Free Hardware and Free Hardware Designs

by Richard Stallman <https://www.stallman.org/>

To what extent do the ideas of free software extend to hardware? Is it a moral obligation to make our hardware designs free, just as it is to make our software free? Does maintaining our freedom require rejecting hardware made from nonfree designs?

Definitions

Free software is a matter of freedom, not price; broadly speaking, it means that users are free to use the software and to copy and redistribute the software, with or without changes. More precisely, the definition is formulated in terms of the four essential freedoms </philosophy/free-sw.html>. To emphasize that “free” refers to freedom, not price, we often use the French or Spanish word “libre” along with “free.”

Applying the same concept directly to hardware, *free hardware* means hardware that users are free to use and to copy and redistribute with or without changes. However, there are no copiers for hardware, aside from keys, DNA, and plastic objects' exterior shapes. Most hardware is made by fabrication from some sort of design. The design comes before the hardware.

Thus, the concept we really need is that of a *free hardware design*. That's simple: it means a design that permits users to use the design (i.e., fabricate hardware from it) and to copy and redistribute it, with or without changes. The design must provide the same four freedoms that define free software.

Then we can refer to hardware made from a free design as “free hardware,” but “free-design hardware” is a clearer term since it avoids possible misunderstanding.

People first encountering the idea of free software often think it means you can get a copy gratis. Many free programs are available for zero price, since it costs you nothing to download your own copy, but that's not what “free” means here. (In fact, some spyware programs such as Flash Player and Angry Birds </proprietary/proprietary-surveillance.html> are gratis although they are not free.) Saying “libre” along with “free” helps clarify the point.

For hardware, this confusion tends to go in the other direction; hardware costs money to produce, so commercially made hardware won't be gratis (unless it is a loss-leader or a tie-in), but that does not prevent its design from being free/libre. Things you make in your own 3D printer can be quite cheap to make, but not exactly gratis since the raw materials will typically cost something. In ethical terms, the freedom issue trumps the price issue totally, since a device that denies freedom to its users is worth less than nothing.

We can use the term “libre hardware” as a concise equivalent for “hardware made from a free (libre) design.”

The terms “open hardware” and “open source hardware” are used by some with the same concrete meaning as “free-design hardware,” but those terms downplay freedom as an issue. They were derived from the term “open source software,” which refers more or less to free software but without talking about freedom or presenting the issue as a matter of right or wrong </philosophy/open-source-misses-the-point.html>. To underline the importance of freedom, we make a point of referring to freedom whenever it is pertinent; since “open” fails to do that, let's not substitute it for “free.”

Hardware and Software

Hardware and software are fundamentally different. A program, even in compiled executable form, is a collection of data which can be interpreted as instructions for a computer. Like any other digital work, it can be copied and changed using a computer. A copy of a program has no inherent preferred physical form or embodiment.

By contrast, hardware is a physical structure and its physicality is crucial. While the hardware's design might be represented as data, in some cases even as a program, the design is not the hardware. A design for a CPU can't execute a program. You won't get very far trying to type on a design for a keyboard or display pixels on a design for a screen.

Furthermore, while you can use a computer to modify or copy the hardware design, a computer can't convert the design into the physical structure it describes. That requires fabrication equipment.

The Boundary between Hardware and Software

What is the boundary, in digital devices, between hardware and software? It follows from the definitions. Software is the operational part of a device that can be copied, and modified with a computer; hardware is the operational part that can't be. This is the right way to make the distinction because it relates to the practical consequences.

There is a gray area between hardware and software that contains firmware that *can* be upgraded or replaced, but is not meant ever to be upgraded or replaced once the product is sold. Or perhaps it is possible but unusual, or the manufacturer can release a replacement but you can't. In conceptual terms, the gray area is rather narrow. In practice, it is important because many products fall in it. Indeed, nowadays keyboards, cameras, disk drives and USB memories typically contain an embedded nonfree program that could be replaced by the manufacturer.

We can think of the difference between built-in firmware and equivalent hardware as a minor implementation detail, provided that we are sure in either case that it won't be changed. A hardware circuit can't be changed; that's its nature. If it's acceptable for a device to be implemented with internal circuitry that no one can alter, then an internal program that no one can alter is no worse. It would not be sensible to reject an equivalent internal software implementation, when operationally they are indistinguishable.

The equivalence falls apart, however, when the software implementation is not totally internal and some company can modify that code. For example, when firmware needs to be copied into the device to make the device function, or included in the system distribution that you install, that is no *internal* software implementation; rather, it is a piece of installed nonfree software. It is unjust because some manufacturer can change it but you can't.

In order for a firmware program to be morally equivalent to hardware, it must be unmodifiable. What about when the device can't possibly run without some firmware and it offers a way to modify that? We can make that firmware unmodifiable in practice by taking care never to let that replacement happen. This solution is not entirely clean, but no entirely clean solution has been proposed; this is the only way we know to preserve some meaning for the rejection of nonfree software while using that device. This is much better than just giving up.

But we can't have it both ways. To make preinstalled firmware effectively unmodifiable by not letting anyone invoke the method to change it, we must carry that out without exception even when there are changes we would wish were installed. That means rejecting all upgrades or patches to that firmware.

Some have said that preinstalled firmware programs and Field-Programmable Gate Array chips (FPGAs) “blur the boundary between hardware and software,” but I think that is a misinterpretation of the facts. Firmware that is installed during use is software; firmware that is delivered inside the device and can't be changed is software by nature, but we can treat it as if it were a circuit. As for FPGAs, the FPGA itself is hardware, but the gate pattern that is loaded into the FPGA is a kind of firmware.

Running free gate patterns on FPGAs could potentially be a useful method for making digital devices that are free at the circuit level. However, to make FPGAs usable in the free world, we need free development tools for them. The obstacle is that the format of the gate pattern file that gets loaded into the FPGA is secret. For many years there was no model of FPGA for which those files could be produced without nonfree (proprietary) tools.

As of 2015, free software tools are available for programming the Lattice iCE40 <https://web.archive.org/web/20211106213411/http://www.clifford.at/icestorm/>, a common model of FPGA, from input written in a hardware description language (HDL). It is also possible to compile C programs and run them on the Xilinx Spartan 6 LX9 FPGA with free tools </gothub/Wolfgang-Spraul/fpgatools/>, but those do not support HDL input. We recommend that you reject other FPGA models until they too are supported by free tools.

As for the HDL code itself, it can act as software (when it is run on an emulator or loaded into an FPGA) or as a hardware design (when it is realized in immutable silicon or a circuit board).

The Ethical Question for 3D Printers

Ethically, software must be free </philosophy/free-software-even-more-important.html>; a nonfree program is an injustice. Should we take the same view for hardware designs?

We certainly should, in the fields that 3D printing (or, more generally, any sort of personal fabrication) can handle. Printer patterns to make a useful, practical object (i.e., functional rather than decorative) *must* be free because they are works made for practical use. Users deserve control over these works, just as they deserve control over the software they use. Distributing a nonfree functional object design is as wrong as distributing a nonfree program.

So make sure to choose 3D printers that work with exclusively free software; the Free Software Foundation endorses such printers <https://ryf.fsf.org/>.

Must We Reject Nonfree Digital Hardware?

Is a nonfree digital [1] hardware design an injustice? Must we, for our freedom's sake, reject all digital hardware made from nonfree designs, as we must reject nonfree software?

Due to the conceptual parallel between hardware designs and software source code, many hardware hackers are quick to condemn nonfree hardware designs just like nonfree software. I disagree because the circumstances for hardware and software are different.

Present-day chip and board fabrication technology resembles the printing press: it lends itself to mass production in a factory. It is more like copying books in 1950 than like copying software today.

Freedom to copy and change software is an ethical imperative because those activities are feasible for those who use software: the equipment that enables you to use the software (a computer) is also sufficient to copy and change it. Today's mobile computers are too weak to be good for this, but anyone can find a computer that's powerful enough.

Moreover, a computer suffices to download and run a version changed by someone else who knows how, even if you are not a programmer. Indeed, nonprogrammers download software and run it every day. This is why free software makes a real difference to nonprogrammers.

How much of this applies to hardware? Not everyone who can use digital hardware knows how to change a circuit design, or a chip design, but anyone who has a PC has the equipment needed to do so. Thus far, hardware is parallel to software, but next comes the big difference.

You can't build and run a circuit design or a chip design in your computer. Constructing a big circuit is a lot of painstaking work, and that's once you have the circuit board. Fabricating a chip is not feasible for individuals today; only mass production can make them cheap enough. With today's hardware technology, users can't download and run a modified version of a widely used digital hardware design, as they could run a modified version of a widely used program. Thus, the four freedoms don't give users today collective control over a hardware design as they give users collective control over a program. That's where the reasoning showing that all software must be free fails to apply to today's hardware technology.

In 1983 there was no free operating system, but it was clear that if we had one, we could immediately use it and get software freedom. All that was missing was the code for one.

In 2014, if we had a free design for a CPU chip suitable for a PC, mass-produced chips made from that design would not give us the same freedom in the hardware domain. If we're going to buy a product mass produced in a factory, this dependence on the factory causes most of the same problems as a nonfree design. For free designs to give us hardware freedom, we need future fabrication technology.

We can envision a future in which our personal fabricators can make chips, and our robots can assemble and solder them together with transformers, switches, keys, displays, fans and so on. In that future we will all make our own computers (and fabricators and robots), and we will all be able to take advantage of modified designs made by those who know hardware. The arguments for rejecting nonfree software will then apply to nonfree hardware designs too.

That future is years away, at least. In the meantime, there is no need to reject hardware with nonfree designs on principle.

We Need Free Digital Hardware Designs

Although we need not reject digital hardware made from nonfree designs in today's circumstances, we need to develop free designs and should use them when feasible. They provide advantages today, and in the future they may be the only way to use free software.

Free hardware designs offer practical advantages. Multiple companies can fabricate one, which reduces dependence on a single vendor. Groups can arrange to fabricate them in quantity. Having circuit diagrams or HDL code makes it possible to study the design to look for errors or malicious functionalities (it is known that the NSA has procured malicious weaknesses in some computing hardware). Furthermore, free designs can serve as building blocks to design computers and other complex devices, whose specs will be published and which will have fewer parts that could be used against us.

Free hardware designs may become usable for some parts of our computers and networks, and for embedded systems, before we are able to make entire computers this way.

Free hardware designs may become essential even before we can fabricate the hardware personally, if they become the only way to avoid nonfree software. As common commercial hardware is increasingly designed to subjugate users, it becomes increasingly incompatible with free software, because of secret specifications and requirements for code to be signed by someone other than you. Cell phone modem chips and even some graphics accelerators already require firmware to be signed by the manufacturer. Any program in your computer, that someone else is allowed to change but you're not, is an instrument of unjust power over you; hardware that imposes that requirement is malicious hardware. In the case of cell phone modem chips, all the models now available are malicious.

Some day, free-design digital hardware may be the only platform that permits running a free system at all. Let us aim to have the necessary free digital designs before then, and hope that we have the means to fabricate them cheaply enough for all users.

If you design hardware, please make your designs free. If you use hardware, please join in urging and pressuring companies to make hardware designs free.

Levels of Design

Software has levels of implementation; a package might include libraries, commands and scripts, for instance. But these levels don't make a significant difference for software freedom because it is feasible to make all the levels free. Designing components of a program is the same sort of work as designing the code that combines them; likewise, building the components from source is the same sort of operation as building the combined program from source. To make the whole thing free simply requires continuing the work until we have done the whole job.

Therefore, we insist that a program be free at all levels. For a program to qualify as free, every line of the source code that composes it must be free, so that you can rebuild the program out of free source code alone.

Physical objects, by contrast, are often built out of components that are designed and build in a different kind of factory. For instance, a computer is made from chips, but designing (or fabricating) chips is very different from designing (or fabricating) the computer out of chips.

Thus, we need to distinguish *levels* in the design of a digital product (and maybe some other kinds of products). The circuit that connects the chips is one level; each chip's design is another level. In an FPGA, the interconnection of primitive cells is one level, while the primitive cells themselves are another level. In the ideal future we will want the design be free at all levels. Under present circumstances, just making one level free is a significant advance.

However, if a design at one level combines free and nonfree parts—for example, a “free” HDL circuit that incorporates proprietary “soft cores”—we must conclude that the design as a whole is nonfree at that level. Likewise for nonfree “wizards” or “macros,” if they specify part of the interconnections of chips or programmably connected parts of chips. The free parts may be a step towards the future goal of a free design, but reaching that goal entails replacing the nonfree parts. They can never be admissible in the free world.

Licenses and Copyright for Free Hardware Designs

You make a hardware design free by releasing it under a free license. We recommend using the GNU General Public License, version 3 or later. We designed GPL version 3 with a view to such use.

Copyleft on circuits, and on nondecorative object shapes, doesn't go as far as one might suppose. The copyright on these designs only applies to the way the design is drawn or written. Copyleft is a way of using copyright law, so its effect carries only as far as copyright law carries.

For instance, a circuit, as a topology, cannot be copyrighted (and therefore cannot be copylefted). Definitions of circuits written in HDL can be copyrighted (and therefore copylefted), but the copyleft covers only the details of expression of the HDL code, not the circuit topology it generates. Likewise, a drawing or layout of a circuit can be copyrighted, so it can be copylefted, but this only covers the drawing or layout, not the circuit topology. Anyone can legally draw the same circuit topology in a different-looking way, or write a different HDL definition that produces the same circuit.

Copyright doesn't cover physical circuits, so when people build instances of the circuit, the design's license will have no legal effect on what they do with the devices they have built.

For drawings of objects, and 3D printer models, copyright doesn't cover making a different drawing of the same purely functional object shape. It also doesn't cover the functional physical objects made from the drawing. As far as copyright is concerned, everyone is free to make them and use them (and that's a freedom we need very much). In the US, copyright does not cover the functional aspects that the design describes, but does cover decorative aspects <https://www.copyright.gov/title17/92chap13.html#1301>. When one object has decorative aspects and functional aspects, you get into tricky ground [2].

All this may be true in your country as well, or it may not. Before producing objects commercially or in quantity, you should consult a local lawyer. Copyright is not the only issue you need to be concerned with. You might be attacked using patents, most likely held by entities that had nothing to do with making the design you're using, and there may be other legal issues as well.

Keep in mind that copyright law and patent law are totally different. It is a mistake to suppose that they have anything in common. This is why the term “intellectual property </philosophy/not-ipr.html>” is pure confusion and should be totally rejected.

Promoting Free Hardware Designs Through Repositories

The most effective way to push for published hardware designs to be free is through rules in the repositories where they are published. Repository operators should place the freedom of the people who will use the designs above the preferences of people who make the designs. This means requiring designs of useful objects to be free, as a condition for posting them.

For decorative objects, that argument does not apply, so we don't have to insist they must be free. However, we should insist that they be sharable. Thus, a repository that handles both decorative object models and functional ones should have an appropriate license policy for each category.

For digital designs, I suggest that the repository insist on GNU GPL v3-or-later, Apache 2.0, or CC0. For functional 3D designs, the repository should ask the design's author to choose one of four licenses: GNU GPL v3-or-later, Apache 2.0, CC BY-SA, CC BY or CC0. For decorative designs, it should suggest GNU GPL v3-or-later, Apache 2.0, CC0, or any of the CC licenses.

The repository should require all designs to be published as source code, and source code in secret formats usable only by proprietary design programs is not really adequate. For a 3D model, the STL format <https://en.wikipedia.org/wiki/STL_%28file_format%29> is not the preferred format for changing the design and thus is not source code, so the repository should not accept it, except perhaps accompanying real source code.

There is no reason to choose one single format for the source code of hardware designs, but source formats that cannot yet be handled with free software should be accepted reluctantly at best.

Free Hardware Designs and Warranties

In general, the authors of free hardware designs have no moral obligation to offer a warranty to those that fabricate the design. This is a different issue from the sale of physical hardware, which ought to come with a warranty from the seller and/or the manufacturer.

Conclusion

We already have suitable licenses to make our hardware designs free. What we need is to recognize as a community that this is what we should do and to insist on free designs when we fabricate objects ourselves.

Footnotes

1. As used here, “digital hardware” includes hardware with some analog circuits and components in addition to digital ones.
2. An article by Public Knowledge gives useful information about this complexity <https://web.archive.org/web/20211203021432/https://www.publicknowledge.org/assets/uploads/documents/3_Steps_for_Licensing_Your_3D_Printed_Stuff.pdf>, for the US, though it falls into the common mistake of using the bogus concept of “intellectual property” and the propaganda term “protection </philosophy/words-to-avoid.html#Protection>.”

Most of this article was published in two parts in *Wired* in March 2015.
