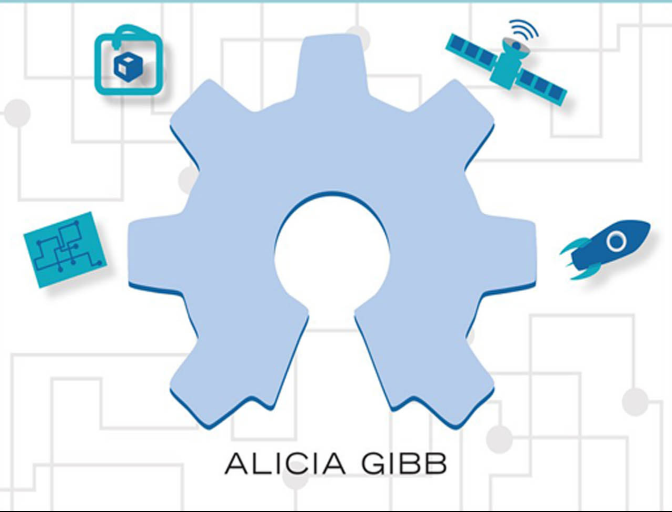
BUILDING OPEN SOURCE HARDWARE

DIY Manufacturing for Hackers and Makers



ALICIA GIBB

Building Open Source Hardware

Building Open Source Hardware

DIY Manufacturing for Hackers and Makers

Alicia Gibb
with
Steven Abadie
Ed Baafi
Matt Bolton
Kipp Bradford
Gabriella Levine
David A. Mellis
Catarina Mota
Joshua Pearce
Becky Stern
Tiffany Tseng
Addie Wagenknecht
Michael Weinberg
Amanda Wozniak
Lars Zimmerman

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Control Number: 2014952506

ISBN-13: 978-0-321-90604-5
ISBN-10: 0-321-90604-7

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, December 2014

❖

Dedicated to Aaron Swartz, a friend, a mentor,
and the greatest teacher of open source.

❖

Contents

This page intentionally left blank

Introduction

Building Open Source Hardware is an anthology written to get users and makers of open source hardware to the next step of developing for the masses and manufacturing. This book involves a hands-on approach, providing guides for developing and manufacturing open source hardware. Although many books have been published on specific pieces of open source hardware, to date there has not been a book published on the community or the steps to work all the way through designing and manufacturing a piece of open source hardware. There has been a burst of activity around making and do-it-yourself (DIY) projects, but the DIY and maker movements are growing to a new stage, wanting to produce on larger scales and move projects to products. If you have been hacking on some hardware in your basement and want to start building multiples of it and selling them on your website as open source, this book is for you.

This book covers both the theoretical side of open source hardware and the practices and methods necessary to create a piece of open source hardware. It is intended to be a holistic experience, moving from developing to manufacturing of open source hardware, while explaining the benefits, standards, and incentives found at the various stages of this process. This book includes beginner- to intermediate-level technical concepts and is coupled with an open source hardware kit that can be purchased separately to foster experimentation.

The intended audience of this book includes people from a multitude of fields, all of whom are interested in creating open source hardware and would like a guide for the theory, standards, and hands-on advice. Individuals and companies, large and small, that are already interested in the DIY and maker movement, but still need some help on how to create, document, and think about licensing, manufacturing, and selling open source hardware will also benefit from this book.

I chose not to self-publish for a number of reasons. The major one, however, was that without a publisher inviting me to write a book on this topic, the thought would have never occurred to me. My publisher is also well known in the open source software community for publishing portions of books with open source licenses, so this book is partially open source, too! The chapters written under a Creative Commons license are listed on the copyright page.

What Is Open Source Hardware?

Open source hardware—sometimes abbreviated OSH or OSHW—is hardware whose source files are publicly available for anyone to use, remanufacture, redesign, and resell. The open source hardware movement, similar to the DIY and maker movement, is not a new concept, but rather is a revitalization of historical methods that were displaced as modern manufacturing came to the fore. Modern manufacturing produces hardware cheaply and efficiently (albeit stifled with legal boundaries) and, as a result, has created a consumer culture, rather than a DIY culture. In the past 10 years, the pendulum has begun to swing back in favor of creating and fixing things rather than buying them.

Open source hardware values sharing, transparency, and accepting predecessors and successors to your work, both in the form of a company that might build something off your hardware and a project that might copy part or all of your hardware design. Transparency in hardware is becoming increasingly important as technologies become more opaque as their size dwindles, making it more difficult to discover with the naked eye how they work. As more complexities are added, the design also gets harder to discern. Open source hardware, in contrast, offers freedom of information in a physical format. Freedom of information for hardware means that the source files are accessible and easily available to rebuild the object. Source files may include schematics, diagrams, code, and assembly instructions, to name a few options.

Open source hardware does have some restrictions; in that sense, it differs from the total freedom found in the public domain. As Wendy Seltzer, a renowned legal professional, reminded the community when writing the definition of open source hardware, any limitations that we add to hardware make the hardware more closed than it already is, as hardware is actually open until it is patented. The basic open source hardware limitations are fairly simple: Anyone has the freedom to remix, remanufacture, and resell an item, provided that the hardware remains open source and attribution is given.

Maturity of the Open Source Hardware Movement

It would be irresponsible to write this book as though every aspect of open source hardware has been figured out and that there is a manual to follow to a "T." The definition of open source hardware created by the community even upholds "the spirit" of open source hardware as a consideration for labeling your hardware as open source. This open-ended sentiment shows the underdeveloped nature of the movement, and accepts the likelihood that future formats and defining characteristics will change.

For example, much of the gray area within open source hardware arises from the fact that openness does not yet extend to all layers of hardware. The process of getting raw materials out of the ground is not considered open, since most of us have no idea where the copper comes from that we use in our boards. Moreover, several software programs used to build hardware are not open. Even integral pieces of hardware, such as the chip, are often closed. I'm excited to say, however, that while this book was being written, Parallax announced its launch of an open source silicon.[1] This is a giant step forward for open source hardware. As you can see from the preceding examples, the community suspends

1. www.parallax.com/microcontrollers/propeller-1-open-source

the reality of fully open sourced hardware, and considers the existing limitations to be acceptable. Open source hardware is a malleable movement, subject to change when more openness comes along. The authors in this book represent a slice in time of the state of open source as it pertains to current hardware availabilities and challenges.

The Open Source Hardware Community

The open source hardware community includes people from many backgrounds and several different industries. In a recent OSHWA survey in this area, people identified with more than 45 different job titles, ranging from engineer to journalist. Although the open source hardware community first became popular in the electronics industry, several other industries are now making open source hardware.

Arduino was the first large-scale success in open source hardware. It was produced by a team at Ivrea Institute and was derivative of Wiring in its hardware and Processing in its integrated development environment (IDE). The community grew around Arduino and it quickly became a permanent feature within the open source hardware community. We first saw component-level modification based on Arduino and lots of break-out boards and electronic kits, but we're now seeing advancements in open source tools—for example, laser cutters, jigsaws, and 3D printers. In 3D printing, the success of Makerbot (formerly open source) was due to it being open source hardware and building off the RepRap community, which had operated in the open source hardware space for the past decade. Other industries like ecology, DIY bio (creating things like open polymerase chain reaction [PCR] devices), automotive design, and disaster relief have all joined the open source hardware community. For a longer list of industries opening up physical things and materials, see Chapter 10. Given the growing number of successful companies selling open source hardware, the movement is quickly taking shape.

During this time period there was also a growth of hackerspaces in the United States. Hackerspaces (sometimes called makerspaces) are collectives of people who experiment with art, technology, and science and who generally use nontraditional methods for innovation. Hackerspaces focus on a shared space, shared tools, and shared knowledge. Many hackerspaces teach classes and have open hack nights for the public to come learn some tricks of many different trades. During the past 10 years, the DIY movement also picked up its pace, with a resurgence of people focusing on building their own projects, reusing items, and fixing things themselves. These trends combined promote growth of the open source hardware community.

The open source hardware community is also a global community. According to 2012 and 2013 survey data from the OSHWA surveys, open source hardware projects are under way in 79 countries. This number is most likely an underestimate due to the survey being conducted only in English. Having such a widespread global movement is challenging in that the laws governing open source projects are likely to be different in each country. In addition, at a cultural level, we may not always have the best understanding of one another. An unfortunate example of this is the xenophobia that Americans display when they talk about Chinese-made items being copies and rip-offs. Some of this language has drifted into the open source hardware community, with people forgetting that open source hardware by definition can be directly copied. It may help to point out that China wrote its patent laws in 1984, so perhaps the country just has a norm of sharing and copying rather than rebuilding the wheel. The more than 200-year-old patent system embedded within the United States and some European cultures obfuscates our view, causing us to forget that there is nothing natural about a patent system. Intellectual property exists because of human-made governing structures. The open source hardware community aims to be welcoming to all types of people, no matter what their culture, gender, race, and skill level (e.g., beginner or master manufacturer). Thus it is inappropriate for the open source hardware community to be xenophobic regarding other countries' practices vis-à-vis sharing.

In further effort to be a welcoming community, every year the Open Hardware Summit establishes an anti-harassment policy[2] for the conference, which is derived from the Ada Initiative's[3] policy. The 2012 survey reported that only 4% of the open source hardware community identified as female. The anti-harassment policy, along with offers of travel grants to women, is a direct response intended to boost the number of women in the open source hardware community.

Open Source Software

Some history of open source hardware has followed in the footsteps of the history of open source software. The open source software movement is well established as a household name, enjoying popularity with developers and being a well-known concept to the masses. Open source software has been around 20 to 30 years longer than the hardware movement has. Thus, as the open source hardware movement builds in popularity, it can glean many lessons from the open source software movement. Open source hardware looks to the history of open source software for forms of governance within nonprofit and company structures, and the different options regarding implementation that open source offers.

As open source software licenses are ported to hardware, the differences in dealing with hardware versus software are becoming apparent. While the spirit behind open source software and hardware is relatively similar, some key differences emerge when working with atoms rather than bits. The main differences between open source hardware and open source software are the legal aspects regarding patent versus copyright, physical resources, creating copies, and distribution. There are other differences as well. Hardware and software are viewed by the law differently, with hardware being protected by patents and software by copyright. In the software world, resources tend to be humans and servers, but buying and selling hardware can broaden to include dependencies on specific materials, such as copper, silicon, and ABS plastics. For hardware, copying and creating a physical good often takes specialized machines, which can come with a high price point

2. http://2014.oshwa.org/policies/
3. http://adainitiative.org/

that has not yet become low enough to permit purchase by the average user. This difference is comparable to software's early days when owning a computer (or a computer with enough space and speed) was not always feasible for the average user. Distributing hardware means shipping, which adds another extra cost to hardware-based ventures (open source or not). In contrast, open source software is typically easy and cheap to copy and distribute via the Internet, typically through a repository.

What Is the Open Source Hardware Association?

In 2012, a newly formed 501(c)3 nonprofit association for open source hardware took on the challenge of advocating, educating, and uniting stewardship of the open source hardware movement. The Open Source Hardware Association (OSHWA; pronounced ä-sh-wa) aims to be the voice of the open source hardware community, ensuring that technological knowledge of open source is accessible to everyone, and encouraging the collaborative development of technology that serves education, environmental sustainability, and human welfare. OSHWA was created largely to fill the need for an umbrella organization that would encompass many communal efforts, including channeling the funds needed to support the Open Hardware Summit. The need for an organization to handle expenditures and act as a uniform resource unaffiliated with a company became apparent. The leadership of this movement involves and celebrates many individuals. The history of OSHWA is written in Chapter 1.

Since its inception, OSHWA has functioned to support the open source hardware community. We expect that these functions may change as the community develops. The next years for OSHWA will be crucial to program development that reflects these purposes. The organization runs on donations and memberships. Because this book was written with the help of so many community members in support of open source hardware, proceeds from the book's sales will go to OSHWA.

How This Book Is Organized

I have many years of involvement in the open source hardware community, chairing the Open Hardware Summit, running OSHWA, and serving as a sounding board for much of the community in those two roles. As a reflection of my experiences, this book is laid out to give useful advice to the most often asked questions and concerns. The community as a whole is moving from building things for individual purposes, to building things en masse and starting businesses, which are two very different problem sets. Building things for yourself is covered in many other formats, both in print and online. Indeed, there are a great many examples in the form of guides, tutorials, blogs, and articles. This book is meant to cover the entire process of building things on an open source basis, for which there are not yet as many resources. It is meant to be a practical resource organized in three parts.

The first part, Open Source Hardware Theory, covers the "what" and "why" of open source hardware. What does open source hardware entail, and why was it determined that way? What do the license structures mean, and why and when should you use licenses? Which types of standards do we need to be looking for in the future and why are they important? All of these questions are addressed in Part I.

The second and third sections are the "how" of open source hardware. Part II is called Hands On! Each chapter in this part walks through a different aspect of how to do something with open source hardware, be it working through a design process, making various derivatives, 3D printing, creating wearables, or figuring out source files for different types of materials. Part III, Production Bits, takes you through the production processes step by step. It covers how to manufacture products at several different scales using different methods. Production covers many different aspects, not just manufacturing, so this section also includes documenting, setting up your business, and producing open lab equipment in the research and academic field.

This book is not necessarily meant to be read from cover to cover. You may find it useful to skip to the sections or chapters that best fit your current needs. If you're researching the theory of open source hardware, you'll probably want to start at the beginning, with Part I including the theoretical chapters. Chapter 1, History of the Open Hardware Movement, is closely tied to how and why Chapter 2, OSHW Definition and Best Practices, came to be. To jump straight to the hands-on section, go to Part II. That is where you'll find ways to start building and modifying open source hardware and the acceptable ways of doing so. Part II is for people who want to dip their toes in and see the practical nature of how open sourcing hardware works. Chapter 6 provides step-by-step instructions for how to make a derivative, which you can do with existing open source hardware, and others can do with your open source hardware. Chapter 7 teaches board shape modification, and picks up where Chapter 6 leaves off. Chapters 8 and 9 delve into two open source fields, 3D printing and wearables, respectively. Chapter 10 exemplifies a number of projects that consider different types of materials and source files. If you already have your open source hardware product prototyped and you're looking for advice about going through the manufacturing process, flip to Part III. There are chapters on DIY fabrication (Chapter 11), manufacturing (Chapter 12), and troubleshooting manufacturing problems (Chapter 13). If you have already started your manufacturing process and need help ensuring your documentation is written to the standards of the open source hardware community, skip to Chapter 14. If you're most interested in benefits of starting an open source hardware business, go to Chapter 15. If you work in academia and are interested in producing open source lab equipment, flip to Chapter 16. During the course of this book, while the focus will be on open source hardware, general building and manufacturing are also covered because certain methods are not specific to open or closed source development.

Given that the open source hardware community has many contributors, it seems only right that this book should also reflect the communal voice of the movement. You will notice that different chapters have different authors. Due to the multiple authors, the voice may differ from chapter to chapter.

This authors and arrangement of the book are as follows:

Part I: Open Source Hardware Theory

1. History of the Open Hardware Movement by Catarina Mota

   The history of the open source hardware community is mirrored in this chapter from the oshwa.org website. This chapter describes when and how decisions on open source hardware were made. Catarina Mota has been instrumental in the open source hardware community: researching hackerspaces, leading the open source hardware community surveys, and having been a previous OSHWA board member and chair of the Open Hardware Summit.

2. OSHW Definition and Best Practices by Alicia Gibb

   In 2010, a definition of open source hardware was widely adopted by the open source hardware community. In 2013, the community came out with best practices, both are recorded in this chapter with some historical references.

3. Licensing Open Source Hardware by Michael Weinberg

   The appropriate times when one should use trademarks, copyrights, and patents can be confusing to the average hardware builder. This chapter was written by a legal professional to help educate the open source hardware community about the forms of intellectual property (IP) on which open source alternatives are dependent. Michael Weinberg has been very active from the start of the open source hardware community; he continues to write about open source hardware at Public Knowledge, and organizes relevant events in Washington, D.C.

4. Standardization of Open Source Hardware by Ed Baafi

   Standardization refers to making open source hardware parts more open, focusing on the interfaces between hardware and software, and standards that make open source easiest to understand. Ed Baafi has been promoting these types of standards for the past few years within the open source hardware community. He is the founder of Modkit, and an advocate for open source hardware in education.

Part II: Hands On!

Part II of the book teaches you to use open source hardware in different ways.

5. The Design Process: How to Get from Nothing to Something by Amanda Wozniak

   The design process is the first chapter you should read to dig into the hands-on portion of this book. Amanda Wozniak has spoken at multiple Open Hardware Summits on this topic and is well known in the open source hardware community for her knowledge of engineering, systems, and the design process with regard to open source.

6. Making a Derivative by Alicia Gibb

   This chapter walks through an example of how to make a derivative. An open source hardware kit, sold separately, accompanies this chapter, which you are free to use, remake, remix, and resell. I wrote this chapter so that I would have the ability to open the hardware and use it as an example of how derivatives work.

7. Modifying the Shape of an Arduino by Tiffany Tseng

   Tiffany Tseng has reformed her own Arduino derivative as part of her research at MIT. She wrote this chapter based on her experience with form from a design perspective, and her engineering know-how of walking through all the steps it takes to create an Arduino derivative based on the form factor of the board.

8. Remix a 3D Print(er) by Steven Abadie

   This chapter gives you resources for using open source hardware 3D printers, and walks through the steps to remix a 3D printable object. Steven Abadie is the chief operating officer of Aleph Objects, which produces the Lulzbot, an open source hardware line of 3D printers. The Lulzbot printers are regarded as the most innovative line of 3D printers in the open source hardware community, and were derived from RepRap.

9. Wearables by Becky Stern

   As the concept of wearables becomes more widely known, this chapter reminds us how to make wearables open source. Becky Stern, a talented technologist and seamstress, is a well-known individual in open source hardware circles, working first at *Craft*, then writing for *Make,* and now serving as the Wearables Director at Adafruit.

10. Physical Materials by Gabriella Levine

    Gabriella Levine, who serves as president of the OSHWA, has highlighted different industries and different types of source files that are employed in each industry. Because new industries are coming into the open source hardware fold, this important chapter establishes examples of source files that may not come from traditional electronics sources.

Part III: Production Bits

11. Personal Manufacturing in the Digital Age by David Mellis

    Personal manufacturing is a concept in which David A. Mellis is considered an expert. He is part of the Arduino team, and studied personal fabrication for his PhD. His chapter looks at case studies of hardware and tools used to make things yourself, also called personal manufacturing.

12. Accelerate from Making to Manufacturing by Matt Bolton

    Matt Bolton is the Director of Production at SparkFun Electronics. His role at SparkFun is integral to what it means for a hardware hacker, DIYer, or maker to manufacture a product. This chapter explains the manufacturing process at a beginner level, for those looking to take their open source hardware to a larger scale.

13. Troubleshooting from Your Design to Your Manufacturer by Kipp Bradford

    This chapter should be read alongside the various manufacturing chapters within the book for further advice on what to do when you need to troubleshoot your hardware. This chapter should make troubleshooting easier, if you follow Kipp Bradford's advice. Kipp has worked in manufacturing in multiple fields, from toys to military-grade equipment, and has played a part at every Open Hardware Summit to date.

14. Taxonomy of Hardware Documentation by Addie Wagenknecht

    Addie Wagenknecht, founder of Lasersaur, knows all about documentation because of the unique way Lasersaur has initiated its bill of materials (BOM) as a parts list that can be purchased at any hardware store around the world, rather than focusing on collecting and shipping materials. Addie covers the integral documentation needed for hardware source files and other useful documents to help your users. Addie is also chair of the Open Hardware Summit and interfaces with the open source hardware community on a regular basis.

15. Business by Lars Zimmerman

    This chapter explores the possibilities and options that open source hardware businesses can leverage and benefit from. It was important this chapter was written by a third party rather than any single open source hardware business to show the entire landscape of open source hardware business models. Lars is the co-founder of the Open It Agency, which helps businesses learn about and implement open source hardware.

16. Building Open Source Hardware in Academia by Joshua Pearce

    Dr. Joshua Pearce is a professor at Michigan Tech University and has written the book *Open-Source Lab.* He has created open source lab equipment to take the place of closed source equipment, a topic that is highlighted in his chapter. This chapter also discusses the crucial nature of marrying open source and education together, so that students may learn without boundaries.

Special Elements

Within the chapters in this book, readers will also find a few special elements. Community members wrote anecdotes giving small snippets of their experiences with open source hardware. These can be found throughout the chapters of the book in gray boxes identified by word "Anecdote." The authors for the anecdotes further the perspectives and examples of each chapter.

This book can be accompanied with a hardware kit that was built for Chapter 6, but also used in several places as an example throughout this book. The kit can be thought of as an add-on for hands-on learning and is sold separately. You can purchase at bit.ly/blinkybuildings or at SparkFun.com.

This page intentionally left blank

Acknowledgments

This book would have never been written if Debra Williams-Cauley had not invited me to write it. I thank Debra and her team for being absolutely awesome editors to work with—and for having the idea in the first place.

Of course, a huge "thank you" to the authors who undertook writing each chapter. Your thoughts, time, and willingness to give the royalties to OSHWA is so appreciated! Thank you to the community members who wrote anecdotes to add perspectives to the chapters, and to Davy Uittenbogerd and Geoff Steele for sharing open source files for me to base the Blinky Buildings project from! Thank you to Stephen Murphey and Adrianna Danaila for letting me modify their icons on the cover of this book.

Thank you to my readers for taking their time to make this book better: Max Whitney, who did an outstanding job of catching both technical errors and improving my critical thinking skills, along with Nathan Seidle, Bryan Smith, and Jeffrey Osier-Mixon.

Thank you to my loving husband, Nathan. Nathan has made this book possible in so many ways, from fixing me three square meals a day to being one of my readers and discussing my thoughts anytime day or night. It definitely helps to have a partner in the same industry and educated about the subject matter when writing a book!

And finally, thanks to Mom and Daggles for continuously asking me, "What is it you do again?" Their questions honed my ability to define and explain the reasons why open source hardware is important.

This page intentionally left blank

About the Authors

Alicia Gibb is an advocate for open hardware, a researcher, and a hardware hacker. Alicia has worked within the open source hardware community since 2008. She is the founder and Executive Director of the Open Source Hardware Association (OSHWA), an organization to educate and promote building and using open source hardware. She directs the BTU Lab at Colorado University at Boulder, where she teaches in the areas of physical computing and information technologies. Previous to serving OSHWA, Alicia was a researcher and prototyper at Bug Labs, where she ran the academic research program and the Test Kitchen, an open R&D lab. She was awarded a National Science Foundation SBIR grant for her sensor-based data collection module while at Bug Labs. She is a member of NYC Resistor, where she has curated two international art shows; cofounded and co-chaired two Open Hardware Summits; and sits on the board of the Ada Initiative. Her electronics work has appeared in *Wired* magazine, *IEEE Spectrum, Hackaday,* and the *New York Times.* When Alicia is not researching at the crossroads of open technology and innovation, she is prototyping work that twitches, blinks, and might even be tasty to eat.

Steven Abadie is an MFA graduate from the University of Georgia. His research as a sculptor included work with open hardware 3D printing. This experience in 3D printing evolved into his current position as the COO of Aleph Objects, a company with a fierce commitment to open hardware and free software.

Ed Baafi is an educator and entrepreneur focused on making tools for technology innovation accessible to all. As an educator, Ed helped shape Boston's Learn 2 Teach, Teach 2 Learn youth technology program, along with the Boston Fab Lab. As an entrepreneur, Ed founded Modkit (http://modkit.com) to develop accessible programming tools for controlling the physical world. He is also a co-founder of Made by Cafe (http://madebycafe.com), which aims to bring an innovative cafe-like environment to designers and makers.

Matt Bolton is the Director of Production at SparkFun Electronics, where he leads the manufacturing team in building more than 500 unique DIY electronics circuit boards and kits for the makers, inventors, and hackers of the world. He is on a mission to make manufacturing sexy once again in the United States, with a vision for a new generation of manufacturing that contradicts the "dirty, dumb, and dangerous" stereotype that manufacturing held only a few decades ago. Matt is also the emcee every year at SparkFun's annual Autonomous Vehicle Competition. In his spare time, he loves to take part in just about every "quintessentially Colorado" activity possible: hiking, bouldering, snowboarding, cycling, sipping on a mug of hot black coffee, strumming his guitar, and posting pictures of his amazing dog, Olive, to Instagram.

Kipp Bradford is an entrepreneur, technology consultant, and educator with a passion for making things. He is the founder or co-founder of start-ups in the fields of transportation, consumer products, HVAC, and medical devices, and holds numerous patents for his inventions. Some of his more interesting projects have been turned into kippkitts. Kipp is the author of Distributed Network Data (hardware hacking for data scientists, with Alasdair Allan) and is one of the co-founders of the Data Sensing Lab. He is an advisor to Highway1, the leading hardware start-up accelerator, and founded the Innovation Institute, a National Science Foundation–funded project that teaches innovation to underserved youth in New York City. Kipp also co-founded Revolution by Design, a nonprofit education and research organization dedicated to empowerment through technology. He founded and co-organizes Rhode Island's Mini Maker Faire and the Washington, D.C., Mini Maker Faire. He is one of the USA Science and Engineering Festival's "Nifty Fifty." Kipp was the Demo Chair of the 2013 Open Hardware Summit, was on the program committee and a keynote speaker for the O'Reilly Solid conference, and has been recognized as a leading innovator at Frost & Sullivan's GIL 2013. As the former Senior Design Engineer and Lecturer at the Brown University School of Engineering, Kipp taught several engineering design and entrepreneurship courses. He serves on the boards of the Maker Education Initiative, the Rhode Island Museum of Science and Art, and the Providence Athenaeum. He is also on the technical advisory board of *Make* magazine, is a Fellow at the College of Design, Engineering and Commerce at Philadelphia University, and is an Adjunct Critic at the Rhode Island School of Design.

Gabriella Levine is an artist with a background in hardware design and biology. She holds a master's degree in design and technology from ITP Tisch School of the Arts, New York University. She has created robotics start-ups including Protei, open hardware shape-shifting sailing robots to sense and protect the oceans, which are being used for sensing radioactivity, mapping plastic trash in the Pacific Ocean, and collecting oil near sites of oil spills; and Sneel, bio-inspired swimming robotic snakes that can sense environmental data in the water. Gabriella serves as President of the Open Source Hardware Association (OSHWA.org), promoting open source accessible technologies worldwide. Gabriella has exhibited work internationally, including at Ars Electronica, Eyebeam (NYC), the Science Gallery (Dublin), Interactive Art Fair Miami Art Basel, and Fountain Art Fair New York Armory Show. She received the 2012 Prix Ars Electronica Hybrid Arts Award, Gulfstream Navigator Ocean Exchange Grant, and Awesome Foundation Grant, and was a fellow of Unreasonable at Sea, a radical experiment circumnavigating the world by boat. She teaches at ITP and CIID, and her work has been written up in *Wired,* the *New York Times, Creator's Project, Vice, Scientific American,* and *Hyperallergic.* Gabriella is currently an engineer on the Rapid Evaluation team at Google and a part-time artist-in-residence at Autodesk's Instructables.

David A. Mellis is a PhD student at the MIT Media Lab. He has a master's degree in interaction design from the Interaction Design Institute Ivrea (Italy) and taught at the Copenhagen Institute of Interaction Design (Denmark). David is one of the creators of Arduino, an open source hardware and software platform for electronic prototyping, and a member of the board of directors for the Open Source Hardware Association.

Catarina Mota is co-founder of Open Materials (do-it-yourself smart materials), Everywhere Tech (open source technology transfer), and AltLab (Lisbon's hackerspace). She has taught numerous hands-on workshops on high-tech materials and simple circuitry with the goal of encouraging people with little to no science background to take a proactive interest in science, technology, and knowledge sharing. Catarina is wrapping up her PhD dissertation on the social impact of open and collaborative practices for the development of physical goods and technologies. She is currently a visiting scholar at ITP-NYU, Research Chair at the Open Source Hardware Association, TED Fellow, and member of NYC Resistor. Previously, she co-chaired the Open Hardware Summit 2012, served on the board of directors of the Open Source Hardware Association, taught as an adjunct faculty member at ITP-NYU, and was a fellow of the National Science and Technology Foundation of Portugal.

Joshua M. Pearce received his PhD in materials engineering from the Pennsylvania State University. He then developed the first sustainability program in the Pennsylvania State System of Higher Education as an assistant professor of physics at Clarion University and helped develop the applied sustainability graduate engineering program while at Queen's University, Canada. He currently is an Associate Professor cross-appointed in the Department of Materials Science & Engineering and in the Department of Electrical & Computer Engineering at Michigan Technological University, where he runs the Open Sustainability Technology Research Group. His research concentrates on the use of open source appropriate technology to find collaborative solutions to problems in sustainability and poverty reduction. His research spans areas of electronic device physics and materials engineering of solar photovoltaic cells, and RepRap 3D printing, but also includes applied sustainability and energy policy. He is the author of *The Open-Source Lab: How to Build Your Own Hardware and Reduce Research Costs.*

Becky Stern is the Director of Wearable Electronics at Adafruit. Each week she publishes a new do-it-yourself craft+tech project tutorial and video and also hosts the YouTube Live show called "Wearable Electronics with Becky Stern." Becky has been combining textiles with electronics since 2005, and has helped develop the Adafruit FLORA wearable Arduino-compatible product line. She has been shooting video since age 5 and sewing since age 8. Becky studied at Parsons The New School for Design and Arizona State University; she now teaches at the School of Visual Arts' Products of Design graduate program. She is a member of the Brooklyn art combine Madagascar Institute and the Internet-based group Free Art & Technology (FAT).

Tiffany Tseng is a mechanical engineer and designer who is researching new ways to help makers document and share what they design. She is currently a PhD student at the MIT Media Lab in the Lifelong Kindergarten Group and is developing Build in Progress, a platform for people to share the story of their design process. Prior to joining the Media Lab, Tiffany received an MS in mechanical engineering from Stanford University and a BS in mechanical engineering from MIT. Along the way, she has worked at a variety of design companies, including IDEO and Fisher-Price, with a special interest in designing products for children. She is also a college radio DJ and a reviewer of tasty snacks.

Addie Wagenknecht is an American artist based in Austria, whose work explores the tension between expression and technology. She seeks to blend conceptual work with traditional forms of hacking and sculpture. Wagenknecht's work employs a peculiar blend of hacking and visual aesthetics drenched with conceptualism. Her past exhibitions include MuseumsQuartier Wien, Vienna, Austria; La Gaîté Lyrique, Paris, France; the Istanbul Modern; and MU, Eindhoven, Netherlands; and Phillips Auction House, New York City. Addie is a member of the Free Art & Technology (FAT) Lab and chairs the Open Hardware Summit. She also co-produced the open source laser cutter Lasersaur. Her work has been featured in numerous academic papers, books, and magazines, such as *Time, Wall Street Journal, Vanity Fair,* the *Economist,* and the *New York Times.* She holds a master's degree from the Interactive Telecommunications Program at New York University, and has previously held fellowships at Eyebeam Art + Technology Center in New York City, Culture Lab UK, Institute HyperWerk for Postindustrial Design Basel (CH), the Frank-Ratchye Studio for Creative Inquiry at Carnegie Mellon University, and, most recently, the Warhol Foundation. She is represented by bitforms gallery in New York City.

Michael Weinberg is a Vice President at Public Knowledge, a nonprofit digital advocacy group in Washington, D.C. As part of its advocacy mission, Public Knowledge has pushed to introduce the concept of open source hardware to policymakers and members of Congress. Michael oversees PK Thinks, Public Knowledge's in-house think-tank, and is involved in a wide range of issues focusing primarily on copyright issues before the Federal Communications Commission (FCC) and emerging technologies such as 3D printing and open source hardware.

Amanda "w0z" Wozniak is a professional engineer with a passion for open source hardware. She holds SB and MEng degrees from MIT in electrical engineering and computer science with a minor in biomedical engineering. She has worked as an Applications Engineer for Analog Devices, as a Staff Engineer at the Wyss Institute for Biologically Inspired Engineering, and most recently as a Principal Engineer at NxStage Medical. Her freelance projects have included the Ninja Networks electronic badges for DEFCON 17 and 18, along with presenting industry best practices to the maker community at three Open Hardware Summits. She has a strong interest in learning, understanding the failure modes of complex systems, building useful things, and enabling others to do the same. She lives in Boston.

Lars Zimmermann (Berlin, 1980) is a Berlin-based artist interested in economy. In 2009, he started a project on open source for regenerative design and production called the OWi project (http://owiowi.org). Zero waste and resource life management with open source dynamics. This was his way into the field and the discourse on open source. Now he works as an open source economist trying to develop and push the field in many different ways. Everything is still tied to his initial interest in making openness work for a zero waste economy. In 2014, Lars co-founded the Open It Agency (http://openitagency.eu), an agency that helps companies, projects, and communities discover, develop, and use open source strategies. He is researching better tools for hardware documentation and communication and blogs a lot these days on open source hardware and freedom, business models, material cycles, education, and more. Visit his blog and website (http://bloglz.de).

This page intentionally left blank

6 Making a Derivative

Alicia Gibb

"Its province is to assist us in making available what we are already acquainted with."

—Ada Lovelace, on the Analytical Engine

This chapter gives an example of the source files and a physical object that you can copy, modify, make, and sell as a derivative under the Open Source Hardware Definition. This chapter first discusses derivatives and attribution, and then walks through a simple open source hardware kit named Blinky Buildings that readers are encouraged to alter or modify. Appropriate methods for creating a derivative are discussed. (The Blinky Buildings hardware kit can be purchased at www.bit.ly/blinkybuildings or at www.Sparkfun.com.) Readers can follow along with the instructions, thereby making their own derivative kit. You may have also noticed that this kit is referenced in other chapters throughout this book. The skills used in creating a derivative board consist of modifying the source files and understanding how to appropriately label derivative files and give credit. The Blinky Buildings kit is labeled with the open source hardware logo, meaning it is okay to copy and create derivatives from it. If you attempt to copy and create derivatives of hardware that is not open source, you may receive a cease and desist letter from the originating company. To be safe, look for the open source hardware logo, and stick to creating derivatives from what you know to be open.

Derivatives and Open Source Hardware

One of the reasons people open source their hardware is to allow derivatives to be built from that hardware. People create derivative hardware for many different reasons, ranging from personalized features to economic advantage. The Open Source Hardware Definition makes the following statement about derived works:

4. Derived Works. *The license shall allow modifications and derived works, and shall allow them to be distributed under the same terms as the license of the original work. The license* *shall allow for the manufacture, sale, distribution, and use of products created from the design files, the design files themselves, and derivatives thereof.*

Clearly stated in the definition is the approval to create hardware from the original design files, to make copies and distribute the design files themselves, or to create a derivative from the original design. Because open source hardware grants the right to make copies, the terms "clone" and "counterfeit" get thrown around a lot when talking about derivative works. Here are the definitions of these terms when referencing open source hardware derivatives:

Derivative: A derivative is open source hardware that has been altered or modified but is based on an original design by another person or company.

Clone or **Copy:** A clone or copy is an open source hardware product that has been directly copied and conforms with the Open Source Hardware Definition because it does not infringe on the trademarks of other companies.

Counterfeit: With a counterfeit piece of open source hardware, the trademark has been copied onto a clone or derivative piece of hardware and does not abide by the Open Source Hardware Definition because the trademark is not owned by the person or company creating the derivative. Proper attribution does not include copying trademarks. Copying trademarks is illegal.

There are many examples of open hardware derivatives. In particular, the 3D printing and Arduino communities are great places to find open hardware and their derivatives. Keep in mind that Arduino itself is a derivative of Wiring, developed by Hernando Barragan, and Processing, developed by Ben Fry and Casey Reas. Some derivatives have small changes from the original; others have large changes. Changes for derivatives generally fall within four categories: (1) The function of the device is altered; (2) the form of the device is modified; (3) the change is economic, with the creator selling the same product at a different—usually lower—price point; or (4) the change enables a better design for manufacture (DFM), making it easier to manufacture or supply parts. Economic and DFM changes often go hand in hand and can be difficult to separate. All of these changes are permitted within the Open Source Hardware Definition, including a combination of the four.

An example of a board that changed drastically in both form and function is the LilyPad, which was created by Leah Buechley. The LilyPad was mashed up with the Arduino board, altered in both form and function so that it could be sewn into textiles. This particular derivative was quite extreme in the amount of changes made to the original Arduino hardware. The reason the alterations were so drastic was that Leah invented a sewable microcontroller prior to the development of the Arduino product. (For more on the history of the LilyPad, see the anecdote in Chapter 9.) When Leah's design was put together with the Arduino board, one could argue that the Arduino's shape, the form factor of pinouts, the thickness of the PCB, the typical construction materials used, and the main purpose of the board were all altered. This particular Arduino derivative's function was to be embedded in wearables—a vastly different use than the Arduino team had previously imagined for their microcontroller. The circular, thin (not to mention purple) LilyPad is to be sewn into wearables with a needle and thread rather than solder and wire.

Of course, not all derivatives are this different. In fact, some are even more or less copies of the original.

Let's take the Arduino example one step further by considering a derivative of the derivative. Adafruit's Flora is a derivative of the LilyPad (which is derivative of the Arduino board). The Flora derivative has the same form factor as the LilyPad—it is circular in shape and flat, and has copper petals around the exterior for ease of sewing—but has a different function, with a different chip on board than found in the original LilyPad. The Flora hardware introduced the ATmega32U4 chip into wearables with different functionalities than the ATmega328 on the LilyPad (such as allowing for a USB hookup rather than using an FTDI cable). Because these designs are all open source, the LilyPad developer was then able to roll the Flora's changes back into their design, and now LilyPad also offers an Atmega32U4 product. Naturally, both products can compete in the marketplace, because they are open source hardware, nobody is suing over rights; rather, everyone is focused on innovating. You can access the source files for LilyPad and Flora and compare and contrast the design files for yourself:

Original LilyPad files linked from SparkFun's product page: www.sparkfun.com/products/9266

Flora derivative files listed in Github: github.com/adafruit/Adafruit-Flora-Mainboard

New ATmega32U4 LilyPad design rolling in the Flora's ATmega32U4 improvements: https://www.sparkfun.com/products/11190

This is how derivatives of open source work! People build off improvements and ideas from others rather than reinventing the wheel each time. This process moves innovation forward at a more efficient and more productive pace.

Why the LilyPad Arduino Has "Arduino" in Its Name

The fact that the LilyPad carries the Arduino brand name is a very important point to note. The name Arduino is a trademark held by the Arduino company. Leah Buechley made an agreement with the Arduino company to license its Arduino trademark for a fee. This arrangement should *not* be confused with Leah giving the Arduino team attribution for their original board. Arduino has tried to make an important distinction in its trademark over the years. Although it is an open source project, the logo and company name are trademarked, much as any other company in the open source hardware space (and even in open source software, for that matter) can obtain a trademark for its products. We use trademarks because trademarks protect consumers and say something about the quality of the brand they are buying, rather than to protect the intellectual property of the hardware. Unless you obtain a license from Arduino, as Leah did to enable her project to be called a LilyPad Arduino, you cannot use the word "Arduino" in the name of your derivative as a way to give credit or attribution because it is a trademarked name.[1] You can

1. http://arduino.cc/en/Trademark

help the community understand correct attribution of Arduino derivatives by attributing Arduino in your README file or your project description.

Giving Correct Attribution

The Open Source Hardware Definition states the following about attribution:

> *The license may require derived documents, and copyright notices associated with devices, to provide attribution to the licensors when distributing design files, manufactured products, and/or derivatives thereof. The license may require that this information be accessible to the end-user using the device normally, but shall not specify a specific format of display. The license may require derived works to carry a different name or version number from the original design.*

When creating your derivative, you will want to give credit to the original design without infringing on the trademark of one of original creation. As Michael Weinberg reminds us in Chapter 3, "Including a 'share alike' provision in a CC license is not a polite request that anyone who builds upon the work contribute back to the commons; rather, it creates a legal requirement." This goes for attribution provisions as well. Due to the murky nature of licensing hardware, we tend to read the source files (which can be licensed cleanly with copyright or a copyright alternative) to understand the intention to list attribution or share it alike with the same license.

Attribution is like citing someone else's work in a research paper; it is not copying and pasting the logo of the original creator and applying it to your board. Attribution can also be thought of as giving the work provenance. In the art world, giving correct provenance means identifying who had a particular piece of art before you owned it. In open source hardware, the equivalent is who hacked on that particular design file or piece of hardware before you. List their names just as you would in a citation or provenance document.

Ego or Accuracy?

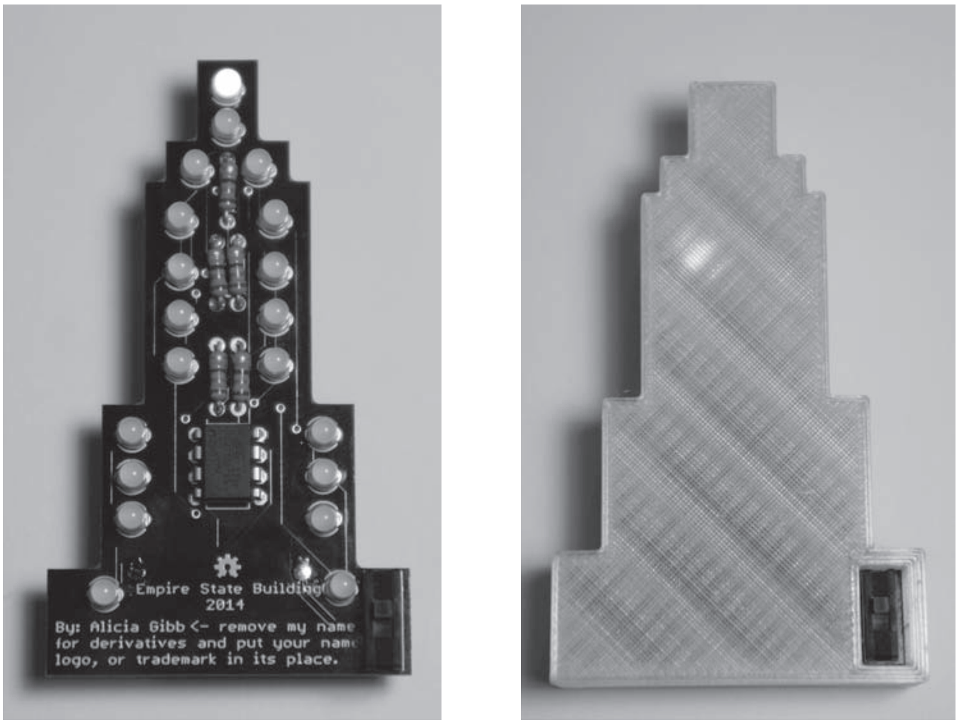
Call it ego or call it accuracy, but the open source community loves credit. Credit, or attribution, is one of the many benefits to sharing your project openly. Getting attribution for something you created is at the root of most open source licensing structures, be it in hardware or software.

Accurate attribution is important to the life of your project. Giving accurate attribution lets the community know what your project was built on. Contributors, be they original creators or makers of derivatives, may be known within the community for their quality, work style, community involvement, approach, knowledge on a particular subject, and so on. Listing creators for your derivative gives users more information and certain expectations about your derivative.

How far do you go back? Most projects don't include credit to the inventors of the transistor when using one on their board, or to the inventors of the C programming language when using Arduino. That practice is accepted within the community. We generally do not step further back than the first or second layer of original creators, although there will always be gray areas where credit is due. When in doubt, give credit. Even if your project no longer reflects any of the original design, you still may want to reference that previous versions were based on so-and-so's contraption so that people do not feel left behind or forgotten. No one will fault you for giving too much credit to other people who wrote code or built hardware before you. Perhaps the open source hardware industry will eventually grow in such a way that our README files will start to look like movie credits and go on for at least seven minutes after the movie is over.

Blinky Buildings Project

The Blinky Buildings project is a simple kit that you can use as an example of how to create an open source hardware derivative. My intention in creating this kit was to ensure that the community has something to experiment with and gives you the rights to create your own derivative. The goal of this kit is to inspire different derivatives of buildings, which together create a whole world of Blinky Building kits. My Blinky Building kit is shaped like the Empire State Building (Figure 6.1a); in its enclosure (Figure 6.1b), yours


Empire State Building
2014
By: Alicia Gibb <- remove my name for derivatives and put your name logo, or trademark in its place.

(a) (b)

Figure 6.1 Blinky Buildings: Empire State Building (b) with enclosure. (Source: Image CC-BY-SA Alicia Gibb)

can be shaped like a different building, city, or landscape structure. Your derivative Blinky Building may include any of the four alterations discussed earlier: modify the shape of the original, modify the function of the original, modify the economics, modify the DFM, or make your own copy.

Source Files

This section walks through which pieces of other people's open source material I used to create my kit; it also explores my source files that are shared with you. The source files include a circuit board created with the free version of Eagle and a 3D printing file for the enclosure. You can find all these files at www.bit.ly/blinkybuildings or in Appendix F. You will need PCB layout software, such as the following options, to be able to replicate or build off the derivative file:

- Fritzing[2]
- Eagle[3]
- KiCad[4]

You will also need a 3D printing software if you choose to print out or modify the enclosure, such as:

- Blender (reviewed in Chapter 8)
- OpenSCAD
- SketchUp

When making an open source hardware project, the most important thing to consider is whether people can rebuild the project from your source files. If so, you have a successful open source hardware project! If not, you need to release more source code or include more documentation.

As described in Chapter 5, I started my project by laying out the design process. My design purpose was to elegantly blink 20 LEDs in the shape of the Empire State Building. Given the scope and the specifications and requirements, I decided I would need a small, low-cost chip and would have to charlieplex the LEDs to drive 20 of them.

2. Fritzing is an open source project licensed under GNU GPL v3, which can take you all the way from a schematic to making the PCB.
3. Eagle offers a freeware version of its software that can be used for boards as simple as this example but is closed source software. However, this software is quite accessible in the open source hardware community, as the majority of users are familiar with Eagle.
4. KiCad is open source software for electronic designs such as schematics and PCB layout that is licensed under GNU GPL v2.

I discovered a project close to my needs that charlieplexed 20 LEDs in a falling snowflake pattern. The file was licensed as CC-BY-SA. This designation means the schematic can be copied or used for a derivative, but the new schematic must give attribution to the original and must also share alike with the same terms. In addition, this schematic came with recommended code, also licensed as CC-BY-SA. Before I did anything else, I contacted both of the original designers—the hardware schematic designer and the code author—and asked if it would be okay to make a derivative of their work and include that derivative in my book. The open source hardware definition does not require this step, but the best practices recommend it.

I started with the schematic in Figure 6.2, which was created by Davy Uittenbogerd (daaf84). The file can be opened with Fritzing: fritzing.org/projects/charlieplex-snowfallshooting-star-20-leds.

A link to the code to run this circuit is included on the Fritzing page. The code was written by Geoff Steele (strykeroz). This code can be found in this GitHub repository: github.com/strykeroz/ATTiny85-20-LED-snowflakes.

When I copied and altered the code, I added a statement at the top of the code (known as the comment block) explaining where the original code was downloaded from and who the original author was: Geoff Steele. This gives Geoff attribution. I added a brief statement about which parts of Geoff's code I altered. I included comments throughout the code when I changed something as well. Geoff included which pin numbers correlate with which color of wire on the Fritzing schematic in the code. It is good practice to include basic instructions for the hardware pinouts in the comment block.

Here are the altered chunks of code, including the attribution in the comment block. To view the full code, refer to www.bit.ly/blinkybuildings.

```
----> /* downloaded from http://code.google.com/p/avr-hardware-random-number-
generation/ Original code by Geoff Steele. Alicia Gibb altered the code by
commenting out the fade functions so the building blinks LEDs on and off rather
than fade LEDs on and off.

The original code is still all there if others wanted to keep playing with it;
just take out the duty cycle comments.

The delays have also been changed, but can easily be reinstated by looking at the
original code:

https://github.com/strykeroz/ATTiny85-20-LED-snowflakes/blob/master/ATTiny85_
Charlieplex20Snow.ino

*/
```

I explained each of my code alterations by commenting that I altered the code from the original. I used a `charlieOFF` command rather than the original `charlieON` command in line 121.

```
------>

   if (current > 19) charlieOFF(19); //This is altered from the original code, to
turn LEDs off once the blink is over
```

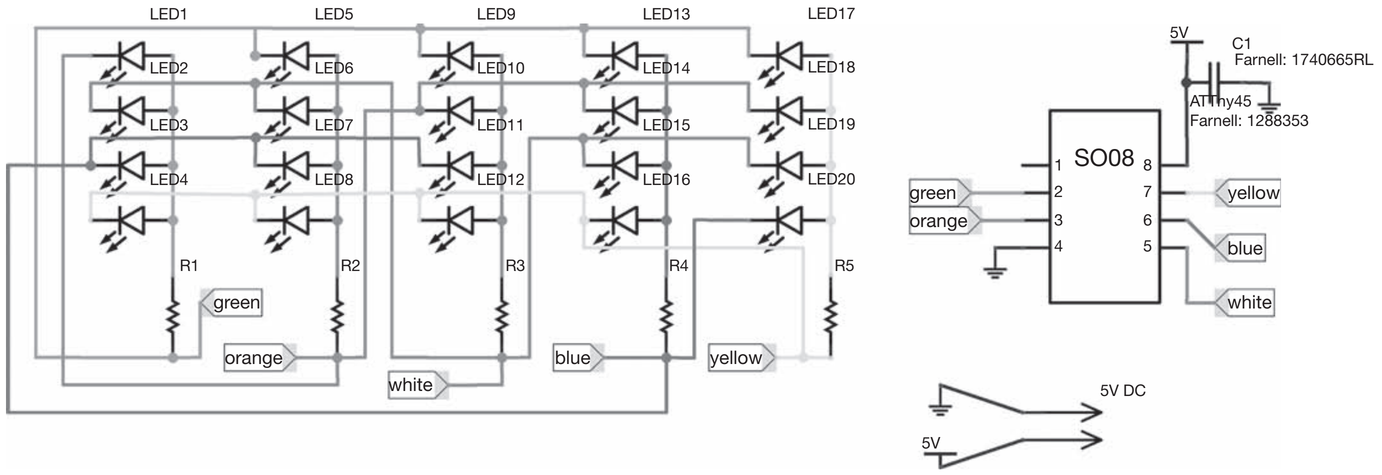


Figure 6.2 Schematic by Davy Uittenbogerd drawn in Fritzing.
(Source: Image CC-BY-SA Davy Uittenbogerd)

I explained in another portion of the code that I commented out the time delays for fading an LED so it blinks rather than fades:

```
  current++;
  if(current==23) { //// start over
     //Alicia commented out the below code to make the LEDs blink on and off
rather than fade out.
    //// now fade out the snowflake in that final position #19
   for(int dutyCycle = 3; dutyCycle <= 15; dutyCycle += 3) {
      //loopCount = 0;
      //timeNow = millis();
      //while(millis() - timeNow < (displayTime+current*2)) { //// fade out as
slow as animation has achieved by now
      // loopCount++;
       if(!(loopCount % dutyCycle)) charlieON(19);
      else charlieOFF(19);
     // }
    }
```

Once I had the code working on a bread-boarded prototype, I drew the schematic in Eagle following the Fritzing diagram. My schematic in Figure 6.3 is licensed as Creative Commons-By-Share Alike (CC-BY-SA), because the original schematic was licensed as CC-BY-SA. Due to the share-alike license, I must share it the same way. Any derivatives of this kit must also be shared alike as well, with the same Creative Commons license (CC-BY-SA) attached to the source files.

From the schematic, I created a board layout in Eagle (Figure 6.4) to be shaped like the Empire State Building. For instructions on how to give PCB boards an interesting shape, read Chapter 7. This board file is covered by a CC-BY-SA license: Because the schematic was posted under a share-alike license, and the board file is generated from the schematic, I must share it the same way. Note, however, that because Davy Uittenbogerd did not create this particular Eagle schematic or board file, it is licensed as CC-BY-SA Alicia Gibb and I will give him attribution in the README file and the product description. Labeling Davy as the creator at this point would cause confusion as to who produced and manufactured this product.

Bill of Materials

I am creating a kit for my Blinky Building that users will put together themselves. Since this is a kit, the bill of materials (BOM) will not go to a manufacturer, and it is not as detailed as the examples in Chapter 14. Generally, for a simple do-it-yourself (DIY) kit, the BOM serves the purpose of telling people what is in each kit. If the parts are standard, general parts that you could find at any hackerspace, there is no need to go into greater detail than the information shown in Table 6.1. In other documentation, it is advisable to include the data sheet of your chip as well.

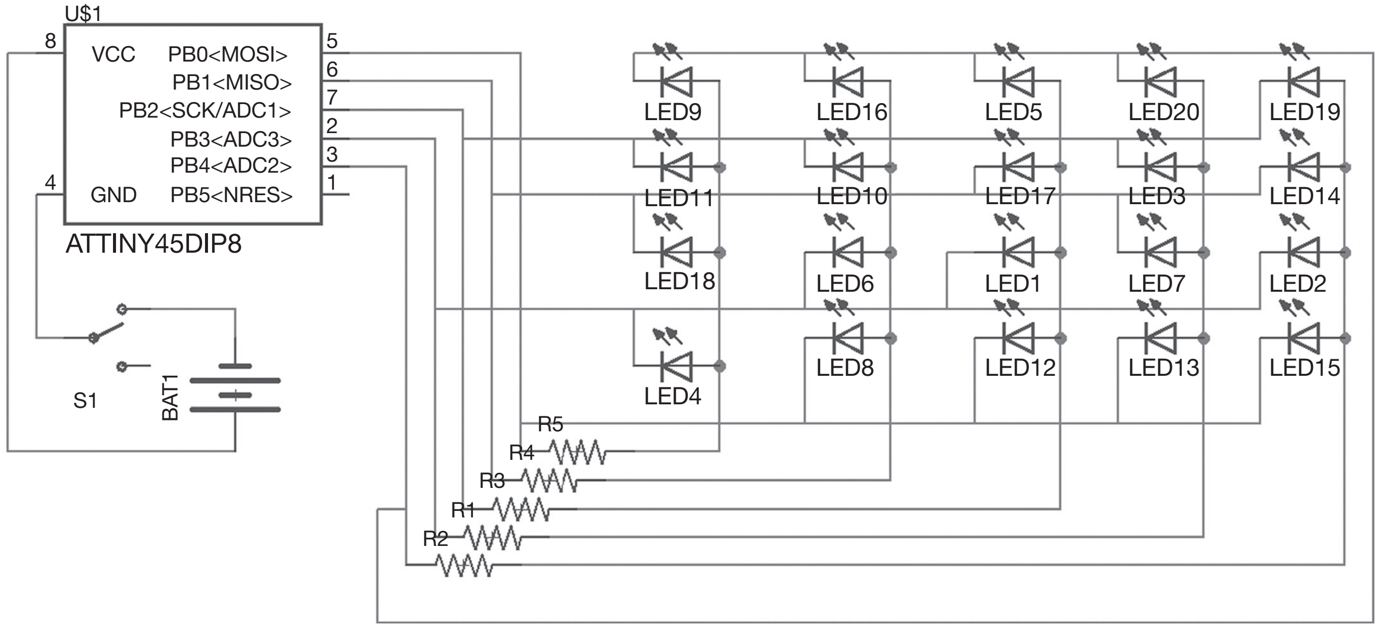


Figure 6.3 Schematic drawn in Eagle.
(Source: Image CC-BY-SA Alicia Gibb, derived from Davy Uittenbogerd's Fritzing diagram)

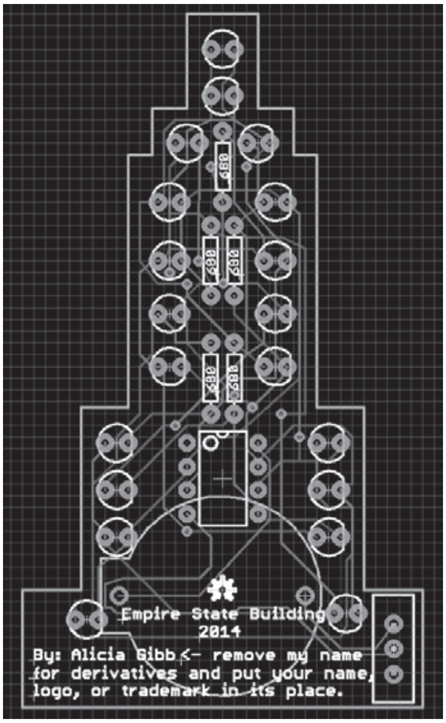

680
680
680
680
680
Empire State Building
2014
By: Alicia Gibb <- remove my name
for derivatives and put your name,
logo, or trademark in its place.

Figure 6.4 Blinky Buildings Board file in Eagle.
(Source: Image CC-BY-SA Alicia Gibb)

Table 6.1 **BOM List**

Quantity	Description	Package	Vendor Part Number	Manufacturer Part Number
1	Empire State PCB	Dimensions 1.7" × 2.95"	Golden Phoenix	
20	Round white diffused LED 8K MCD	3 mm	EBay	
5	RES 680 ohm 1/4W 5% carbon film	Axial	Digikey CF14JT680RTR-ND	CF14JT680R
1	8-bit microcontroller: MCU 8kB Flash 0.512kB EEPROM 6 I/O pins	PDIP-8	Mouser 556-ATTINY85-20PU	ATtiny85-20PU
1	Switch micro-mini slide 30V	Through hole	Digikey 679-1854-ND	MMS1208
1	Holder cell 2032 w/gold pins	Through hole	CTECHi BH32T-C-G-ND	BH32T-C-G
1	Battery lithium coin 3V 20 mm	CR2023	Digikey P189-ND	BH32T-C-G

In addition to my README file in Appendix F, this concludes the source files for this kit. With the schematic, board layout, BOM, and README file, others should be able to reproduce my Blinky Buildings Empire State kit.

Cost–Benefit Analysis of Suppliers

For complete transparency, the following lists include the suppliers from which I received quotes for each item listed on the BOM. Listing the results of the cost–benefit analysis of supplies is not required for open source hardware, but in teaching people how to make a derivative, it is important to know some economics behind what is created. As stated in Chapter 15: Business, typical mark-up on hardware is between 2.6 to 4 times your BOM

Board Manufacturer	Price/Unit	100 Pieces	Timeline
Gold Phoenix PCB	$3.11	$311	5 day turn + 8 day shipping
OHSPark	$5.00	$500	4 week turn + shipping
Advanced Circuits	$4.33	$433	4 week turn + shipping

Parts: ATtiny85	Price/Unit	100 Pieces	Timeline
Digikey	$1.95	$195.00	3–4 day shipping
Mouser	$0.75	$75.00	3–4 day shipping

Parts: Resistors	Price/Unit	500 Pieces	Timeline
Digikey	$0.008	$4.00	3–4 day shipping
Mouser	$0.33	$165.00	3–4 day shipping

Parts: LEDs	Price/Unit	2000 Pieces	Timeline
Digikey	$0.209	$418.50	3–4 day shipping
Evil Mad Scientist	$0.20	$400.00	3–4 day shipping
EBay: LED shop 2010	$0.02	$40.00	2–4 weeks

Parts: Batteries Holders	Price/Unit	100 Pieces	Timeline
Digikey	$0.60	$60.00	3–4 day shipping
CTECHi	$0.35	$35.00	5–7 day shipping

Parts: Batteries	Price/Unit	100 Pieces	Timeline
Digikey	$0.28	$28.00	3–4 day shipping
CTECHi	$0.35	$35.00	5–7 day shipping

Parts: Switch	Price/Unit	100 Pieces	Timeline
Digikey	$0.96	$96.00	3–4 day shipping
SparkFun	$1.20	$120.00	Pick up in CO or 3–4 day shipping

costs. I went with the cheapest possible BOM and marked up the Blinky Buildings kit 2.6 times the BOM.

Note

Pricing per unit were calculated based on the number of pieces being ordered. For example, a single switch at Digikey was $1.20, but buying 100 units brought the price down to $0.96.

Anecdote: Enclosure Case for Blinky Buildings

Jason Brownstein

An enclosure was made for the supplied Blinky Buildings board (Figure 6.5) in an effort to further illustrate the process of making a derivative including a 3D printed piece. This anecdote describes the design process and materials used. The enclosure described here was made using 3D modeling software, slicing software was used to produce G-code, and finally the enclosure was printed on an open source 3D printer. Chapter 8 provides links to the open source 3D modeling and slicing software packages.

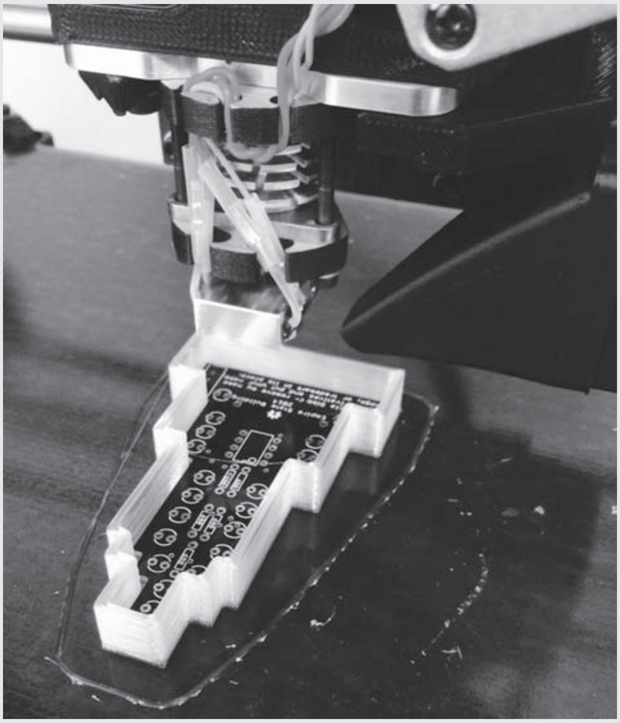


Figure 6.5 3D printing the enclosure.
(Source: Image CC-BY-SA Jason Brownstein)

(*continues*)

When stepping through the design process, many iterations were made to the enclosure to improve its functionality, rather than the overall aesthetic. Beginning with the documents and Blinky Buildings board dimensions, constraints were established on the design of the enclosure. The purpose of this enclosure is to extend the current shape profile and complement the blinkiness. The dimension tool in the board's Eagle file can be used to obtain the exact dimensions of the board, which becomes the minimum dimension of the enclosure and forms the cavity. Keeping with the building theme, the enclosure profile was chosen to match the board dimensions, with a wall thickness of 2 mm or roughly 1/16 inch. This wall thickness allows for a rigid part, but permits some minor flexibility that will be utilized later in the design process.

Several iterations were made to enable the removal of the board from the enclosure, while still accessing the switch. The first iteration of the design had two full-length tabs on the top and bottom of the enclosure to hold the circuit board. However, with a full tab, the switch on the board would not allow the board to tilt into place. A modified route was then taken in the second iteration in which snap-fit features were used on the inner side walls of the enclosure. These lofted extrusions on each wall of the enclosure allow the board to slip over it and be retained. However, the snap-fit features were still too tight to get the circuit board in and out. Thus part of the bottom tab was removed so the board could slide into place.

Figure 6.6 contains three images of the iteration process, moving from the initial model to the final model. The files to print this enclosure for the Blinky Buildings kit are provided at www.bit.ly/blinkybuildings. The print files are licensed as CC-BY-SA Jason Brownstein.

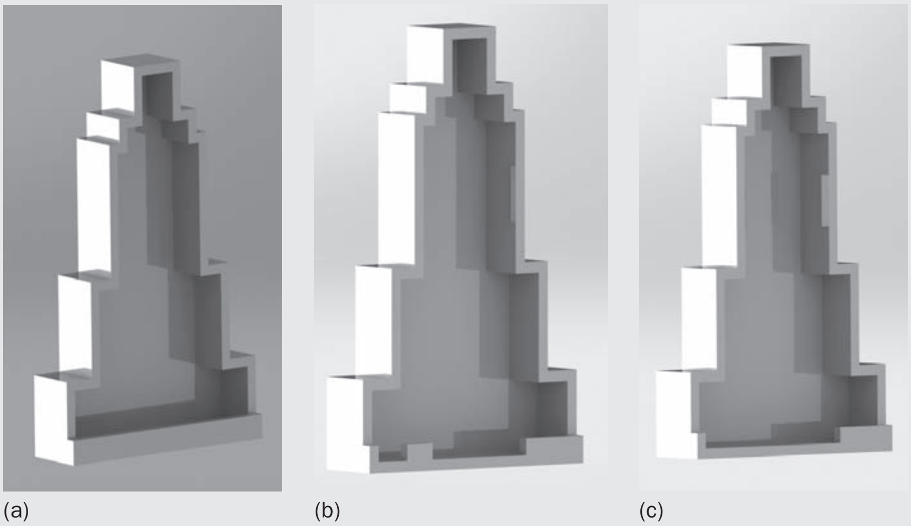


Figure 6.6 (a) First version. (b) Second version. (c) Third version. (Source: Images CC-BY-SA Jason Brownstein)

To complement the blinkiness of the board, materials selection was considered as part of the design process. Clear plastic filament is available for use on 3D printers but it differs in transparency. A trial print was completed with standard clear PLA available from Lulzbot, and the result was considered acceptable. In the second iteration, T-Glase Nylon clear filament was used. This filament was chosen because of its optical properties, which enable it to carry light in much the same manner as fiber-optic cable. The T-Glase made the PLA look slightly dull in appearance. The appearance of the T-Glase diffused the light much better and provided an almost shiny finish to the enclosure. But with any change in the design process comes consequences, which may cause you to rethink another aspect of the design or manufacture. When using PLA, the settings are relatively standard for the Lulzbot, and the printing operation was executed with a high success rate. By comparison, the T-Glase nylon required reworking the settings, including increasing the extruded temperature, slowing the print head movements down by more than half the PLA speed, increasing additional cooling with use of the onboard fan, and adjusting the extrusion layer height. Each of these settings ensured the print layers fused together well, while minimizing shrinkage and unwanted slanted enclosure walls. Print one off for yourself or make a derivative!

Making a Derivative of This Kit

All of the files to re-create this kit or make a derivative of this kit live at www.bit.ly/blinkybuildings. These files are licensed as CC-BY-SA Alicia Gibb. (Remember, if you make a derivative of this project, you cannot include a noncommercial or no-derivatives clause to the source files or hardware.) Whether you change functions in the schematic, change the board form factor, or change the economics of the project, feel free to make derivatives! Turn your board into another building in your city or a landmark that is near and dear to your heart. Make the building roll away on wheels or create a glow-in-the-dark version by 3D printing a new case. Use it as a nightlight, a flashy model train landscape, or just a means to impress your friends!

Giving Correct Attribution: An Example

Earlier in this chapter, the "Giving Correct Attribution" section discussed what giving correct attribution means. This section shows you what correct attribution looks like.

Blinky Buildings is a communal descriptive title of this product that you can use—but be aware that having a communal project name is not always the norm. However, the same is not true of my name, Alicia Gibb; my company, Lunchbox Electronics; or my logo. Only I can use my name, along with other people named Alicia Gibb. But I'm the only person who can use my company name or my logo; even other people named Alicia Gibb cannot use my company name or logo. This is why the board contains the following text: By Alicia Gibb ← remove my name for derivatives and put your name, logo, or trademark in its place. Follow these instructions and place your own name or logo here and move my name (and the names of the other original creators) to an attribution section in the README.txt for your derivative.

When working on my Blinky Buildings projects, I leaned heavily on what has already been created within the open source hardware community. I have no idea how to figure out the code and schematics for charlieplexing, but I wanted to drive 20 LEDs off a small, low-cost chip. When I discovered a project close to my needs, I looked at the licensing to ensure I could use it openly, and I looked at who was behind the project to give correct attribution. I had to do a little Internet digging, but found both of the respective creators on Twitter and contacted them to ask permission to use their work (even though it was already licensed as CC-BY-SA, I wanted to ensure it would be okay to write about in a book) and for their preferred name/handle. When you contact someone to tell them you're using their open source hardware, you'll probably make their day. One reason to open source your hardware is to allow it to grow and change in ways you never expected.

Here is an example of how to correctly give another creator attribution on your project. This is the attribution section of the README.txt for my project:

```
Attribution -

The code for this project was downloaded from http://code.google.com/p/avr-
hardware-random-number-generation/ Original code by Geoff Steele, altered by
commenting some code out to blink LEDs on and off by Alicia Gibb.

The original Fritzing design of charlieplexing 20 LEDs was downloaded from
http://fritzing.org/projects/charlieplex-snowfallshooting-star-20-leds by Davy
Uittenbogerd. Alicia Gibb drew this schematic in Eagle and altered it into the
Empire State Blinky Building form factor.

The original code and the hardware files are both under a CC-BY-SA creative
commons license.

Code: CC-BY-SA: Geoff Steele

Fritzing layout: CC-BY-SA: Davy Uittenbogerd

Blinky Building schematic, board file, and BOM: CC-BY-SA: Alicia Gibb
```

I didn't take either source's names, logos, or trademarks and pass them off as my own. Instead, I gave credit for their work by acknowledging the work they created. The standard procedure for open source hardware is to include attribution both in the README.txt and in the software comment block. (For more information on README files, refer to Chapter 14, Taxonomy of Hardware Documentation.)

Onboard Byline

The physical board carries only my name, because putting the entire README file, or two other names, on the board file would take up too much space. Physical objects have a footprint with limited space for attribution, a fact that is well understood by the hardware community. There is usually not enough room on the piece of hardware itself to write the names of everyone who worked on it, including the original creators.

Figure 6.7 shows the open source hardware logo front and center on my board, so everyone who sees it will know it follows the Open Source Hardware Definition and can be copied under those terms. My board has the attribution text reflected in Figure 6.7: "By Alicia Gibb ← remove my name for derivatives and put your name, logo, or trademark in

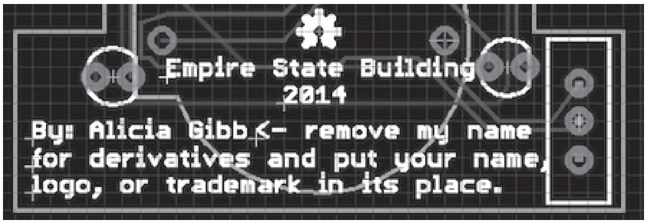

Empire State Building
2014
By: Alicia Gibb <- remove my name
for derivatives and put your name,
logo, or trademark in its place.

Figure 6.7 Byline on the board file in Eagle.
(Source: Figure CC-BY-SA Alicia Gibb)

its place." These are instructions on what to do when creating your derivative. In other words, the attribution on the board you create should read: "By: [Insert Your name here]." But unless you are Alicia Gibb, do not use that name!

Notice that on the board I used the open source hardware logo and the word "By" instead of CC-BY-SA. This usage is meant to alleviate any confusion about copyright claims. The physical hardware is not under copyright, so a CC license would not be applicable. CC licenses can be applied only to the source files and documentation, so I did not reflect the CC-BY-SA terms on my board for clarity of separating the IP that protects written documents and hardware. The term "By" does not close down my board, but patenting it would. Generally, creators do not bother to include "By" on their boards, but for the exercise of creating derivatives I wanted to spell out all the elements associated with attribution.

The open source hardware community does not yet have a standard for applying an attribution icon with the open source hardware logo. Over time, the community will most likely come to some sort of consensus as to how the open source hardware logo and other terms should be displayed, along with other conditions such as attribution.

Summary

By now, you should understand how to make a derivative of open source hardware, and be aware of the issues and benefits surrounding derivatives. You can create your own derivatives of the Blinky Buildings kit using the source files highlighted in this chapter and available at www.bit.ly/blinkybuildings. If you create a Blinky Buildings derivative, please email me at amgibb@gmail.com so that I can link to your building as well!

Remember to read the licenses of the original creators when making derivatives and follow the license terms. The most important issue that open source hardware faces with respect to derivatives is giving correct attribution without copying a trademarked logo or name. You can help open source hardware become a stronger brand by taking care to give attribution without infringing on another person's trademark.

This page intentionally left blank

This page intentionally left blank

11
Personal Manufacturing in the Digital Age

David A. Mellis

"From my point of view, the greatest developments to be expected of technics in the future . . . will not be, as we are usually led to think, in the direction of universalizing even more strenuously the wasteful American system of mass production: no, on the contrary, it will consist in using machines on a human scale, directly under human control, to fulfill with more exquisite adaptation, with a higher refinement of skill, the human needs that are to be served. . . . Much that is now in the realm of automatism and mass production will come back under directly personal control, not by abandoning the machine, but by using it to better purpose, not by quantifying but by qualifying its further use."

—Lewis Mumford, *Art and Technics* (1952)

Digital technology is enabling new alternatives to industrial production. Computer-aided design (CAD) tools encode objects as information, allowing their designs to be freely shared online—the practice of open source hardware. Digital fabrication machines turn this information into objects, allowing for precise, one-off production of physical goods. A variety of sophisticated off-the-shelf electronic components enable complex sensing, actuation, communication, and interfaces. Together, these technologies enable individuals to produce complex devices from digital designs, a process we can think of as *personal manufacturing.*

Because open source hardware involves treating physical objects as digital information, it suggests that we may be able to apply principles and practices from other kinds of online collaboration to the design of hardware. Open source software, Wikipedia, and other digital artifacts incorporate the creativity of many different individuals working without the direction of markets or firms, a process known as peer production. It works because the means of production of digital goods—computers and software—are widely distributed, the Internet makes communication and coordination efficient, and the work can be divided into pieces that individuals can choose to work on based on their own interests, needs, and abilities. The extent to which peer production can apply to hardware will shape the extent to which this approach can provide a viable alternative to mass production for the technology in our lives.

To make electronic devices amenable to these peer production approaches, we need to design with them in mind. This process yields devices that look very different than ones that are industrially produced. Such devices are optimized for translation from the digital design to the physical object. They make use of a variety of processes, from the much-hyped 3D printing to the more prosaic (but potentially more useful) techniques of laser cutting, CNC milling, and circuit board fabrication. They allow for a variety of materials and aesthetics. They can be adapted by individuals for their own needs and interests. They allow for different business models, in which objects can be made on demand or in small quantities to serve specific markets or particular individuals.

Of course, none of this eliminates the need for individual skill, whether in the design process or in the use of the fabrication machines. Good CAD tools can make the process easier, but translating an idea into concrete form requires many decisions and compromises that rely on human skill, experience, and intuition. Similarly, making effective use of a fabrication machine relies on knowledge of its configuration, operation, limitations, and quirks. Technology offers possibilities, but people turn those possibilities into reality. Similar considerations exist in open source software, where peer production doesn't eliminate the need for expertise on the part of contributors but rather provides new ways of organizing and combining those individuals' skills and efforts.

The two case studies discussed in this chapter—dealing with Arduino boards and my own consumer electronic devices—illustrate different possibilities and limitations of working with these techniques. Together, they illustrate this new personal manufacturing ecosystem, highlighting its implications for product design, for collaboration, and for business. They show some of the ways that digital technology can transform the production of objects, but also indicate some of the constraints derived from industrial systems that persist in personal manufacturing. They provide some hints of what a peer production ecosystem for electronic devices might look like, yet also point out some of the difficulties to be overcome in creating one.

The next section gives an overview of personal fabrication and the considerations involved in going from an open source hardware design file to an actual physical object. This discussion is followed by the two case studies. The lessons from the case studies are used to derive some general principles for open source hardware and personal manufacturing. Finally, I conclude with some questions and thoughts for the future.

Personal Fabrication, Processes, Parts, and Materials

Digital fabrication machines translate open source hardware designs into actual physical objects. In theory, this process depends only on the digital file and the choice of fabrication machine, allowing for iteration and refinement through successive changes to the file. In practice, though, the constraints and intricacies of various fabrication processes mean that a certain amount of skill is required to use the machine and that the results can vary each time. As a result, open source hardware depends on the selection of appropriate processes and effective use of them. This section discusses some of the considerations involved in various popular fabrication processes.

3D Printing

The purest of these digital fabrication processes are the various forms of 3D printing. These turn digital design into physical objects by gradually adding material in the desired locations, allowing for a wide range of possible geometries. The term *3D printing* encompasses a broad range of machines, from personal plastic printers costing a few hundred dollars to industrial machines that sinter metal and cost hundreds of thousands of dollars. Different machines work with different materials and offer different resolutions and tolerances. The materials may have different strengths, optical properties, appearances, finishing possibilities, and so on. Depending on the object being fabricated, some or all of these characteristics may be crucial to creating a useable result. In designing and sharing objects for 3D printing, therefore, it's important to specify not just their geometries, but also the required tolerances, materials, and other characteristics—most of which are less easily captured in digital form. In addition, many 3D-printing processes need some form of manual post-processing, such as removal of support material, finishing, or curing. These require an operator with appropriate knowledge and skill—and can create variations from one print to the next, even with the same file and machine. Finally, 3D printing technology is evolving and diversifying rapidly. For all these reasons, it's important not to think of 3D printing as a way to automatically create things from information, but rather as a material process with specific qualities and affordances.

Milling and Cutting

Other fabrication processes work by cutting or removing pieces of a larger stock material. Laser cutters cut 2D shapes out of plywood, cardboard, acrylic, and other flat materials. Vinyl cutters do the same, but with a knife that cuts through thin materials like paper or adhesive-backed vinyl. The water-jet cutter handles stronger and thicker materials like wood, metal, and glass, cutting with a stream of hard particles in a powerful jet of water. CNC (computer-numeric control) machines, like mills or routers, work in three (or more) dimensions, removing material from solid blocks of stock with a variety of cutting bits. They are often capable of very precise operations, albeit only within specific axes of movement. Compared with 3D printers, these cutting and milling tools have the advantage of being able to work with a variety of existing materials, including natural ones with complex structures that are difficult or impossible to replicate with the homogenous stock of most 3D printers. They are more limited in the geometries they can produce, however, and often require more steps in fabricating or assembling the parts.

In addition to specifying the geometry of the design itself, it's important to be explicit about the nature of the stock material and the characteristics of the cutting process. Whether two parts press-fit tightly together, slip past each other, or don't fit at all depends as much on the precise thickness of the stock (which can vary even across nominally equivalent materials) and the thickness of the cut as on the shape in the file. Some constructions may be infeasible to achieve given the tolerances of a particular machine. (Laser cutters may yield slightly different cut thicknesses on different sides of their working area; water-jet cutters can give rough, nonvertical edges, for example.) Traditional engineering drawings often capture the required tolerances for various surfaces and the material to be used. A quickly created CAD file used for a prototype and then thrown up on a webpage may not. Parts might be sanded, glued, pounded together, or otherwise tweaked in ways not reflected in the design files. Generating tool paths for a CNC machine is a complex process with a significant impact on the form and finish of the resulting object; this complexity may not be possible to capture in a way that can be easily shared with others, particularly if they are using a different machine. Finishing and assembling parts created with CNC devices requires careful craft, which might be difficult to communicate or learn. All of these factors need to be kept in mind when designing or sharing a digital file for someone else to replicate.

Other Fabrication Machines

A variety of other digital fabrication processes exist, each with its own affordances and constraints. For example, a host of machines are available for working with soft materials: CNC embroidery machines apply custom designs to fabric, knitting machines generate colors and constructions based on digital files, and Jacquard looms are possibly the oldest digital fabrication machines in existence. Industrial production uses a variety of automated machines, including robot arms and other adaptable parts of an assembly line. Furthermore, as digital fabrication becomes more established, more people are creating their own machines for custom purposes of various kinds.

Printed Circuit Boards and Electronics

The production of printed circuit boards (PCBs) can also be considered a digital fabrication process—and a relatively mature one. Digital designs are etched from copper or other materials using a photographic process, then covered with an isolating layer and text and other annotations. While the processes for creating circuit boards in this way are generally toxic and the automated systems for doing so are expensive, many services will produce PCBs on demand for individual customers with small or nonexistent minimums and standard specifications and tolerances. (As a board's specifications get more demanding, however, costs can increase, sometimes dramatically.) Circuit boards can also be manually etched or milled on a CNC machine, processes that are more directly accessible to individuals but also less robust and precise. While some circuits are sensitive to the precise characteristics of circuit board's substrate or the exact tolerances of the fabrication process, a great many can be shared with relative confidence that they will work when made on a different machine from a different provider.

In reproducing circuits, then, the main difficulties are typically getting the necessary parts and assembling them. While vast quantities of components are available to individuals—and many distributors specifically target hobbyists—advanced parts with specific functionality may not be accessible. These may be simply impossible to purchase, require an extended procurement process that makes replication infeasible, or be difficult or impossible to assemble with the processes available. As parts are optimized for size and automated assembly, they become harder for individuals to work with. Even easier-to-solder parts rely on manual skill and the knowledge to troubleshoot problems. Different electronic components may be available or preferred in different locations. Parts may go out of stock, become obsolete, or cease being made altogether. All of these factors mean that while making a PCB may be a robust and accessible process, much work must be done to ensure that individuals are able to replicate a complete electronic circuit for themselves. (It's also worth noting that while the problem may be worse for electronic components, other materials—such as plywood or 3D printer stock—are also industrial products and may not be available everywhere or all the time.)

Access to Fabrication

Access to digital fabrication processes comes in a variety of forms. Some machines, particularly 3D printers and vinyl cutters, are being targeted at individual consumers via low-cost, easy-to-use models. Local workshops, whether at schools, libraries, community centers, or commercial locations, provide access to larger, messier, and more expensive machines. They also offer opportunities for people to learn how to use the machines and can provide a community of like-minded individuals. Online services offer an alternative for those without local, hands-on access. They can provide a larger variety of processes and materials than those found in a single workshop and obviate the need to learn to operate the machines directly. On the downside, the time required for parts to be produced and shipped—and the lack of direct control over the process—can make it harder to iterate and refine designs when using an online service. Additionally, online services generally involve higher per-part prices than direct machine access, since they need to cover the cost of the machines, labor, and infrastructure required to support the service.

Case Studies

There's a lot more to open source hardware than just the fabrication and electronics technology. The following case studies draw on my personal experiences with open source hardware to discuss some of the real-world issues involved. The first case study looks at the Arduino electronics platform, a well-known open source hardware project. The second case study discusses my research at the MIT Media Lab, building open source and DIY consumer electronic products.

Case Study: Arduino Microcontroller Development Boards and Their Derivatives

Arduino is a platform for building interactive objects. It consists of microcontroller-based circuit boards and the software for programming them (both of which are open source), along with relevant documentation and community support.

Arduino builds on the work of many other projects, including the Wiring electronics prototyping platform, the Processing development environment, the GNU C Compiler (gcc), AVR libc, avrdude, and more. Since the Arduino electronics prototyping platform started in 2005, it has spawned and participated in a diverse ecosystem of software, hardware, communities, and companies. The relationships between the various actors in the Arduino ecosystem take many different forms: some specifically relate to the open source nature of the Arduino hardware, others reflect its open source software, and still others are based on more traditional business factors. As a co-founder of Arduino, I've witnessed many of these stories over the years. This case study attempts to make sense of the lessons of Arduino for open source hardware and personal manufacturing.

Because the original Arduino circuit boards are relatively simple, were created with a low-cost circuit design tool (Eagle), and use widely available parts, it's relatively straightforward for someone to make their own versions of them. This has led to a proliferation of Arduino derivatives, with a number of different modifications. These boards reveal an open source hardware ecosystem with a very different structure than that of most open source software projects. Successful open source software projects typically involve decentralized collaboration efforts, in which a number of individuals contribute to a single body of source code. The derivatives of the Arduino hardware, in contrast, tend to be produced by a small group of people, often the same ones who sell the resulting product. These derivatives often undergo few public revisions, even though some have remained available for purchase for a number of years. Moreover, relatively few changes have been contributed back to the design of the official Arduino boards. Overall, the derivatives constitute a diverse set of alternatives from different producers, in contrast to the centralized codebase that seems to prevail in most open source software projects (including, in many respects, the Arduino software itself). While some derivatives (like the LilyPad Arduino) have been incorporated into the official Arduino product line, very few (if any) modifications have been contributed to existing boards.

There are multiple obstacles to collaboration on centralized hardware designs that go beyond the need for human skill and motivation common to other domains (like open source software). One is the difficulty and expense of fabricating and assembling boards. Soldering them by hand can be done in small quantities (facilitating changes and the creation of unique variations) but is time consuming, error prone, and limited in the parts it can work with. Automated assembly is more efficient but typically requires larger quantities, limiting the frequency of changes to the circuit's design. Even worse, it can be difficult to switch between these approaches because they may require the use of different components.

Another obstacle is the relative unavailability of tools for tracking and merging changes to the design of circuit boards. Open source software has robust version control tools that allow the tracking and merging of changes by many different people. The free and low-cost circuit design tools used by most of the people designing Arduino derivatives don't provide automated methods for viewing or merging changes. Without these capabilities, the process of proposing changes to the design of an Arduino board is one that in many ways fails to take advantage of its digital nature. That is, you describe the change you'd like to see and rely on the original designer to re-create it for themselves, if desired, rather than providing a digital encoding of the change that can be automatically previewed and merged. This lack of easy methods for merging the efforts of multiple individuals reduces the viability of peer production for electronic circuits.

A final obstacle to centralized collaboration is the complications relating to the role of money and business in the production of hardware. When developers have to invest money and time in making and testing changes to the design of a circuit, they may be motivated to recoup those investments by selling their own version of a product rather than contributing their changes back to the original producer. This can create confusion around identity and branding as well, as it can become challenging to distinguish between boards of similar or identical design from different producers. As a result of these complications, Arduino has trademarked the Arduino name, using it to identify only products made by the company. This decision was initially contentious but since seems to have become an accepted practice.

The Arduino ecosystem also points out the importance of open sourcing the complements to the hardware itself. Because the Arduino software is open source (as are its underlying software tools), it gives the makers of derivatives a platform that people can use to program their boards. This factor has slowly pushed the Arduino software to become ever more general; it originally supported only a single AVR processor, then spread to most of the AVR product line, and now can support multiple processors with completely different architectures. This provides a uniform, centralized software platform for the whole ecosystem of derivatives. It also allows others to customize the software along with the hardware, adopting it to both specific uses and available resources. Online documentation (especially if liberally licensed) also makes it easier to support a new board, as that product doesn't need to be documented from scratch. This open source software and documentation, combined with accessible circuit board fabrication and electronic components, together yields a healthy ecosystem of Arduino derivatives and alternatives.

It's not clear what the relative importance of these various factors has been in promoting the vibrant Arduino ecosystem that exists today. Certainly, the Arduino software is more sophisticated (and, therefore, would be more difficult to re-create from scratch) than the basic Arduino circuits—and, in many cases, the derivative circuit designs have been re-created from scratch rather than derived from the files for the original Arduino. In theory, if the Arduino software were simply flexible and extensible, but not actually open source, it could still support a variety of derivatives of the Arduino hardware. In practice, it seems clear that many of the improvements that have been contributed to the software (including those for better support of third-party hardware) have relied in various ways on the fact that the code for the software is available. It's hard to guess which kinds of extensibility people will need, and we probably would have done a bad job if we had tried to predict those directions; instead, individuals have been able to modify the Arduino software in whatever ways they needed and the most useful of these changes have been merged back into the main codebase.

In short, it's difficult to separate electronic devices from the software that works with them. On the one hand, open sourcing just the hardware limits the modifications that can be made (without requiring reimplementation of the software). On the other hand, if the software is open source but the hardware isn't, it might not be clear that derivative designs are allowed. Thus open sourcing the hardware can help make it clear that it's acceptable to create derivatives of or add-ons to an electronic device. In addition, the design files can serve as a de facto specification and reference, facilitating the creation of compatible products. In general, the more aspects of a device's design are shared, the more likely it seems that others will reproduce or modify it.

Case Study: Open Source Consumer Electronic Products

While Arduino has demonstrated that open source hardware can create a thriving ecosystem, it's sobering to note that the vast majority of devices that people use remain proprietary. In my research at the MIT Media Lab, I've been researching the possibilities for people to build devices for use in their daily lives. I started with well-known consumer electronic products: a radio, speakers, a mouse, and, most recently, a cellphone (Figure 11.1).

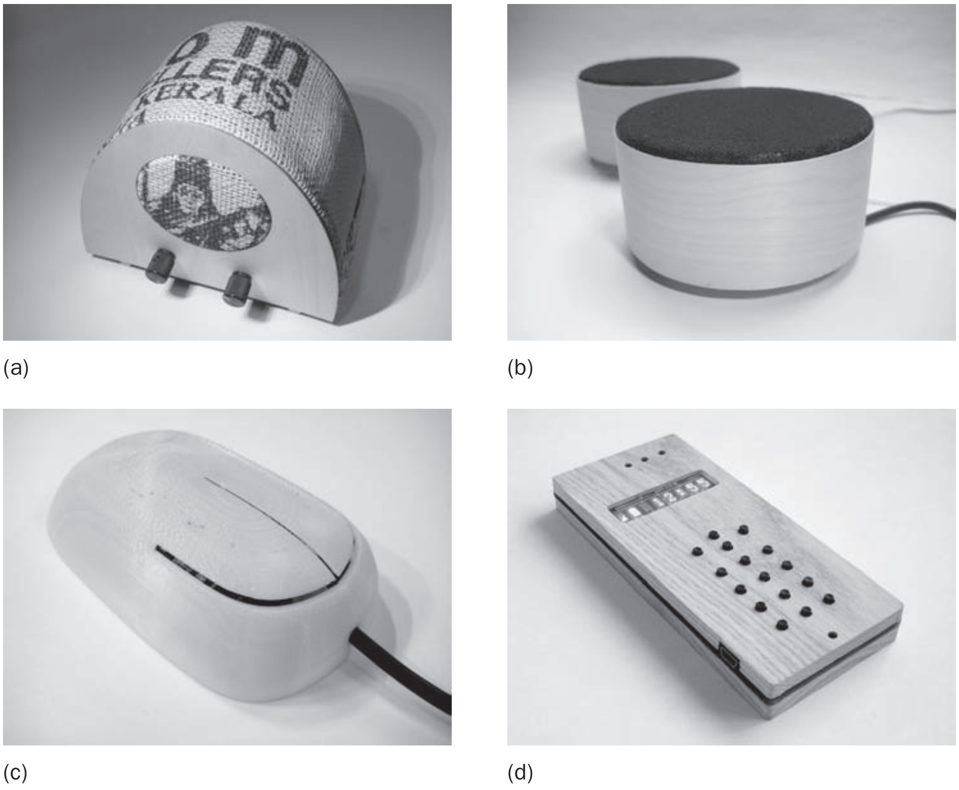


Figure 11.1 (a) A radio. (b) Speakers. (c) A mouse. (d) A cellphone.
(Source: Images CC-BY 2.0 David A. Mellis)

I design the products, prototype them, and use them in my life. In workshops, I help others to make and modify the products for themselves. In general, I try to start from technologies that are accessible to individuals and find ways to put them together into robust and attractive devices. This requires integrating the enclosure, electronic circuit, and embedded software into a complete product—and doing so in a way that lends itself to replication and modification by other individuals. It also means designing specifically for personal fabrication, which has very different opportunities and constraints than the mass production that creates most of our electronic devices.

Electronics

For the electronic aspects of a device, determining the core functionality, interface, and components is an important first step in the design process. Knowing the parts that will compose a device gives a general sense of the required form and shapes the specifics of the electronic circuit. For example, in the speakers, the decision to use three AAA batteries as the power source placed constraints on the size and shape of the speakers and on the design of amplification circuit. Component selection is also crucial for mass-produced devices, of course, but many additional constraints apply when designing devices for personal fabrication. The components have to be available to individuals and possible to assemble without expensive machines or processes. Often, there are limited possibilities available, especially for key components. The radio, mouse, and cellphone all have, at their core, an electronic part that performs much of the basic functioning of the device (receiving radio signals, interpreting the mouse movements, or communicating with the cellular network). For all three, I've had problems finding or maintaining a supply of these core components: the radio receiver and mouse chip that I used have since become unavailable and the cellphone module may become obsolete as cellular networks are upgraded.

This experience points out the essential role that industry can play in DIY: the components it makes available shape the devices that individuals can make for themselves. (There are efforts to produce open source or DIY implementations of core technologies like microcontrollers and cellular baseband modules but, in general, these don't yet seem to offer a feasible alternative to commercial components.) For these reasons, the personal fabrication or DIY process is perhaps better viewed as an individual's ability to assemble the available technologies into a desired product rather than the ability to make everything oneself, from scratch.

The limitations on component selection need to be considered when designing the circuit. For example, with the cellphone, I had to carefully balance the functional requirements against the overall size of the circuit board to yield a usable device. This imposed severe limitations on the functionality: the screen (an LED matrix) shows only eight characters at a time and there's no headphone jack, loudspeaker, removable storage, or many other common features. Even so, the phone can send and receive calls and text messages, keep time, and function as an alarm clock, which is enough functionality for me to have used it as my main phone for the past year. Cramming in more functionality may have made the device too big or fragile to actually use.

Being selective about the functions I needed (and being able to choose them for myself) allowed me to find a compromise that worked for me. In addition, because I faced similar constraints on component selection as many other hobbyists, I ended up using components for which I could download open source libraries. I was also often able to find existing circuit designs incorporating the same parts, which I could use as a reference in designing my own board.

Developing successive versions of my devices has also shown me that it's important to design your circuit boards with iteration in mind. Building extra flexibility into a design speeds up the process by making it possible to try out different forms and functions without having to fabricate a new board. For example, breaking out additional microcontroller pins, allowing for multiple types of power, and providing different mounting options can allow the board to be used in new and unexpected ways. Another approach that's sometimes useful is to provide a footprint on the board for some parts but not actually solder them on unless they're needed. These development techniques mean that you can try out new variations on a device's form and function without having to wait for a new circuit board to be fabricated, assembled, and tested.

Finally, while the design files for a device capture the components selected, they don't necessarily document the requirements or tradeoffs that led to those decisions, which may make it more difficult for others to create their own modifications of the device. In the cellphone, for example, I restricted the circuit to components shorter than 6 mm so that they'd fit within the laser-cut enclosure. This decision isn't shown in the circuit's design file but is an important constraint on the components that can be used.

Enclosures

Most of my devices have been housed in cases made on the laser cutter. This has allowed me to use natural materials, such as wood and fabric, that are rarely seen in commercial devices. It has, however, required finding clever ways to combine the flat pieces made by the laser cutter into three-dimensional objects. The radio and speakers use two parallel laser-cut plywood faces connected by struts. The faces are then wrapped with another material (either fabric or veneer). For the cellphone, I've sandwiched the circuit board with two pieces of plywood and then covered them with veneer. (In general, I'm not a fan of the finger-jointed boxes found in many laser-cut projects.) All of the designs have fairly simple contours, making it fast to laser-cut them. That constraint has allowed me to quickly iterate through designs by actually making them and seeing how well the parts fit together and how they relate to the electronics. Because the parts are designed in a simple, open source 2D drawing software (Inkscape), they are relatively easy for someone else to modify, whether by simply adding personal text to be engraved or by changing the overall form.

Another approach, which I used for my computer mouse, is to model the circuit board in 3D (using Rhino or similar software) and then use it as a reference when designing the enclosure. This strategy takes advantage of the relative flexibility of 3D printing and the resulting ability to visualize the desired object in software. In addition, 3D-printing parts can be relatively slow, particularly if you are using a high-end machine via an online service. By working in CAD before printing the enclosure, it's possible to experiment with various designs and iterate on their form and relationship to the electronics. Because this process uses more sophisticated 3D modeling software, it tends to be more difficult for a novice to modify the design of the enclosure, even in a simple way. Conversely, experts can capture more of their work in the 3D model, potentially achieving greater leverage of their skills as they share those models with others.

Assembly

I've tried to take advantage of the manual assembly required for my devices by using this step as an opportunity to engage people in their design and production. The radio and speakers include a fabric element that can be chosen by the individual making the device, giving it a unique appearance and personal significance. Other users, particularly those with prior CAD experience, have created more distinctive variations on the design of the products—creating an owl-shaped pair of speakers, in one case, or producing cellphone enclosures from a variety of materials. Assembling a device offers an opportunity and an engaging context for learning or practicing various skills, such as soldering or hand work. Many of the participants in my workshops are motivated by the desire to create a finished device but, in the process, gain experience with and appreciation for the skills involved in the process. In addition, the mere fact of putting an object together for oneself can invest it with a meaning not present for purchased products.

General Principles

The case studies suggest that there is more to making an open source hardware project successful than simply sharing its design files. These guidelines attempt to distill their lessons in ways that can be applied to other open source hardware efforts:

- *Use standard parts and materials (in conjunction with your open source design files).* For others to make use of an open source design, they need to be able to get the parts that it relies on, whether those are electronic components, screws, stock material, or something else. The more standard and widely available the parts you use are, the easier it will be for someone else to reproduce your design. That might require foregoing components that are convenient for you if they're not available to others. Note that this guideline is in some ways opposed to some quick prototyping techniques, which may favor the materials at hand regardless of their future availability.
- *Understand and design for the fabrication process used.* Different fabrication processes are good for different things—and they also have different processes and constraints. By designing for a specific fabrication process, you can take advantage of its strengths, avoid its weaknesses, and optimize for its parameters. Be specific: different kinds of 3D printers have very different possibilities, as do different stock materials that you might cut with a laser cutter or CNC machine. Working with a particular machine or process as you iterate on your design allows you to learn the capabilities of the machine and ensure that your designs are compatible with it. Of course, other people trying to reproduce your design might not have access to exactly the same machine or process, so try to find ways to avoid relying too heavily on individual quirks or features. Pay special attention to the tolerances of your chosen

fabrication process. Don't create designs that rely on a precision that's not possible to reliably achieve with the machine (e.g., if you have to laser-cut 10 parts to get 2 that actually work, you might want to rethink your design). Hand-soldering is not a particularly exact process; when designing enclosures for a circuit, remember that some components may not end up exactly where the design file specifies they should.

- *Pursue unique meanings, functions, and aesthetics.* The power and efficiency of mass production make it difficult to compete with this approach on its own terms. Instead, try to find unique values for your open source devices. Those might come from solving a problem that's of interest to only a small group of people, albeit possibly of great value to them. It might mean using unusual materials or aesthetics to differentiate your devices in ways that might not appeal to a mainstream consumer but might be appealing to someone looking for an alternative. Or the unique value might simply flow from finding ways to meaningfully involve individuals in the production of the devices. Take advantage of the fact that personal fabrication allows you to make devices in small quantities to find audiences that aren't well served by existing commercial products.
- *Find ways to make iteration faster, cheaper, and easier.* A key benefit of digital fabrication is that every part it produces can be different. To take full advantage of this ability, find ways to iterate on your design rapidly. Getting direct access to a laser cutter, for example, might mean you can try out a few designs in an afternoon instead of waiting a week or two to get a single one in the mail. Similarly, having the electronic components on hand to solder them to a newly fabricated circuit board will allow you to test that board more quickly and update its design accordingly. Identify the biggest barrier or barriers to iteration and try to find ways to remove them, whether by getting hands-on access to a machine, using software tools to refine your design before fabricating it, or being able to modify or update a part after it's been made.
- *Open source the complements to the hardware itself.* Someone who wants to re-create or modify your design will likely need more than just a raw CAD file. Provide whatever additional information seems likely to be useful—for example, parts lists, assembly instructions, firmware, and user documentation. Furthermore, by providing the original sources for these additional resources (not just compiled binaries or hard-to-edit documents like PDFs), you enable others to update them together with your hardware files when creating new variations on a design.
- *Clearly distinguish between open source design files and the products based on them.* Selling a physical product is very different from sharing a hardware design file, even if the former is based on the latter. Someone who buys a product may have higher expectations for its functionality, reliability, and safety than someone who makes a device for himself or herself based on your design. If you make and sell products based on someone else's design, be sure to distinguish between the two, making it clear that the product is from you but giving credit to the original designer.

Questions for the Future

Even if we continue to improve our practices along the lines suggested in the general principles, it's not clear what the future holds for open source hardware and personal manufacturing. The pace at which technologies of digital fabrication and embedded computation are evolving shows few signs of slowing down (notwithstanding the impossibility of the exponential growth of Moore's law continuing forever). The extent to which these improvements will extend the capability of individuals and the possibilities for open source hardware, however, is not so easy to predict. Here are three questions about the future of open source hardware and personal manufacturing—questions that I hope will encourage us to think about the future we'd like to see and to work toward making it a reality:

- *Will the technologies that can be made by individuals keep pace with those produced by large companies?* Although technology continues to improve, it doesn't necessarily do so in ways that are accessible to everyone. As a result, it's unclear to what extent open source, DIY, and peer production will be able to keep up with the devices that are produced and sold by large companies. While the potential scope of open source hardware continues to expand as technology improves, the gap between it and proprietary products may limit the extent to which it can serve as a feasible substitute for them. We should remember that the decisions we make influence the potential scope of open source hardware. If we encourage manufacturers to make their technologies available, support open tools, make use of open standards, and make our own hardware open source, we can expand that extent to which individuals are able to create, modify, and control the technologies they use in their lives.
- *Will peer production of open source hardware improve?* Although there are exceptions, open source hardware currently seems less likely than other domains (e.g., open source software) to involve collaboration between many individuals on a centralized design or repository, in which small contributions are combined together into a complex whole. Although there are many reasons for this pattern, if open source hardware is to thrive, it seems crucial to facilitate better collaboration between large numbers of distributed and diverse individuals. This will require improved tools, more efficient processes, and, perhaps most importantly, a focus on fostering communities that have a shared interest in the development of open source hardware.
- *Will the culture of open source hardware expand to include new people and applications?* Although digital fabrication and embedded computation allow for a wide variety of activities and outputs, it's easy to get caught up in the technologies themselves as opposed to their many contexts and applications. For early adopters, an interest in the technology itself can be helpful, as its uses may not be immediately clear or accessible. Even so, this emphasis on technology for technology's sake will not appeal to everyone. Thus, as we think about the future of open source hardware, we should remember to not just play with the technology, but also find ways to make it relevant and useful to new people and situations. In part, this evolution may happen

naturally as technologies mature and we come to take them for granted, but it also relies on those of us with early access to and expertise in technology to think about how to make it relevant and useful to others.

Depending on the answers to these questions, the future of open source hardware and personal manufacturing may look very different. My hope is that we will find ways to make them increasingly relevant and valuable, by expanding the technologies they can make use of, the collaborations that can produce them, and the applications and contexts to which they can be applied. If our practices can keep pace with the growth of technology, open source hardware should offer a powerful alternative to mass production for the technology in our lives.

Summary

Personal fabrication offers a potential alternative to mass production for the creation of hardware. This requires effective use of the available fabrication processes, such as 3D printing, laser cutting, and printed circuit board fabrication. As the Arduino case study demonstrates, the success of open source hardware also depends on a number of other factors, like the complements of the hardware itself and the business decision of various actors. Bringing open source hardware into our daily lives (as I try to do in my research) requires careful design of the devices themselves and the process of involving people in their production. Personal fabrication suggests new design principles. The effectiveness of these, and broader questions about the future of technology, will determine the extent to which digital technology can make peer production feasible as an alternative to mass production for hardware.

This page intentionally left blank

Index