of each other in general. These activities result in the development of a large number of quality tutorials and guides for engaging projects (such as laser harps, instruments, gardening monitors,

RFID lock systems, etc.), which in turn gain press attention and greater exposure for the platform.

Example - RepRap

The RepRap is another classic example of the hacker branch of OSHW. According to the RepRap wiki, "RepRap is a free desktop 3D printer capable of printing plastic objects" (RepRap, n.d.c). While 3D printing is on the forefront of computing technology, the RepRap leverages older technology to carry out this task at a low cost. Specifically, the RepRap is a product that employs an Arduino-compatible microprocessor named the Sanguino in order to control the motors and plastic extruders (RepRap, n.d.b). In order to print an object, the RepRap lays down layer after layer of molten plastic in a process called fused deposition modeling. In contrast, industrial machines may use a different process, which may produce more complex models known as selective laser sintering (SLS). The SLS process uses a laser to sinter layers of powered material; thus, SLS can produce models with overhangs, as the overhanging portions are supported by the surrounding loose material.

The importance of the RepRap is not in the quality of the models produced. In fact, some might argue that the quality of the models is far beneath that of those produced by a commercial machine. However, the RepRap can be purchased in a fully assembled form for under $1500 at the time of this writing. By contrast, low-end industrial quality 3D printers cost about USD $20,000 (ZCorporation, n.d.). Furthermore, if one were to build a RepRap themselves, the cost would arguably be even lower (non-assembled kits are currently under $1000) (MakerGear, n.d.).

One contributing factor towards the low cost of the RepRap, and the overall goal of the project, is that it aims to become a self-replicating machine. In other words, it can produce many (but not yet all) of its own

replacement parts (RepRap, n.d.c). Specifically, the plastic structural joints that hold the electronics and metal beams together may be printed. These joints are the only components that cannot be commonly found. All other parts (such as metal rods or electronics) can be easily located.

Professional Branch

While the hacker branch concerns itself with reusing older, slower technology, the professional branch seems to be concerned with open sourcing newer, faster technologies. This branch focuses on creating components such as open-source processors, video cards, and computing cores. For example, OpenSPARC, The Open Graphics Project, and OpenCores are projects related to the development of open-source integrated circuits.[17] These projects rely on the use of FPGAs and the sharing of programs written in hardware description languages such as Verilog (Verilog, n.d.). The end results of these projects are rarely consumed in the same fashion as those of the hacker branch. More specifically, the products created from these projects are not meant for end user or consumer use. Rather, these products are one step further away from being a finished product and meant to be integrated into other products.

While this branch is essential to the OSHW movement and will play a large role in the development of OSHW for the foreseeable future, the products of this branch are less compelling to the humanities since the humanities are more concerned with the use of tools rather than the tools themselves. Thus, while I will briefly discuss the types of projects that are part of this branch, for the remainder of this thesis, I will focus on hacker OSHW.

[17] See OpenSPARC (n.d.), Open Graphics Project (n.d.), and OpenCores (2011) for more details regarding each project.

Example - OpenSPARC

OpenSPARC is a project focused on producing and promoting an open-source 64 bit micro-processor. In contrast to the other OSHW projects mentioned so far, this one stands out in that it has commercial backing akin to larger open-source software endeavors such as Mozilla Firefox or Ubuntu. The project is run by Sun Microsystems, and the processor was developed by them and then later released as an open-source project (OpenSPARC, n.d.).

Sun Microsystems' UltraSPARC T1 microprocessor was released in 2005, and was at the same time renamed OpenSPARC T1 and released as hardware description language along with all the tools necessary to work with the code. Two years later, the second version of this processor was released, the OpenSPARC T2.

OpenSPARC (n.d.) states that the goals of the OpenSPARC initiative include:

- "Eliminating barriers to the next big build-out of the Internet."
- "Improving collaboration and cooperation among hardware designers."
- "Enabling community members to build on proven technology at a markedly lower cost."
- "Encouraging innovation."
- "Fostering the ability to bring bold new products to market."

Sun Microsystems stands to gain from the project in the same way that Mozilla and Canonical do from their respective products. Namely, the communities' improvements to the OpenSPARC processor will be rolled into technology that is used in Sun Microsystem's other products.

Example - The Open Graphics Project

The Open Graphics Project (OGP) produces an open-source graphics card. Currently, they have developed an FPGA based development board which they are both using to develop an ASIC version of the graphics card and selling the development board to other projects in order to fund further development.

Example - The Open Cellphone Project

The open cellphone project is quite appropriately concerned with creating a cellphone that is both open hardware and open software (Open Cellphone Project, n.d.). The creators do not intend to create a cellphone that will compete cost effectively with major cellphone companies. However, for the same reasons a person may prefer a home-brewed beer rather than a commercial beer, someone might prefer a home-made cellphone.

Example - OpenCores

Unlike the other examples, OpenCores is not a project aimed at producing one single artifact. However, it does fill an extremely valuable niche within the OSHW environment.

OpenCores is a repository, much like Sourceforge.net is a repository for FOSS. It seeks to keep a library of processing cores for the development of other projects. For example, open arithmetic cores, communication cores, and video controllers can be found within the OpenCores library.

The OpenCores mission statement provides the reasoning for building an open library of computing cores as follows:

> Today, deep sub-micron designs include many millions of gates in a single application-specific integrated circuit. The number of gates continues to grow, with the result that design times get longer and

> longer. This can result in excessive time-to-market and excessive cost. The technical solution to the problem is to reuse cores and to share the expanding workload of verification.
>
> Currently, cores available for integration are closed-source and must be purchased from established vendors, often at very high prices. These costs can be burdensome, especially for small design teams with limited funding. Closed-Source cores are hard to integrate due to the multiplicity of incompatible design and test tools. (OpenCores, 2011)

These cores, as mentioned in their mission statement, are open for use by other organizations. For example, a group of people interested in creating an open-source DVD player may take a video controller core from OpenCores and integrate it into their product. Essentially, this stops companies from re-inventing the wheel over and over; instead, they can devote resources to developing true innovations.

Definition of Open-Source Hardware

As of Feburary 10, 2011, the first version of the Open-Source Hardware Definition was released (Open Hardware Summit, 2011). In general, OSHW is defined by Freedom Defined (n.d.a) as "hardware whose design is made publicly available so that anyone can study, modify, distribute, make, and sell the design or hardware based on that design" (Statement of Principles section, para. 1). This definition, like the definition of open-source software, will guide the creation of licenses for OSHW. Many of the clauses are influenced directly by the definition of FOSS.

More specifically, OSHW refers to tangible artifacts distributed under licenses that adhere to the following clauses:

- Documentation

- Necessary Software
- Derived Works
- Free redistribution
- Attribution
- No Discrimination Against Persons or Groups
- No Discrimination Against Fields of Endeavor
- Distribution of License
- License Must Not Be Specific to a Product
- License Must Not Restrict Other Hardware or Software
- License Must Be Technology-Neutral

Of these clauses, only three are unique to the OSHW definition, while the remaining clauses are identical to those of the FOSS definition.

Documentation

The definition stipulates that either documentation (eg. design files, drawings, CAD models, written instructions, etc.) or a straightforward (and preferably free) method to acquire the documentation must be released alongside the hardware. Furthermore, this documentation must provide the end user with the ability to modify the files. In addition, the OSHW definition stipulates that intermediate forms such as printer-ready and unmodifiable (eg. locked PDF) files are not a valid substitution. These printer-ready files are analogous to compiled computer code which is not a suitable replacement for source code under the FOSS definition.

Finally, an optional stipulation allows licenses to require documentation to be released in a fully-documented and open file format. The documentation of an OSHW project is analogous to the source code in a FOSS project. This clause is central to any and all OSHW licenses.

Scope

The documentation for the hardware must clearly specify what portion of the design, if not all, is being released under the license.

Necessary Software

If the hardware requires software to operate, the hardware must also release the software under an OSI-approved open-source software license. However, if the software is simple to produce by the end user, the hardware project may simply release pseudo-code to clearly illustrate the hardware in operation.

OSHW Licensing

Example Licenses

OSHW has not yet matured to the point of having very many of its own licenses. OSHW projects have so far mainly relied upon FOSS and copyleft[18] licenses when publishing their work in the absence of its own licenses. For example, the Arduino is published under a creative-commons license while the RepRap is published under the GNU General Public License. On May 25, 2007, the TAPR Open Hardware License (OHL) was published by the Tucson Amateur Packet Radio organization. Interestingly, the OHL predates the definition of OSWH by four years. As a result, the current definition of OSHW incorporates many ideas originally found in the TAPR OHL. The TAPR OHL does not protect software, firmware, or code loaded into programmable devices (Tuscon Amateur Packet Radio [TAPR], n.d.). Rather, the TAPR OHL recommends that a software-oriented license such as the GNU General Public License be used to cover anything other than hardware artifacts. Currently, one major OSHW project is released under the TAPR OHL, the Open Graphics Project (Open Graphics Project, n.d.).

[18] See Free Software Foundation (2011) for an explanation of this term.

In this context, two key definitions should be noted. First, the license refers to documentation as the schematics, blueprints, diagrams, or any other piece of information related to the design of the object. Second, the license refers to the product as the object produced from the documentation. The remainder of the TAPR OHL is quite similar to many FOSS licenses; therefore, I will forgo discussing the clauses in this thesis.[19]

Challenges Facing OSHW

While OSHW is promising, a number of obstacles must be overcome before OSHW will be as successful as FOSS.

Increasing Complexity

Electronic components are once again taking another step forward in complexity to facilitate easier manufacturing of mass-produced products. For example, surface-mount components are beginning to replace through-hole components. Surface-mount components are smaller and have either exceedingly small or no leads. This is an improvement for industrial manufacturing for a number of reasons. Pick-and-place machines work better with surface-mount components than they do with through-hole components. Furthermore, fewer holes need to be drilled through the PCB when using surface-mount components. Also, more devices can be placed onto a single PCB, given that each one is smaller. Finally, surface-mount devices can be placed on both sides of the PCB without interfering with the other side.

While this is perfect for industry and has become the de-facto standard of production, surface-mount components are more difficult for DIY enthusiasts to use. The lack of leads requires a different soldering process which is often intimidating to the novice and more difficult to carry out.

[19] See TAPR (n.d.) for the full text of the license.

The surface-mount components are often much smaller and more difficult to work with by hand without a magnifying glass and forceps than through-hole components. Also, due to the small size of these surface-mount devices, they are more sensitive to the heat produced during the soldering process than through-hole components and thus easier to damage than larger components. Finally, surface-mount components cannot be used directly with a breadboard which makes prototyping devices more difficult.

This increased complexity due to surface-mount components can eventually be overcome with new techniques. A number of resources are available that are devoted to teaching people how to work with surface-mount components. Inventive uses of things like frying pans, toaster ovens, or heat guns has so far allowed the DIY community to continue working with surface-mount components; however, it is likely that technology will again outpace DIY techniques.[20]

The Difficulty of Making

Making physical objects will always be more difficult than making software. This is a major challenge to OSHW as it is much easier to purchase a pre-made object than something that requires assembly. An interview by LinuxToday.com with Richard Stallman perfectly describes the difficulty of making hardware as a barrier to OSHW:

> Freedom to copy hardware is not as important, because copying hardware is hard to do. Present-day chip and board fabrication technology resembles the printing press. Copying hardware is as difficult as copying books was in the age of the printing press, or more so. (Stallman, 1999)

[20] See Seidle, N. (n.d.a), Colin. (n.d.), and Seidle, N. (n.d.b) for the tutorials describing each technique.

However, the outlook for OSHW overcoming this problem is quite positive. A basic solution to this problem is the PCB cutting services that are now readily available to DIY enthusiasts. Online services are available for individuals to produce custom PCB designs without having to etch them by themselves. Batching services have lowered the costs to individuals by combining small orders into one large order before manufacturing them (BatchPCB, 2005).

More intriguing are rapid-fabrication devices that are now becoming readily available. Rapid-fabrication has been used in industry for a number of years to create prototypes and models. Rapid-fabrication is a broad term which encompasses manufacturing techniques such as 3D-printing, laser-cutting, and CNC routing. Many OSHW enthusiasts are interested in developing open devices used to create objects using these rapid-fabrication techniques on a domestic scale.

These open-source desktop CNC machines, the RepRap, Makerbot, and Fab@Home, are getting progressively cheaper, and are constantly improving in quality.[21] Compared to professional rapid-fabrication machines, which are prohibitively expensive, these devices can be purchased for under $1500 at the time of this writing. This type of home manufacturing has been hailed by popular media as the spark that will ignite another industrial revolution.[22]

As these devices mature, the barrier to individuals creating their own physical objects becomes negligible. Hopefully in the future, making a chair will become as easy as pressing a button.

---

[21] See Fab@Home (n.d.), RepRap (n.d.c), and MakerBot Industries (n.d.) for details regarding each 3D printer.

[22] See Randerson (2006), Rowan (2011), Cascio (2009), Daw (2010), Vance (2010) for articles detailing the excitement about do-it-yourself 3D printing technologies.

Licensing vs. Patent Law

In comparing FOSS and OSHW, it is clear that OSHW faces a large challenge related to licensing. The output of FOSS consists of two items: the source code, and the compiled code. Luckily both of these items are covered under copyright legislation. As stated by the Open Hardware Summit (n.d.), this is not the case for OSHW:

> In promoting Open Hardware, it is important not to unintentionally deceive designers regarding the extent to which their licenses actually can control their designs. Under U.S. law, and law in many other places, copyright does not apply to electronic designs. Patents do. The result is that an Open Hardware license can in general be used to restrict the plans but probably not the manufactured devices or even restatements of the same design that are not textual copies of the original. (Open Hardware Summit, n.d.)

This calls into question the validity of the OSHW license movement and is a hotly debated issue in the community. As of this writing, there is no clear resolution to this debate in sight. Richard Stallman provides this quote regarding OSHW:

> Circuits cannot be copylefted because they cannot be copyrighted. Definitions of circuits written in HDL (hardware definition languages) can be copylefted, but the copyleft covers only the expression of the definition, not the circuit itself. Likewise, a drawing or layout of a circuit can be copylefted, but this only covers the drawing or layout, not the circuit itself. What this means is that anyone can legally draw the same circuit topology in a different-looking way, or write a different HDL definition which produces the same circuit. Thus, the strength of copyleft when

> applied to circuits is limited. However, copy-lefting HDL definitions and printed circuit layouts may do some good nonetheless. (Stallman, 1999)

Essentially, Stallman (1999) is arguing that a document outlining an OSHW project can be legally produced from its licensed document, and after making minor cosmetic changes, that new device may be patented. This fact of US patent law fundamentally undermines the principles promoted by the OSHW movement.

While promising, it is clear that there are still substantial issues standing in the way of OSHW becoming as readily accepted as FOSS. However, the OSHW community is constantly working to overcome these and other challenges.

Relationship between OSHW and Academic Research

Scholars have been contributors to OSHW since the second wave in the late 1980s with the development of open-source chip design methodology and software. Higher education's prior involvement has almost exclusively been from a computing sciences or engineering perspective and related to professional OSHW. However, as chip design programs became closed-source and chips gained complexity, most of these individuals moved on to other pursuits and chip design was mostly an industrial endeavor.

Recently, scholars have returned to OSHW; however, these scholars are approaching OSHW from a different perspective. Some researchers at academic institutions are still involved with creating software, hardware, and processes employed in the professional branch of OSHW. MIT, for example, has a small collection of FPGA and ASIC designs licensed under

the MIT open-source license.[23] In addition to this traditional contribution, researchers with a humanities focus are now also being drawn to hacker OSHW.

The entry barrier into developing hardware lowered; as such, researchers, artists, and performers have found new ways to incorporate different types of hardware into their work. Individuals no longer require significant amounts of highly specialized knowledge to create interactive objects. The nature of OSHW ensures that the instructions and training necessary for building objects is accessible to humanists so that they may successfully create an interactive object.

"Open-source" has an interesting relationship with the academic milieu. There are many projects developed by academic researchers which are then opened, and given to the community. Conversely, many open-source projects are incorporated into researcher's teaching and research tools. In fact, open-source projects tend to suit academic researchers well. There are a few examples of academics both benefitting from and contributing to OSHW.

Arduino at Ivrea

The Arduino itself was developed at the Interaction Design Institute, which operated from 2001 to 2005 in Ivrea, Italy. The device was developed by Massimo Banzi, an Italian designer. Banzi taught as an associate professor at the Interaction Design Institute Ivrea during the four years that the institute was open.

The Arduino was designed by Banzi as a tool for his students as he found they were often without a cheap and accessible platform for designing

[23] See Computer Science and Artificial Intelligence Laboratory (2007) for the collection.

interactive objects.[24] Without such an alternative, students would have to rely on what was currently available in the market or simplify their projects.

To be clear, the Arduino is neither the first nor the only micro-controller development platform. However, the other offerings in micro-controller development were and for the most part still are complex, expensive, and hard to find. For example, Kurt (2006) mentions that to get started with the BASIC stamp, an alternative micro-controller, one requires both the controller itself and a development board, which together cost approximately USD $100. It is possible, but more difficult, to program with only the controller and no development board.

Another example is the i-Cube system developed by Mulder (1995), a hardware platform designed to "facilitate, for artists, the design and creation of interactive art." In contrast to the BASIC stamp and Arduino, the i-Cube system consists only of a set of sensors, actuators, and digitizers which are used in conjunction with a desktop computer rather than a micro-controller. Despite this, the i-Cube system can be used for many of the same purposes as a micro-controller such as performance art and interactive installations. The basic digitizer, which sends analog signals from sensors and actuators to the computer, costs USD $90. In addition, the performer must purchase a range of sensors and actuators depending on what they wish to do. These sensors and actuators range in price from USD $22 for a green LED to USD $500 for an EEG sensor.[25]

Another product that could be used by students is the Lego Mindstorms NXT controller. However, it costs USD $170, is difficult to find in stores,

---

[24] See Alaejos and Calvo (2010), especially the first interview with Massimo Banzi where he discusses the motivations behind creating the Arduino.
[25] See Infusion Systems (n.d.) for a complete list of products and pricing.

and requires extensive physical modification in order for the device to be used with non-Lego branded parts.[26] Otherwise, the Mindstorms platform is very easy to use for constructing interactive objects as Lego requires no tools.

The high cost of other physical computing development platforms is beginning to change. Provost (2011) claims that as of 2011, we're now "seeing chip companies start to realize that it's worthwhile to have a more pragmatic pricing strategy."

Without the need for a better alternative for students, the Arduino would never have been developed. The device was created for students; however, designers and non-programmers everywhere found the device useful. Due to the OSHW nature of the device, the OSHW community members were able to contribute back to the Arduino project. This improved the product immensely by producing compatible products, project guides, and providing support.

LilyPad at MIT

The Lilypad is an open-source version of the Arduino tailored for e-textiles development (Buechley, 2006). Leah Buechley, who developed the Lilypad project in collaboration with Sparkfun Industries, states that the goal of the project is to generate interest among those who are uninterested in the traditional culture of computing and engineering:

> Instead of trying to fit people into existing computing cultures, we want to spark and support new ones. Rather than trying to recruit young women to robotics clubs and classes, we engage them in computation through electronic textile clubs and classes—venues that young women flock to with no prompting. We believe that

[26] See http://mindstorms.lego.com/ for more about the Lego Mindstorms platform.

> cultural factors, more than a lack of aptitude or intrinsic interest, make computer science inaccessible and unappealing to many students. By making computation more accessible and building computers that look and feel different from traditional ones - computers that are fuzzy, colorful, and feminine, for example - we can begin to change and broaden the culture of computation. We can begin to get a diverse range of people excited by the ways that computers can be used to build beautiful, expressive, and useful objects that are different from anything that has been built in the past. (Buechley, 2010)

Using this device, individuals are able to create interactive sewable electronics using conductive thread instead of solder and wires. By working with Sparkfun, a commercial business that focuses on "niche electronic products like the Arduino and LilyPad," Buechley and Hill (2010) were able to release a commercial kit version of the LilyPad.

Chapter 3: Using OSHW in the Humanities

AXCase

The AXCase is a project that, like the many different versions of the Arduino, builds off of an existing one. This project is an extension of the Monome and Arduinome projects developed by Brian Crabtree and Kelli Cain. Thus, in order to understand what the AXCase is, one must understand what the Monome is. Essentially, the Monome is a minimalist MIDI controller made up of a number of LED-backlit buttons arranged in a square or rectangular grid. The simplest configuration of the device consists of a square grid of 64 back-lit buttons. The primary feature of this device, which allows for interesting interaction, is that each LED and its corresponding button are decoupled. This means that pushing one button may light up an animated pattern of any number of lights. Furthermore, the lights may turn on independently of any button pressed, relying instead on input from other sources such as a computer or accelerometer.

Given that it is simply a controller, the device functions in conjunction with a computer (PC, Mac, and Linux are supported). The signals from the Monome are sent as serial data to custom software known as Monome Serial which then translates the data into Open Sound Control (OSC) or MIDI signals. The OSC/MIDI data is then sent to a programming environment (either Max/MSP, Processing, or Chuck are the most common)[27] where the program itself runs. For example, a Monome might be used as a physical controller to send signals from a button to a Max/MSP patch. From the patch, the signal would be sent to Ableton Live, which would then synthesize sounds played through the computer's speakers. The Max/MSP Patch would also then instruct the Monome to

[27] See Cycling 74 (n.d.), Processing (n.d.a), and Chuck (2002) for information regarding each programming language.

light up a pattern of lights. No computation of any sort takes place on the Monome.

The device is primarily marketed as a musical controller which, due to its minimal and open nature, is adaptable, flexible, and easily accessible. For example, due to its minimalistic design, it has been adapted to many other uses and functions, such as movie editing, gaming, and design.

Since the creators of the Monome believe in that "'open-source' is commercially viable and mutually beneficial for" their company, they have allowed users to create a totally open-source clone using the Arduino micro-controller to drive the device (Monome, 2011). This clone is known as the Arduinome; parts to build one are easily obtained and the instructions are well documented. This is a marvelous thing for those wishing to experiment with one of these devices since they are difficult to purchase. Monome devices are created by hand, from locally sourced materials, and in extremely small production runs. Fresh batches tend to sell out within days of release. They are also relatively expensive with the most basic devices costing around USD $500.00. Since the devices that our group built were of the "open-source" variety, I will only mention the Arduinome. However, the Arduinome and Monome are essentially equivalent.

A completed Arduinome device can be thought of as comprising two parts: the electronic components and the enclosure which houses the electronics. While the process of assembling the electronic components is exceptionally well documented, the same cannot be said for creating an enclosure. There are surprisingly few resources available to those who wish to build an enclosure for their devices. As a result, users have been known to have put the electronics into make-shift cases such as Tupperware containers, children's lunch-boxes, old radios, cardboard, etc.

The potential of the Arduinome devices lies in its simplicity. To be more specific, since the Arduinome is simply a grid of buttons, it is highly flexible as a platform for experimenting with interaction. It is possible for researchers to re-program these buttons to do whatever required. In essence, the Arduinome has a potential to be a powerful research platform for interactivity. Compared to other objects that can be used to experiment with interactivity (such as the Microsoft Kinect, Apple iPad, or even a PC), the Arduinome is cheap. The electronics and enclosure can be purchased and assembled for under $300 including shipping. While obviously the Kinect and iPad are much more advanced and afford potentially more intriguing interaction, they are also more complex to work with and require highly specialized knowledge. Unfortunately, the Arduinome cannot be useful as a research platform without an equally flexible enclosure.

The goals of the AXCase project were to address these concerns. In general, the project goals were to create a high quality case that is well documented and easy to make using easily sourced materials. It was also our goal to create a case that is as flexible as the device itself. Through this flexibility, we sought to create a platform for experimenting with interactivity.

Designs

Our group felt that the original enclosure designed during the class, the predecessor to the AXCase, was deficient in a number of ways. Specifically, the case made for the class was too bulky, unattractive, heavy, and stained easily (Fig 1.). Many of these problems were a result of the material available during the class, MDF. Most importantly, while this case allowed for the attachment of handles and cases, it was not flexible enough. This case was primarily designed for ease of manufacture. Our

group had highly specific design goals that for the redesign of this case. The following section describes each goal and explains how it was met.

Fig 1. Original design of the AX Case

Ease of Production

The case must producible by submitting schematics to a laser cutting service. A low cost of production is also important.

Solution:

Our group feels that this goal was sufficiently fulfilled by ensuring that the case schematics may be downloaded, edited as needed, and then sent to ponoko.com, or to another laser cutter service, for production.

Also, the entire case is made from 3mm acrylic material. This is an easily sourced, cheap, and fairly durable material that comes in a variety of colors. However, a drawback of this material is its lack of resilience; it can be snapped or chipped if dropped. Furthermore, it has a high-gloss finish that, while aesthetically pleasing, is prone to surface scratches. However, for the purposes of this project, the affordability and availability was of greater concern and outweighed the limitations of the material.

Ease of Assembly

The case must be easily assembled with minimal tools.

Solution:

The case is designed to be snap-fitted together and the electronics are secured to the case using one inch sized 4-40 screws and nuts. Like acrylic material, these screws are very common and are easily cut to any length using a screw cutter. While our original design called for stand-offs to be used to secure the top of the case to the bottom, it was found after production that the case fit together tightly enough to forego their use.

The only tools required in assembling the case are a screwdriver that fits the screws and a wrench or some needle-nose pliers to hold the nuts while tightening the screws. If needed, glue or epoxy may be used to secure the side-walls to the bottom panel. This would be useful for attaching handles and other items to the side-wall.

Flexibility

The case must be able to accommodate the use of attachments such as different handles or stands. It must also provide the user with a means for temporarily or permanently attaching multiple cases together. The case should also accommodate a number of grid sizes and configurations. Finally, the electronic components inside must be accessible for modification without tools.

Solution:

The side-walls of the case are designed so that they may be manipulated for length and height. The unique teeth pattern that holds the case

together is adjustable so that the height and length of the case can fit multiple grid sizes (Fig 2.).

Furthermore, the side-walls have optional features that are on separate layers in the design file. The user may easily turn off any features of the case that they do not want. Optional components include slots for fitting attachments, holes for zip ties, and pegs for fitting two cases together. The pegs may be used, in conjunction with magnets, to create a pair of cases that fit together tightly yet come apart easily with no visible modifications other than the slots.

With regards to the electronics, the Arduinome is "open-source" so that modifications to the electronics are allowed. In addition, the AX Case is easily opened and closed providing access to the internal electronics. As an example, this allows the researcher to add an accelerometer to develop some sort of game based on motion control.

These features allow the experimenter to turn the device into anything they can imagine. It can be a simple response system, an ultra-low-resolution display, a gaming console, a billboard, or just a clock. Finally, the software required is also flexible. Max/MSP and Chuck are useful for programming audio applications. Processing or Flash may be used for developing a variety of applications, including visual recognition systems, data visualizations, or animations.

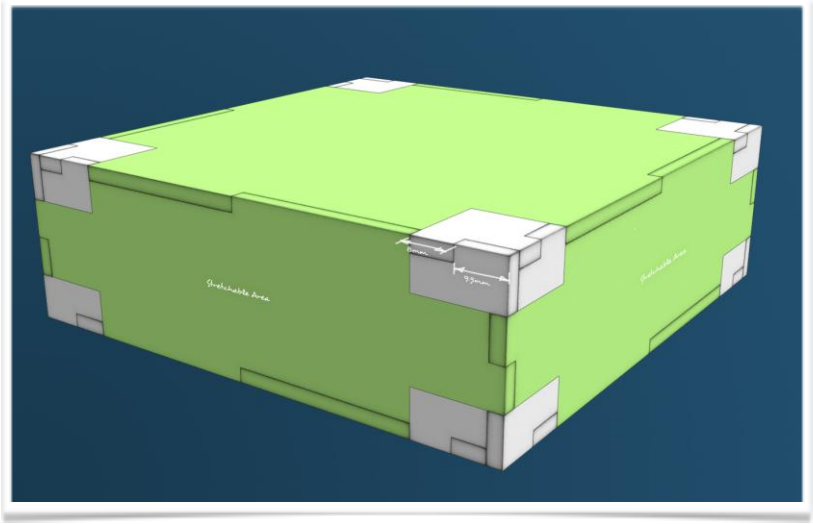

Fig 2. The teeth on the wall edges are designed so that they may be expanded to accommodate a larger grid

Openness

The primary goal related to openness was that the design files must be free to obtain and fully editable by the end users. This means not only that the designs be must readily available, but that users can use an open-source program to use and manipulate the designs.

Solution:

This goal was fulfilled in a number of ways. First, the schematics were released in two separate formats. The first format, Adobe Illustrator, is an industry standard for creating vector graphic files which requires purchasing expensive and closed-source software to edit. The second is the open equivalent .svg format, which can be edited in freely available vector graphics programs like Inkscape.[28]

The files are provided fully unlocked and unsecured, and the user is free to edit any layers or portions that they wish. In fact, optional components are provided on separate layers so that they may easily omit the parts that they do not want by turning off the layers or deleting them.

[28] See Inkscape. (n.d.) for more information regarding the program.

Finally, the files were released with a note declaring that the files were released under a creative commons attribution share-alike license.

Aesthetics

The case must look good (at least to us). Also, it must be made of a fairly resilient material that does not stain easily or sustain water damage.

Solution:

This was the easiest goal to accomplish as it was largely fulfilled through our choice of materials. Acrylic is a fairly attractive material that comes in a large variety of colors and has a nice glossy finish. Acrylic is also much lighter than our previous material. The only drawback to acrylic as previously mentioned is that it is somewhat brittle. However, with normal use the material should be strong enough.

The thickness of the device was minimized which resulted in a case that felt much slimmer and less bulky than the original (Fig 3.). The limiting factor in how thin the device can be is the height of the Arduino and Arduinome shield. In order to make the device thinner, our group would have to significantly redesign the internal components with custom PCBs.

After meeting many of the goals that were set out, the AX Case was deemed ready for release. As with any other project, it was felt that there were areas that could be improved. For example, the case material was brittle and the design of the teeth that held the sides together was a little too snug. While this held the case together so securely that screws were not really necessary, it was more difficult to open the case once closed. Great care needed to be taken when opening the case so that the user did not snap off the teeth.

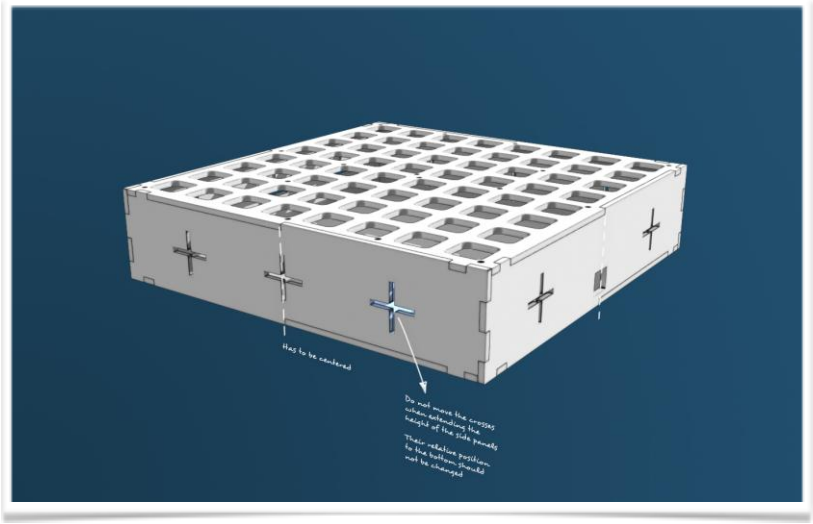

Fig 3. The current design of the AXCase with a slimmer profile and redesigned teeth.

Contribution to the Community

There is a gap in the documentation available to those seeking to design a case. Accordingly, the main contribution of this project to the OSHW community is to make available case designs and documentation pertaining to making an AXCase.

Our group chose to release the schematics as vector graphics rather than in a more sophisticated format so that they may be easily uploaded to ponoko.com. Both sets of schematics used pre-made templates provided by ponoko.com in order to ensure that the files could be easily uploaded for cutting with this service. These templates provide a base to work with which sets parameters such as line weight, colors, and margins to the correct values required by the laser cutting machines.

In order to distribute these schematics, our group created our own website and hosted the files there online.[29] Wordpress was used to create an attractive website that also hosted a number of projects related to the interactivity research group at the University of Alberta.

[29] See AXCase (2011).

This website also included a step-by-step guide for assembling an AXcase after cutting the schematic. Our group had initially intended for this website to be a one-stop-shop for creating an AXCase from beginning to end. However, rather than duplicating already excellent documentation for creating the electronics, our group simply provided references to existing documentation. In addition, we provided small suggestions and tips for issues that our group encountered when building our own devices. For example, we suggested the use of rainbow-colored ribbon cable, which assists in keeping the many wires sorted out.

After completing the website, our group decided that the best way to promote the case was to announce the project to the Arduinome community through the forums.

Reception

The response to the forum post was lackluster with only two users responding. While one user commented that the project looked interesting, he was not yet prepared to make an enclosure. The second user was more interested, but it is ultimately unknown whether or not s/he attempted to make one. Other users may have been interested; however, they did not mention it and our group did not have a system in place to count the number of visitors to our project page.

This first attempt at promoting the AX Case was thus less well received than our group had hoped. There was a noted lack of interest from the majority of the members of the Arduinome community.

Lessons Learned

The lack of interest from the community was as interesting as it was frustrating. Our group felt that there was a niche that needed to be filled; however, there was zero uptake. There might be many reasons for this;

however, I will highlight the most obvious things that could have been done better during the release of the project. First, more could have done been done to promote the device. Our group did not follow Raymond's advice of being open to the point of promiscuity (Raymond, 2002). The official Arduinome community is only one place that where the project can be promoted. There are many other communities that would be interested in projects such as this. For example, instructables.com, hack-a-day.com, and make.com are all blogs that are concerned with OSHW projects not necessarily centered around the Arduinome.

Furthermore, the website hosting the project could have been improved. For example, more pictures of the case or instructions which better illustrate the entire process of creating an AXCase could have been posted on the website.

Another explanation may be that there is not a great deal of need for the AXCase at this time. Perhaps people are perfectly happy with putting their Arduinomes inside of Tupperware or other make-shift containers. Despite this, the project was useful to us for a number of reasons.

The reception of the AXCase reminds us that not all contributions to open-source projects are useful for everyone. The open-source nature of the Arduinome simply ensures that our group has the ability to create a solution for the deficiencies we found in the Arduinome. In this way, "open-source" has fulfilled its promise. It would be wonderful if others found our solution applicable to their own problems; however, it is not the goal of "open-source" to ensure that our solution to our problem is popular or widely applicable.

Furthermore, projects that are not utilized by community still contribute to the overall conversation that pushes the evolution of an open-source

project. By creating the AXCase, our group has made a suggestion for furthering the Arduinome project as a research platform. The Arduinome community, by way of its lukewarm reception, has rejected that suggestion. However, it is the vast number of contributions, both accepted and rejected, that drives an open-source project forward.

OSHW Enabling Research in the Humanities

The AXCase brings to mind three different streams of research relevant and interesting to the digital humanities that have been facilitated by OSHW. First, we could continue on with the AXCase itself and use it to investigate human-computer interaction. Second, we could investigate OSHW and rapid fabrication technologies which may have far-reaching consequences in many aspects of our lives. Finally, we could investigate the implications in humanities disciplines that arise from having a simple way to produce physical artifacts for study.

AXCase and Human-Computer Interaction

The AXCase can and was indeed built for the express purpose of facilitating research and exploration in the field of interactivity. The device can be applied in a number of different scenarios. These are a few different possible projects that are based on the AXCase. Through these possible projects, OSHW and rapid fabrication have allowed us to research interactivity in a new and meaningful way. Previously, this research would have been very cost prohibitive.

Assistive device for the visually impaired

Our group envisioned the AXCase as an assistive device for the visually impaired due to the fact that it is by its very nature a multi-modal device. Zagler and Panek (1998) note that a "strategy of multimodal icons where images, written text, spoken messages and sounds can be combined at will, especially accommodates cognitive and visually impaired users who

cannot interact with graphical information or encounter difficulties if feedback is given only via the visual channel." As Yu et al. (2005) note, "by rendering the spatial visual information via the multimodal interface, visually impaired people are not only informed of these regions of interest, but are also guided to them by the audio and haptics." Sribunruangrit et al. (2003) Also note that "bimodal perception improves the spatial perception ability and the memory of position during the exploration of forms." All of this points towards multimodal interaction being a key feature of technology for the visually impaired. The Arduinome provides this through high-contrast visual, tactile, and audio feedback.

Currently available assistive devices for the visually impaired tend to be extraordinarily expensive and are often devised for a single use. For example, JAWS, a screen reading software costs $900 while hardware screen magnifiers cost around $2500 and the PAC Mate, a PDA for the visually impaired, costs between $2400 and $5500 (Freedom Scientific, n.d.). Furthermore, most of these devices are extensions to existing hardware originally geared towards sighted individuals. While that is beneficial, as it allows sight impaired individuals to use normal technology, our group positioned the Arduinome as something that a general purpose controller tailored specifically to sight impaired individuals might look like.

In order to prove that the AXCase is useful as an assistive device for the visually impaired, we could carry out a number of different studies. We must be able to show that the AXCase is better than (or at least equal to) existing alternatives and available at a lower price. As an example, we could compare response times between the AXCase and a non-multimodal high resolution device such as a touch-sensitive tablet

computer. We would simply light up a square within a grid for each device and measure the time it would take for a visually impaired individual to reach out, and touch the lit square.

If the response times for the AXCase were statistically shorter, then we have suggested that it takes less time for users to respond using an AXCase. If the response times are statistically equal, we have at the very least shown that a cheaper alternative to expensive high-definition tablets exists.

Beyond this rudimentary testing, we would have to develop software that affords specific functions in practical applications. However, the testing principle would be the same. First, we develop our prototype. Second, we then test our prototype against an existing product. Finally, we statistically analyze our results and draw our conclusions.

A general purpose tactile game platform

As part of the original project, in order to demonstrate possible applications of the AXCase, we prototyped a simple Space Invaders clone for the AX Case. The player can move only left and right along the bottom of the AXCase; movement is accomplished by pressing the left or right "wings" on the player's spaceship. Firing a projectile is accomplished by pressing the button making up the nose of the ship.

The game has two possible modes: real-time, and turn-based. The real-time mode plays like a normal arcade game where enemies move around independently. When playing the turn-based mode, the enemy ships only move once for every time the player presses a button. This allows the player to control the speed at which the game flows; this is a feature that could be very helpful for the visually impaired or for children who are not yet able to handle a fast paced game.

This turn-based mode may serve as a great training tool that will develop logic skills, visual skills, and hand-eye coordination. However, in order to really make this claim, further testing is necessary.

Game Jam Development Events

As a way to generate more game prototypes for the AXCase, our group had envisioned that the software would be created through running a game development event.

During the events, programmers would submit games developed for the AX Case into a contest. Not only could the output of these sessions be used for further research projects, but we could also study the creative process that takes place during the development of games. We could perhaps gain an understanding of the specific steps game developers take when attempting to make their games fun for their players.

Open-Source Hardware and Rapid Fabrication as Phenomenon

This project and resulting thesis serve as an over-arching documentation of the current state of OSHW. However, the work in this field is not complete. OSHW along with rapid fabrication will likely revolutionize the way individuals produce, consume, and distribute physical objects the same way that FOSS has revolutionized software. However, this will not be a painless process.

The first area worthy of research is the legality of OSHW. Before OSHW can truly succeed, we need to answer the question of "How will patent and licensing laws protect open hardware?" The answer is currently not clear and is definitely worthy of further investigation.

Second, OSHW and rapid fabrication, if successful, will have an incredibly wide-spread effect on almost all aspects of modern culture. Will society become hyper-disposable with people printing new bikes every day to

match their outfits? Or perhaps will millions of postal, assembly line, and logistics employees suddenly find themselves out of work? On the bright side, will third-world and remote populations find themselves more capable of producing their necessary goods? Currently, most printed objects are made from plastics, how will this technology affect the environment? While it is true that current 3d-printing technologies are limited, these are all questions that are worthy of exploration.

Fabrication in the Humanities

Closely tied to OSHW is the advent of rapid-fabrication and its possible applications in the humanities.

Turkel (2008) provides a few arguments on his blog as to why this new technology might be important to the humanities. These arguments are paraphrased as follows:

- We can't predict the future: Individuals in the early age of computing who struggled to gain individual access to computers were thought to be simply pursuing a hobby with no real application in the humanities. In the same vein, it is too early to say that hobbyists who are interested in personal rapid-fabrication are engaging in a superfluous activity with no academic application.
- Mind and hand: Making is an acceptable path to knowing. Making has also been a part of the humanities for many people in different periods (Archimedes, Da Vinci, Vaucanson, etc.).
- Historic experimentation: Until recently, researchers in fields such as experimental archaeology required direct access to their physical materials in order to study objects. Now, with rapid-fabrication and 3D scanning, we can easily digitize and replicate objects. This process is akin to the digitization of original manuscripts for the purposes of studying them.

- Tangible/haptic history: With these technologies, we can create physical objects that no longer or never before existed in reality. We can hold objects, devices, and tools that we could only previously read about.
- Critical technical practice: An approach to humanities research that involves both crafting and then reflexively studying the crafted object is common in the digital humanities with digital objects. The extension of this practice to physical objects becomes natural with the rise of rapid-fabrication.

These arguments strongly promote the research of or applied use of OSHW and rapid fabrication in the humanities. Appropriately, the strategy taken with the AXCase is an example of "critical technical practice" in the digital humanities as described by Turkel (2008). By using OSHW and rapid fabrication, we were able to craft and then reflexively study the crafted object and use it to open new doors in humanities computing research.

Conclusion

OSHW projects such as the AXCase allow us to explore new areas of digital humanities research such as interactivity, game design, and assistive devices for the visually impaired. Without OSHW, the project probably would not have been successful, because our group would not be able to modify the Arduinome to serve our needs.

Furthermore, the research area of OSHW rapid-fabrication holds considerable promise for a variety of humanities disciplines ranging from modeling dig sights in archaeology to set design in drama.

Finally, OSHW as a phenomenon in and of itself serves as an excellent research area. If OSHW turns out to be as influential in the world of physical objects as FOSS was in the world of software, it will be a tremendously significant movement for the humanities to learn from, document, and study.

It is not known if OSHW will live up to the hopes and desires of the proponents of this philosophy; certainly, there are significant hurdles that must be overcome in order for this philosophy to become widespread and commonly accepted. However, researchers in the humanities and social sciences should be thrilled at the great potential for societal change that OSHW holds.

This rise of OSHW as a global phenomenon is itself significant and worthy of academic investigation. Specifically, OSHW holds much promise in that it may revolutionize the way that humans produce, transport, and consume objects as a global society. Goods would no longer be produced in a central location, shipped, and then finally reach the consumer at a retail store. This may result in a number of both positive and negative changes.

For example, the consumption of fossil fuels would decrease without the need to ship goods. However, a large number of individuals in the shipping/receiving and manufacturing industries may find that they are unemployed.

The value of an item may also decrease when we can easily replace it; we may want a new bicycle every day in order to match the color of our outfit. This would likely have a negative effect on the environment in the same fashion that disposable bottles, cups, forks, and knives have resulted in much more waste.

Another possible outcome would be an increase in the standard of living for third-world, or not easily accessible populations. For example, an Antarctic research station would be able to produce their own goods rather than wait for shipments while third-world communities would suddenly find themselves capable of producing much of their own necessities such as housing or tools.

The OSHW movement can trace its roots to the very beginning of personal computing (and possibly earlier, if one takes into account the sharing of plans for non-electronic objects). The first personal computers were built by hobbyists, and the communities these hobbyists belonged to often shared ideas, plans, and assisted each other with building the physical hardware. As such, the modern application of FOSS principles to hardware is not new. This beginning is merely a re-emergence of OSHW. The coming together of many different technologies such as modern computing power, the World Wide Web, and rapid-fabrication are once again allowing the sharing and modification of hardware designs.

Simply put, humans have always wanted to share what they know and what they use with one another. Previously, this was done in small,

closely-knit groups. The current rise of OSHW is evidence that we now have the means to do so with many people, on a global scale, and with highly complex physical objects.

Works Cited

Akiba. (2011). Kimono lantern kit. Retrieved from http://tokyohackerspace.org/en/project/kimono-lantern-kit

Alaejos, R. (Director), Calvo, R. (Director). (2010). *Arduino the Documentary* [Motion Picture]. Retrieved from http://Arduinothedocumentary.org/

Anderson, C. (2009). ArduPilot main page - DIY drones. Retrieved from http://diydrones.com/profiles/blogs/ardupilot-main-page

Apache Software Foundation. (2004). Apache License, Version 2.0. Retrieved from http://www.apache.org/licenses/LICENSE-2.0

Apple Inc. (2010). Unauthorized modification of iOS has been a major source of instability, disruption of services, and other issues. Retrieved from http://support.apple.com/kb/ht3743

Apple Inc. (n.d.a). Open-source - Apple developer. Retrieved from http://developer.apple.com/opensource/

Apple Inc. (n.d.b). Open-source - Releases. Retrieved from http://www.opensource.apple.com/

Arduino. (n.d.a). Arduino. Retrieved from http://www.Arduino.cc/

Arduino. (n.d.b). Arduino playground - Similar boards. Retrieved from http://www.Arduino.cc/playground/Main/SimilarBoards#ardAndCompBds

Arduino. (n.d.c). Arduino Uno. Retrieved from http://Arduino.cc/en/Main/ArduinoBoardUno

Arduino. (n.d.d). Forum - index. Retrieved from http://Arduino.cc/forum/

AXCase. (2011). AXCase - Interactivity@UofA. Retrieved from http://interactives.tapor.ualberta.ca/?page_id=8

Baichtal, J. (2011). Hackerspace happenings: MAKE interviews tokyo's Akiba. *Make: Online*. Retrieved from http://blog.makezine.com/archive/2011/03/hackerspace-happenings-make-interviews-tokyos-akiba.html

Balka, K., Raasch, C., & Herstatt, C. (2009). Open-source enters the world of atoms: Astatistical analysis of open design. *First Monday*, *14*(11). Retrieved from http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/2670/2366

BatchPCB. (2005). What is this place?. Retrieved from http://batchpcb.com/index.php/AboutUs

Bell Labs. (2002). An overview of the UNIX* operating system. Retrieved from http://www.bell-labs.com/history/unix/tutorial.html

Bell Labs. (n.d.). Bell labs early contributions to computer science. Retrieved from http://www.bell-labs.com/history/unix/blcontributions.html

Buechley, L. (2006). A construction kit for electronic textiles. Proceedings of IEEE International Symposium on Wearable Computers. Presented at the IEEE International Symposium on Wearable Computers, Montreux, Switzerland. Retrieved from http://hlt.media.mit.edu/publications/buechley_ISWC_06.pdf

Buechley, L. (2010). LilyPad Arduino: Rethinking the materials and cultures of educational technology. Proceedings of the 9th International Conference of the Learning Sciences. International Society of the Learning Sciences.

Buechley, L., & Hill, B. M. (2010). LilyPad in the wild: How hardware's long tail is supporting new engineering and design communities. *Proceedings of Designing Interactive Systems* (pp. 199-207). Presented at the Designing Interactive Systems, Aarhus, Denmark.

Buys, J. (2010). Apple's relationship to open-source. In *Ostatic.* Retrieved from http://ostatic.com/blog/apples-relationship-to-open-source

Canonical. (n.d.a). About Ubuntu. Retrieved from http://www.ubuntu.com/project/about-ubuntu

Canonical. (n.d.b). Ubuntu-certified hardware. Retrieved from http://www.ubuntu.com/certification/

Cascio, J. (2009). The desktop manufacturing revolution. *Fast Company*. Retrieved from http://www.fastcompany.com/blog/jamais-cascio/open-future/material-issue

Chuck. (2002). ChucK: Strongly-timed, on-the-fly audio programming language. Retrieved from http://chuck.cs.princeton.edu/

Colin. (n.d.). Toaster oven reflow soldering. Retrieved from http://www.instructables.com/id/Toaster-Oven-Reflow-Soldering-BGA/

Computer Science and Artificial Intelligence Laboratory. (2007). MIT CSAIL open-source hardware designs. Retrieved from http://csg.csail.mit.edu/oshd/index.html

Conway, L. (1982). The MPC adventures. *Microprocessing and Microprogramming - The Euromicro Journal*, *10*(4), 209-228.

Crosby, K. (1995). Convivial cybernetic devices. Retrieved February 17, 2011, from http://www.opencollector.org/history/homebrew/engv3n1.html

Cycling 74. (n.d.). Cycling74. Retrieved from http://cycling74.com/

Davis, B. (2011). Hackerspaces: The community centers of the future. Retrieved from http://www.detroitmoxie.com/home/2011/2/21/hackerspaces-the-community-centers-of-the-future.html

Daw, D. (2010). The 3D printer revolution countdown: Print your own PC coming shortly. *PCWorld.* Retrieved from http://www.pcworld.com/article/212440/the_3d_printer_revolution_countdown_print_your_own_pc_coming_shortly.html

De Vinck, M. (2009). Roboduino. *Make: Online*. Retrieved from http://blog.makezine.com/archive/2009/01/roboduino.html

DiBona, C., Ockman, S. (Eds.). (1999). Open sources: Voices from the open source revolution. Sebastopol, CA: O'Reilly Media.

DistroWatch. (n.d.). Top ten distributions. Retrieved from http://distrowatch.com/dwres.php?resource=major

Dorkbot. (n.d.). Start a dorkbot in your city!/what is dorkbot?. Retrieved from http://dorkbot.org/startadorkbot/

Fab@Home. (n.d.). Make anything. Retrieved from http://www.fabathome.org/

Free Software Foundation. (2011). What is copyleft?. Retrieved from http://www.gnu.org/copyleft/

Free Software Foundation. (2010). The free software definition. Retrieved from http://www.gnu.org/philosophy/free-sw.html

Freedom Defined (n.d.a). Definition of free cultural works: Endorsements. Retrieved from the Freedom Defined Wiki: http://freedomdefined.org/OSHW#Endorsements

Freedom Defined (n.d.b). Definition of free cultural works: OSHW. Retrieved from the Freedom Defined Wiki: http://freedomdefined.org/OSHW

Freedom Scientific. (n.d.). Products for the blind and visually impaired. Retrieved from http://www.freedomscientific.com/products/fs/blindness-products.asp

Gates, W. (1976). An open letter to hobbyists. Retrieved from http://www.blinkenlights.com/classiccmp/gateswhine.html

Gruber, J. (2006). Why apple won't open-source its apps. *Daring Fireball*. Retrieved from http://daringfireball.net/2006/06/apple_open_source

Hackerspaces. (n.d.). Hackerspaces. Retrieved from Hackerspaces Wiki: http://hackerspaces.org/

Hars, A., & Ou, S. (2001). Working for free? Motivations of participating in open-source projects. In System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on (p. 9 pp.). doi:10.1109/HICSS.2001.927045

Hood & Strong LLP. (2010). Independent auditors' report and consolidated financial statements. Retrieved from http://www.mozilla.org/foundation/documents/mf-2009-audited-financial-statement.pdf

Homebrew Computer Club. (1975). Homebrew computer club newsletter number one. Retrieved from http://www.digibarn.com/collections/newsletters/homebrew/V1_01/index.html

Infusion Systems (n.d.). Products. Retrieved from http://infusionsystems.com/catalog/

Inkscape. (n.d.). About Inkscape. Retrieved from http://inkscape.org/

Instructables (n.d.). Instructables - Make, How To, and DIY. Retrieved from http://www.instructables.com/

JSON. (2002). The JSON License. Retrieved from http://www.json.org/license.html

Klein, E. (n.d.). MITS Altair 8800. Retrieved from http://www.vintage-computer.com/altair8800.shtml

Kurt, E. T. (2006). Arduino, the BASIC Stamp killer. *Todbot Blog*. Retrieved from http://todbot.com/blog/2006/09/25/Arduino-the-basic-stamp-killer/

Lindal, J. (1999). Why a bazaar may fizzle or fail. Retrieved from http://jafl.my.speedingbits.com/bazaar_fizzle_fail.html

Make Magazine. (n.d.). About Make. Retrieved from http://makezine.com/about/

MakerBot Industries. (n.d.). MakerBot. Retrieved from http://www.makerbot.com/

MakerGear. (n.d.). Prusa 3D Printers. Retrieved from http://www.makergear.com/products/3d-printers

McCarthy, J. (1999). VA Linux IPO soars almost 700%. *Network World Fusion.* Retrieved from http://www.networkworld.com/news/1999/1210linux.html

McNamara, P. (2007). Open hardware. *Open-Source Business Resource.* Retrieved http://www.osbr.ca/ojs/index.php/osbr/article/view/379/340

Moore, J.T.S. (Director). (2001). Revolution OS [Motion picture]. Retrieved from http://www.revolution-os.com/

Mulder, A. (1995). The I-Cube System: Moving towards sensor technology for artists. Retrieved from: http://www.xspasm.com/x/sfu/vmi/ISEA95.html

Monome. (2011). About. Retrieved from http://Monome.org/about

Mozilla Foundation. (2010). State of Mozilla - Sustainability. Retrieved from http://www.mozilla.org/foundation/annualreport/2009/sustainability.html

OpenCores. (2011). OpenCores: Mission. Retrieved from http://opencores.org/opencores,mission

Open Cellphone Project. (n.d.). TuxPhone. Retrieved from Open Cellphone Wiki: http://www.opencellphone.org/index.php?title=Main_Page

Open Graphics Project. (n.d.). OGP - Welcome to the Open Graphics Project . Retrieved from the Open Graphics Project Wiki: http://wiki.opengraphics.org/tiki-index.php

Open Hardware Summit. (2011). Open Hardware Definition 1.0 released!. Retrieved from http://www.openhardwaresummit.org/2011/02/10/open-hardware-definition-1-0-released/

Open Hardware Summit. (n.d.). OSHW Definition v1.0. Retrieved from http://www.openhardwaresummit.org/oshw-definition-v1-0/

Open-Source Initiative. (n.d.a). About the Open-Source Initiative. Retrieved from http://www.opensource.org/about

Open-Source Initiative. (n.d.b). GNU General Public License, version 3 (GPL-3.0). Retrieved from http://opensource.org/licenses/GPL-3.0

Open-Source Initiative. (n.d.c). Mozilla Public License 1.1. Retrieved from http://opensource.org/licenses/mozilla1.1.php

Open-Source Initiative. (n.d.d). The BSD License. Retrieved from http://opensource.org/licenses/bsd-license.php

Open-Source Initiative. (n.d.e). The Open-Source Definition. Retrieved from http://www.opensource.org/docs/osd

OpenCores. (n.d.). OpenCores. Retrieved from http://opencores.org/

OpenSPARC. (n.d.). What is OpenSPARC?. Retrieved from http://www.opensparc.net/about.html

Osier-Mixon, J. (2010). Open hardware: What's it all about?. *Linux.com* Retrieved from http://www.linux.com/learn/tutorials/364055-open-hardware-whats-it-all-about

Processing. (n.d.a). Overview. Retrieved from http://processing.org/about/

Processing. (n.d.b). Processing. Retrieved from http://processing.org/

Provost, J. (2011). Why the arduino won and why it's here to stay. *Make: Online*. Retrieved from http://blog.makezine.com/archive/2011/02/why-the-Arduino-won-and-why-its-here-to-stay.html

Randerson, J. (2006). Put your feet up, Santa, the Christmas machine has arrived. *The Guardian.* Retrieved from http://www.guardian.co.uk/science/2006/nov/25/frontpagenews.christmas2006

Raymond, E. (2002). The cathedral and the bazaar. Retrieved from http://catb.org/~esr/writings/homesteading/cathedral-bazaar/index.html#catbmain

RepRap. (n.d.a). Forums. Retrieved from http://forums.reprap.org/

RepRap. (n.d.b). Motherboard 1.2 - RepRapWiki. Retrieved from the RepRap Wiki: http://reprap.org/wiki/Motherboard_1_2

RepRap. (n.d.c). Welcome to RepRap.org. Retrieved from the RepRap Wiki: http://www.reprap.org/wiki/Main_Page

Rowan, D. (2011). 3D printing – An "industrial revolution in the digital age"?. *Wired Magazine*. Retrieved from http://www.wired.com/epicenter/2011/05/3d-printing-an-industrial-revolution-in-the-digital-age/

Rudenberg, H. G. (1969). Large-scale integration: Promises versus accomplishments: the dilemma of our industry. Proceedings of the November 18-20, 1969, fall joint computer conference, AFIPS '69 (Fall) (p. 359–367). New York, NY, USA: ACM.

Seaman, G. (n.d.). Free Hardware Design - Past, Present, Future. Retrieved from http://www.opencollector.org/Whyfree/freedesign.html

Seidle, N. (n.d.a). Reflow skillet. *Sparkfun Electronics*. Retrieved from http://www.sparkfun.com/tutorials/59#Skillet

Seidle, N. (n.d.b). SMD how to. *Sparkfun Electronics.* Retrieved from http://www.sparkfun.com/tutorials/36

Shankland, S. (n.d.). Red Hat shares triple in IPO. *CNET News.* Retrieved from http://news.cnet.com/2100-1001-229679.html

Sherman, B., & Bently, L. (1999). *The Making of Modern Intellectual Property Law*. New York, NY, USA: Cambridge University Press.

Skipp, E. (2010). Finished project: Exploding high-five glove. *Get Excited and Make Things!*. Retrieved from http://eliskipp.com/blog/2010/12/17/finished-project-exploding-high-five-glove/

Sribunruangrit, N., Marque, C., Lenay, C., & Gapenne, O. (2003). Improving blind people's spatial ability by bimodal-perception assistive device for accessing graphic information. GM Craddock LP McCormack RB Reilly & HTP Knops, proceedings of Assistive Technology–Shaping the Future, AAATE, 3, 476–480.

Stallman, R. (1999). On "free hardware". *Linux Today.* Retrieved from http://www.linuxtoday.com/news_story.php3?ltsn=1999-06-22-005-05-NW-LF

Stallman, R. (2002). On hacking. Retrieved from http://stallman.org/articles/on-hacking.html

Stallman, R. (2010). About the GNU Project. Retrieved from http://www.gnu.org/gnu/the-gnu-project.html

Stallman, R. (2011). Why open source misses the point of free software. Retrieved from http://www.gnu.org/philosophy/open-source-misses-the-point.html

System76. (n.d.). System76. Retrieved from http://www.system76.com

Szczys, M. (2010). TI makes a big bid for the hobby market. *Hack a Day*. Retrieved from http://hackaday.com/2010/06/22/ti-makes-a-big-bid-for-the-hobby-market/

Texas Instruments. (n.d.). Code Composer Studio (CCStudio) Integrated Development Environment (IDE) v4.x. Retrieved from http://focus.ti.com/docs/toolsw/folders/print/ccstudio.html

Turkel, W. J. (2008). A few arguments for humanistic fabrication. *Digital History Hacks (2005 - 2008)*. Retrieved from http://digitalhistoryhacks.blogspot.com/2008/11/few-arguments-for-humanistic.html

Tuscon Amateur Packet Radio. (n.d.). The TAPR Open Hardware License. Retrieved from http://www.tapr.org/ohl.html

WebM Project. (n.d.). License. Retrieved from http://www.webmproject.org/license/

Wozniak, S. (n.d.). Homebrew and how the apple came to be. *Atari Archives.* Retrieved from http://www.atariarchives.org/deli/homebrew_and_how_the_apple.php

Vance, A. (2010). 3-D printing is Spurring a manufacturing revolution. *The New York Times.* Retrieved from http://www.nytimes.com/2010/09/14/technology/14print.html

Verilog. (n.d.). Verilog resources. Retrieved from http://www.verilog.com/

Russell, M. (2005). What is darwin (and how it powers Mac OS X). Retrieved from http://oreilly.com/pub/a/mac/2005/09/27/what-is-darwin.html

Yu, W., Kuber, R., Murphy, E., Strain, P., & McAllister, G. (2006). A novel multimodal interface for improving visually impaired people's web accessibility. Virtual Reality, 9(2), 133-148.

Zagler, W. L., & Panek, P. (1999). Assisting the facilitators-interface design and telematic support for IT-based assistive technology. Technology and Disability, 10(2), 129–136.

ZCorporation. (n.d.). ZPrinter® 310 Plus. Retrieved from http://www.zcorp.com/en/Products/3D-Printers/ZPrinter-310-Plus/spage.aspx