

Open-Source Hardware vs. Open-Source Software

David A. Mellis
Sketching in Hardware
July 27, 2008

Five Questions

- What is open-source hardware?
- How does it differ from OS software?
- How is the same as OS software?
- What lessons can we learn from OS software?
- What lessons don't apply?

What is
open-source hardware?

Examples:

Arduino, OpenSPARC, RepRap,
Chumby

Seven layers:
physical, schematic, parts list,
layout, firmware, drivers.
(from PT and ladyada)

My definition:
provision of the digital artifacts
necessary to reproduce,
understand, and modify a piece
of hardware.

What makes
open-source hardware
different from
open-source software?

\$\$\$

Open-source hardware requires money. This bothers a lot of people, but time is valuable too, and open-source software requires that. On the other hand, it's easier to make money from open-source hardware.

Distribution is essential.

Once you've designed the hardware, the process is just starting. Whereas open-source software just needs to be uploaded to the web, open-source hardware needs to be manufactured, sold, distributed, etc.

Tools aren't as good.

The tools aren't as good. There aren't good ways to preserve revision history or see diffs. There aren't many standard file formats. A lot of the software is proprietary and expensive.

Testing is expensive and slow.

Testing is expensive and slow. You can't just hit "compile", you have to pay to get something made, and wait for it to arrive.

Collaboration is difficult.

Collaboration on an open-source hardware design is hard. Merging changes and testing them is expensive, making light-weight contributions more difficult. This means that you tend to have fewer people working on a particular version of the design.

Forking is the norm.

In open-source software, you get involved by contributing improvements to the canonical version of something. In open-source hardware, you often get involved by creating a new version (and competing with the one you're building on).

Encourages entrepreneurship.

Can't upgrade after release.

Extra knowledge required:
sources, manufacturers,
distributor relationships, etc.

Some of these do figure into open-source software, but they aren't essential to it in the way that they are for open-source hardware.

Licenses may not apply.

You can't copyright a circuit, just its expression, which makes it harder to use licenses like the GPL. Some efforts to create licenses for open-source hardware (e.g. the TAPR OHL license).

How is
open-source hardware
the *same* as
open-source software?

The freedoms still apply.

To use the hardware, study how it works, improve it, and share with your neighbors.

You're not dependent on the
people running the project.

You can always create your own variation, or continue the work if they abandon it.

What lessons can we
learn from open-source
software?

Start with the minimum
useful thing.

You need more than the
source.

Just like you can't just upload a piece of source code and expect people to use it, you can't just publish an Eagle file and expect people to manufacture it.

There are many ways to
contribute.

Documentation, workshops, contributing knowledge about related disciplines, etc.

People are driven by many
different motivations.

You need to figure out how to capture contributions from all of them.

Community is key.

There are many different
governance models.

Democracy, dictatorship, oligarchy, etc.

It's okay to make money.

Although you need to be careful about the effects this has on the community.

It helps to have a thick skin.

What lessons don't
apply?

The centrality of the source code.

Even though it's still "open-source", with hardware, the source code is not as important. In open-source software, membership in a project is often defined by those with commit access to a source code repository. This isn't as meaningful in open-source hardware.

What happens when
"compiling" hardware
gets cheap?

Is this just an accident of history? Will we someday be able to distribute hardware like we distribute software, thereby eliminating money and the difficulty of testing and collaboration?

Thank you.