Using the GPL/LGPL for HDL designs
Open questions from a designer's perspective

Javier Serrano

August 1, 2014

1 Introduction

The purpose of this document is to provide the stewards of the GNU GPL [1] with a number of open questions about the use of GPL/LGPL for Hardware Description Language (HDL) designs. The style is informal. I do make an effort to articulate these doubts clearly, so as to enable the GPL stewards to understand the issues designers face and see what is the best way to transport the concepts and values of free software into the domain of HDL designs. The use of first person singular in this document is just a convenience. If I can claim anything in this domain it's that I have more unanswered questions than most. The text is longish for the amount of actual information it contains. As Pascal said, I lacked the time needed to make it shorter. And the talent.

Now, why should GPL stewards care about HDL? Copyleft has proven to be a truly revolutionary concept in software. Many projects, such as the Linux kernel and the GNU Compiler Collection (GCC), are striking proof of its success. It should come as no surprise that designers in other domains wish to foster similar reciprocal behaviors in their fields. By reciprocity I mean that licensees who decide to publish modifications of an original design should do so giving to their own licensees the same freedoms they received from the original licensor. In the case of software, the GPL has established a sound legal basis to kick-start and maintain this virtuous circle of sharing. In fact, it has been so successful that many HDL designers are using GPL – and its unfavored cousin LGPL – for their designs, mostly assuming that all the confidence people have gathered in the software world regarding the soundness of GPL and its beneficial effects applies directly to their case. However, if HDL is software[1], it is definitely a special kind of software, so to which extent the assumption above holds is a legitimate question.

OK, but I still have not answered why the GPL stewards should care about hardware and HDL. To explain it, I first need to introduce Andrew

[1]If one takes software to mean "instructions which tell a computer what to do", then HDL can be thought of as software in which the "what to do" is "how to make a computer".

"bunnie" Huang's *layers of openness* concept [2]. The case for freedom in software development is very strong because that freedom is very easy to exercise once it is allowed. Many people can launch a text editor and code. Therefore, the loss for not having freedom to share software is great. In the hardware world, contribution is sometimes more complicated. There are fewer people who know how to design a Printed Circuit Board (PCB) than people who know how to develop software. However, in this domain – as in many others – the increasing availability of gray matter and the cultural influence of free software are changing things drastically. People routinely share hardware designs in places like the Open Hardware Repository [3], using licensing schemes specifically developed for hardware, such as the TAPR OHL [4], the CERN OHL [5] and the Solderpad license [6]. People who license their PCB designs under these licenses allow all their licensees to study, modify and publish the licensed material. Licensees can also easily build hardware based on the licensed design information, using relatively standard and straight-forward procedures. These include procurement of bare PCBs through PCB manufacturing service providers and purchasing of components through online distributors. There are many such providers competing in the market, so access to these services can to a large extent be considered a commodity. The fact that the licensed information typically does not specify how to build the chips in the PCB out of silicon and plastic is not a big concern currently, because that freedom would be hard to exercise by most. So we are pushing freedoms one more layer down. We want software to be free, of course, and now we want the first layer below software to be free as well, because there is also a lot of sharing potential there.

OK, so far so good. We want free software *and* hardware, and we have appropriate legal instruments to share both. But here comes a strange beast called *HDL*, which sits half-way in between. An HDL description can serve as the basis for simulation of a design in a computer, behaving in that case as traditional software, "interpreted" by another piece of software called a *simulator*. It can also serve as a basis for the manufacturing of an Application Specific Integrated Circuit (ASIC), behaving in that case in the same way as a PCB design. Finally, it can serve as the basis from which to generate a binary file which can configure a programmable logic chip such as a Field Programmable Gate Array (FPGA). This last case could be assimilated to that of software from which a binary firmware file is generated to be run in a microcontroller.

HDL is very relevant to the mission of the FSF. For example, many network appliances, such as switches and routers, are increasingly built out of a CPU and an FPGA. The FPGA carries out the time-critical work, playing the role of a co-processor, and its configurability allows vendors to better manage design risk and react fast to changing requirements. Many FPGA chips incorporate CPUs to allow this combination of a processor and programmable logic inside the same chip, and some CPU vendors have plans to include programmable logic inside their chips too. The frontier between code running in the CPU and HDL used to synthesize co-processing logic is increasingly blurred [7], so all arguments in favor of running free software in the CPU apply directly to the HDL which generates the accompanying logic.

One goal of this article is to show that neither the GPL family nor the licenses which have been specifically developed for hardware designs are a perfect fit for HDL. I focus on LGPL and GPL because they are – in that order – the two most widely used licenses by HDL developers who want to share with a copyleft mindset, as evidenced by the designs published in OpenCores [8].

One solution is of course to develop a new license which works well with HDL. This approach has been tried by Julius Baxter [9] and Eric Anderson [10]. Reading them, I find myself in agreement with many of their arguments regarding the unsuitability of LGPL/GPL. It would be nice, however, if the stewards of GPL took an interest in HDL and proposed a licensing scheme that they support. Users of such a license would then have the backing of a well-established institution, and good reassurance in terms of legal soundness and future evolution. I am conscious that GPL3 was specifically worded using very neutral language, with the intention of ensuring its applicability in domains other than software. It remains true, however, that the overwhelming majority of its users are software developers. These users guarantee a very tight scrutiny of the validity of the license in their domain. LPGL/GPL licensors are much less numerous in HDL design, so I think it is useful to trigger a specific discussion in this area.

In fact, there has already been a review of LGPL/GPL in terms of their appropriateness for HDL cores [11]. It was published by Eli Greenbaum, who is a lawyer and therefore much more competent on legal matters than myself. However, reading it I had the – maybe misguided – impression that it is written from a different perspective, namely that of commercial companies wanting to evaluate the risks involved in using LGPL/GPL to license their HDL cores. I would like to take a different approach: that of a designer wishing to use those licenses and help their stewards improve them if necessary for the case of HDL designs. Regardless of its vantage point, Eli's analysis is very thorough and contains very useful material for my purpose.

In the following, I give a very quick overview of a typical HDL development flow in section 2, limited to what one needs to understand for the purpose of discussing licensing. Then I enumerate a number of open questions in section 3.

2 The HDL development flow

The flow for ASIC design is very well described in Eli's paper [11]. I am going to briefly describe the flow for FPGA design using Xilinx tools and FPGAs, because that's the one I know best. The ideas should apply to FPGA design using other major vendors such as Altera, Microsemi and Lattice.

An FPGA can be summarily described as a sea of configurable logic blocks (CLBs) which are linked by programmable interconnect resources. The process whereby the functions of each CLB and the connections of the interconnect are established is called *configuration*. The configuration bits are typically taken from a persistent memory (e.g. flash) at power-up and fed to the inputs of each CLB and programmable interconnect cell. The FPGA effectively behaves as an ASIC from then on, with a fixed functionality defined by the configuration bitstream.

In order to generate a configuration bitstream, the developer describes the desired behavior of the circuit in HDL. There are a variety of HDLs. The most common are Verilog, VHDL and SystemVerilog. For our purposes, they are just descriptions of behavior in text. A process called *synthesis* takes HDL code as an input and produces a netlist using primitives (such as logic gates and flip-flops) supported by the FPGA chip targeted by the developer. That netlist is then fed to a *place and route* (P&R) tool provided by the FPGA vendor, which decides on a geographical location inside the chip for each primitive and generates configuration information for the programmable interconnect so that the connectivity specified in the netlist is respected. Finally, the bitstream generation tool takes the result of the P&R tool and generates a binary which can be used to configure the FPGA.

Now, from a free software licensing perspective, how is this process different from the well-known process of generating a binary from free software sources using a compiler? There are a couple of potentially relevant differences:

- The HDL source going into the synthesis tool is very rarely completely free [12]. Often, some components need to be instantiated using text files provided by the FPGA vendor or produced by software from the same vendor. These files are typically "all rights reserved". The copyrightability of these files could be challenged because there is no creative content in supplying hundreds of instantiation options to primitives, but there is also sometimes some glue logic which is "creative" in a sense. This work could in principle be done by hand instead of having tools generate the files, but it would be extremely tedious and error-prone. See subsection 3.5 for a discussion about whether the GPL concept of *System Libraries* could apply here.

- In the case of complete FPGA designs, bitstreams are typically published along with the sources because, as opposed to the case of software, there is no guarantee to get the same binary bitstream from two consecutive runs of the same software toolchain on the same HDL sources. This is because the P&R process can be configured to start placing using a random seed each time, or to generate a new random seed and restart the placement if it sees that its first choice has little chances of success. This is important because the distribution of the binary bitstream triggers obligations under GPL and LGPL as we will see later.

Although it is in principle not relevant to the discussion on the licensing of the HDL sources, I think it is important to mention that the software tools themselves (synthesis, P&R and bitstream generator) are non-free. So far, no FPGA vendor has seen an interest in freeing the details of the internal architecture of their chips, which would allow the development of free software tools to target FPGAs. The best we can do today in terms of sharing is publishing the source files for which we hold the copyright, along with instructions to produce the rest of the files (if needed) using the proprietary tools. This forces anybody who wants to see the complete set of design files to own a copy of those tools. And of course, ownership of those tools is absolutely needed to produce a bitstream, since there is no other way of doing so.

Another very important aspect of FPGA design is simulation. There are free simulators like Icarus [13], with varying degrees of coverage in terms of language features for a given HDL. Here the problem is that some of the primitives provided by FPGA vendors, and instantiated in the HDL design, are provided with simulation models that free simulators cannot use. These models are either binaries compiled for a given proprietary simulator or HDL sources encrypted with a key which is only made available to vendors of such simulators.

In addition to performing a purely functional simulation of their designs, FPGA designers can sometimes be interested in simulating their design including the asynchronous delays introduced by the P&R process. The P&R tool can be asked to generate an HDL file which can be fed to a simulator. This HDL file contains the same inputs and outputs as the original design, but inside it's flattened and contains just a netlist of gates, flip-flops and asynchronous delays. These models can in principle be simulated by a free simulator. The only problem is that the designer must have access to the HDL file produced by the P&R tool and to the HDL files containing the models of the primitives, all of which are clear text but non-free.

As can be seen from the previous discussion, many of the obstacles for sharing have more to do with certain policies of the FPGA vendors than with the suitability of GPL for licensing the code provided by a designer. I think however that these issues are still relevant because they might trigger ideas to cope with these situations, or actions to be carried out in parallel with the actual license development efforts. In section 3 I delve into questions which are more specifically related to the suitability of GPL/LGPL, even in this reduced scope.

2.1 Some words on the ASIC design flow

Unlike the case of FPGA design, the final result of ASIC design is the chip itself, not a binary bitstream which configures it. The difference was more clear-cut in the past. Nowadays, some FPGA vendors like Altera and Xilinx offer designers the possibility of shipping them pre-configured, non-reconfigurable chips made by using an HDL design provided by the designer to target an FPGA. This can be a good middle-of-the-road solution for designs which are stable and require a reduction in costs because they are to be deployed in large quantities. Other vendors, like Microsemi, sell FPGAs where the configuration memory is made using antifuse cells, which are one-time programmable. This can be useful e.g. in environments where radiation could result in bit flips in the configuration memory. Antifuse technology is immune to such effects.

Even if the end result is in principle quite different, the design flow for ASICs has many things in common with that for FPGAs. HDL sources go in and binary files come out. In the case of the ASICs, these output files will allow a foundry to manufacture the chip. There is also synthesis, P&R and binary output file generation. Similarly to the FPGA case, in addition to the HDL code describing the design, the designer needs access to a library of primitives or standard cells that the toolchain will use when floorplanning or simulating the entire design. The design and characterization of the behavior of these primitives (and their interconnection) is one of the hardest tasks for a specific semiconductor manufacturing process and foundry. These files are invariably non-free.

One of the most widely used approaches for designing a custom ASIC is testing the HDL code in an FPGA-based prototyping platform, using FPGA-specific primitives. Once the design is stable, the same HDL sources can be used to generate the files for manufacturing the integrated circuit, using standard cells which are specific to a foundry and a process.

3 Open questions

In the following, I focus mostly on FPGA design because I don't know ASIC design well enough. Also, when I use the word *core*, I mean in fact *soft core*. These are HDL designs which get instantiated inside larger designs. In this context, I assume that the *covered work* of GPL3, or the *combined work* of LGPL3, is the configuration bitstream for the FPGA if we are discussing conveyance of non-source forms. In the case of simulation, it would be the complete set of sources needed for running that simulation, excluding the simulator itself. The order of the questions is roughly chosen to illustrate the possible difficulties with GPL first, and move on to LGPL as a second choice if needed [14]. The questions are open even if they don't look that way. My tentative answers just represent the best of my current understanding, and I hope they trigger discussion and correction.

3.1 What does "linking" mean for HDL?

People generally ask the question in these terms. The GPL text prefers to use the notion of *corresponding source* to discuss the extent to which sources should be published for a covered work. The corresponding source is "all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities". If the work is the FPGA bitstream and one wants to distribute it, one needs to make available under the GPL all sources which will allow to generate it. Unfortunately, as we have seen in section 2, the designs which do not include any proprietary source files are not very numerous. Now, what happens if I just want to publish an HDL core in e.g. OpenCores? Then the covered work is just the source of that core. It has no ambition to become an FPGA bitstream, but it can certainly become *part* of one. The problem then is that, assuming the user of the core wants to distribute the complete bitstream, the core will only be usable in the very limited number of designs whose sources are all GPL.

Since the GPL was worded using very neutral language, it can sometimes be difficult to picture what one is really referring to when using a given term. This applies to "linking", but also to "object code", "compiling", etc. It is definitely against all intuition for an HDL designer to refer to an LGPL-licensed core as a "library". Important terms, however, are in general defined in the license texts themselves so, to a certain extent, this is more of an inconvenience than a real problem.

3.2 How does one ensure that products come with instructions for bitstream upgrade?

One of the most compelling reasons for using FPGAs is their reconfigurability. Section 6 of the GPL [1] specifies that conveyance of object code needs to be accompanied by upgrade instructions only if the object code is embedded in a *user product*. A user product is defined in the same section as "(1) a consumer product, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling." I don't think most boards using FPGAs would qualify. This is not a big problem for using GPL. It looks rather like a missed opportunity. There must be a reason why the GPL authors limited the scope of this clause to user products. It may be because restrictions in the license should respond to an identified source of abuse, and user products were the only source of abuse well identified at the time of writing.

3.3 Is the use of GPL desirable at all in this context?

I think it is possible to use GPL, but the whole approach might be self-defeating. GPL is a means to an end. The end is to share in a fair and efficient way. Because of the restrictions imposed by FPGA vendors, and because of the lack of good free alternatives to many HDL cores, it is very difficult to come up with a complete FPGA design which is made of GPL sources exclusively. Cores published under the GPL have chances of being less used as a result. In the case of software, there was an initial impetus by the GNU project to generate a critical mass of GPL code that would make it appealing for others to contribute. In the case of HDL, there is no equivalent of the GNU project. The closest I can think of is OpenCores, but even there they have more LGPL cores than GPL ones, and the degree of quality varies widely.

3.4 How to comply with LGPL3 paragraph 4d?

Assuming that, based on the arguments of subsection 3.3, we decide that weak copyleft is a better approach, would there be any show-stopper for the use of LGPL3? I see clause 4d in the LPGL3 text [15] as a possible obstacle. It specifies that a developer conveying a combined work, which I take for the sake of argument to be the bitstream, must make it possible to upgrade the bitstream with a new version of the *library*, which in this case is the LGPL-licensed HDL code. The only easy way I know of for doing that, is to supply all source files, including those whose distribution is restricted by the FPGA vendors, or their post-synthesis netlist representation, whose distribution is also restricted (see 3.5 for a possible way out). This kind of takes us back to the same problem we described for GPL. In the software world, one can supply a binary for a given part of a program, and link that binary with a library licensed under LGPL. The binary is a unit which can easily be relocated in memory. In the case of HDL designs for FPGAs, the whole design is flattened by the synthesis tool before logic optimization takes place. This optimization can shuffle gates and flip-flops around even across module boundaries, and it is most efficient when it operates at the netlist level. There are ways to supply binaries to be merged with the rest of a design, but they result in less efficient final results due to the inability to fully optimize. In addition, these binary methods are very vendor-specific. The arguments above apply as well to the case of third-party cores and top-level files for which the licensor restricts distribution rights.

3.5 Can “System Libraries” save the day?

We have seen that one of the main issues when one tries to use GPL or LGPL for HDL designs is the inability of the designer to distribute all files which are involved in the preparation of a bitstream or the running of a simulation. GPL allows the non-distribution of these files if they can be considered part of the so-called *system libraries*. These are defined as “anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form.”

This definition clearly does not cover third-party proprietary cores. There are a number of designs, however, which don’t use them. They just rely on sources from the designer and from the FPGA tool vendors. Let’s see if the system library concept could apply to these designs. We need to consider two cases:

- If the end goal is the simulation of the design, the files we need to consider are the models of the primitives. One example is the *unisim* library in the case of Xilinx. The *major component* here would be the ISE or Vivado software suites, with which unisim is usually packaged. The problem is that these files do not enable use of the work with this major component *only*, but rather (and most often) with other, third-party, simulators. So, the only chance of being allowed not to distribute these files is if they “implement a Standard Interface for which an implementation is available to the public in source code form.” Well, the only implementation of the unisim library I know is the unisim library itself, which is available to the public in non-free source code form if that public has purchased ISE or Vivado. It is unclear to me to which extent unisim – and other primitive libraries – would qualify as a system library under GPL.

- If the end goal is the generation of a bitstream, the files we need to consider are the instantiations of primitives generated by the software provided by the FPGA vendor. The applicability of the system library concept here is also unclear to me. These files are not packaged with the vendor tools, but generated by them.

3.6 What about ASICs?

The GPL3 text says: “Copyright also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.” One such law is the *Semiconductor Chip Protection Act of 1984* [16]. This begs the question of what “conveying” should mean in the context of ASICs. If I ship a smart TV with GNU/Linux running on it, then that's conveying, as section 6 of the GPL text says that one of the ways of conveying non-source forms is including "the object code in, or embodied in, a physical product," and then I should make the corresponding source available using one of the possible ways specified in the GPL. I think it's pretty natural to extend this case to FPGAs, replacing the smart TV by a PCB hosting an FPGA and the GNU/Linux binaries by the FPGA bitstream sitting in some kind of persistent memory in the board. But what about ASICs? Does shipping a product containing an ASIC whose design sources are GPL'd constitute "conveyance"? This is shipping atoms, not bits. Does that make a difference?

There is also a subtle difference between the LGPL 2.1 and LGPL3 texts, other than the fact that LGPL3 uses much more neutral language and its application to material other than traditional software feels more natural. Take the case of some of Samsung's DTV system-on-chip ASICs, like the SDP83 B-Series. They include an OpenRISC 1000 core which is licensed under LGPL 2.1 or later. Samsung complies with LGPL 2.1 by publishing the sources of the OpenRISC 1000 in their website. But what about the rest of the design? Should a licensee be allowed to make a new ASIC by upgrading to a new version of the OpenRISC? LGPL 2.1 specifies that the licensor of the whole design should make that possible only "if the work is an executable linked with the Library," so my interpretation is that Samsung does not need to publish the rest of the design in whatever form to make an upgrade possible, since the work is not an "executable." But what if the OpenRISC 1000 had been licensed under LGPL3 or later? LGPL3 does not talk about executables anymore. In that case, would Samsung have been requested to distribute enough of their own part of the design to enable a third party to build a new ASIC with an upgraded OpenRISC?

Summary

In this document I have tried to provide a basis for discussion on where HDL developers with a copyleft mindset should go in terms of licensing strategies. There are three possible paths I can see:

- Modify existing copyleft-like licenses which were written with hardware in mind [4, 5]. This was already discussed in the CERN OHL mailing list [17], and it was deemed cumbersome and unnatural. A quite radical re-rewrite would be needed, at least in the case of the CERN OHL.

- Write new licenses [9, 10]. This has the advantage of being able to get things right from square one and using terms which HDL designers can easily relate to. One down side is license proliferation.

- Try to see how the GPL family of licenses can be used for HDL designs. This involves clarifying concepts within this context [18] – ideally through an official document from the FSF – and trying to find solutions for issues which are specific to HDL design, possibly in collaboration with the tool vendors. The main advantage is natural integration for the use case (simulation) where HDL behaves as traditional software and the involvement of the FSF which guarantees a solid legal foundation and future support and evolution.

My impression is that the third way has not been adequately explored yet, and this document is my humble attempt to kick-start the process. The option of having FSF or another institution adopt one of the new HDL licenses as a starting point for their own new license should also be discussed. This is an important debate, because the number of HDL designers and deployed FPGAs is constantly increasing, and providing these designers with a solid legal platform for sharing can result in great societal benefits. So, where do we go from here?

Acknowledgments

All errors and misunderstandings in this document are exclusively my own. There would be even more of them had it not been for the help of Javier García Lasheras, Alessandro Rubini and Tomasz Włostowski.

References

[1] GNU General Public License, `http://www.gnu.org/licenses/gpl.html`

[2] `http://www.eetimes.com/document.asp?doc_id=1320638`

[3] `http://www.ohwr.org/`

[4] `http://www.tapr.org/ohl.html`

[5] `http://www.ohwr.org/cernohl`

[6] `http://solderpad.org/licenses/`

[7] See e.g. `http://www.eee.hku.hk/~hso/borph.html`

[8] `http://opencores.org/`

[9] Julius Baxter, *Open Hardware Description License*, `http://juliusbaxter.net/ohdl/`

[10] Eric Anderson, *Free HDL License*, work in progress, https://github.com/ewa/free-hdl-license/wiki

[11] Eli Greenbaum, *Open Source Semiconductor Core Licensing*, Harvard Journal of Law & Technology, Volume 25, Number 1, Fall 2011, http://jolt.law.harvard.edu/articles/pdf/v25/25HarvJLTech131.pdf

[12] See https://github.com/ewa/free-hdl-license/wiki/Vendor-core-rights for a discussion of vendor-supplied cores.

[13] http://iverilog.icarus.com/

[14] https://www.gnu.org/philosophy/why-not-lgpl.html

[15] https://www.gnu.org/licenses/lgpl.html

[16] http://en.wikipedia.org/wiki/Semiconductor_Chip_Protection_Act_of_1984

[17] See e.g. http://lists.ohwr.org/sympa/arc/cernohl/2013-07/msg00029.html (you might have to click on "I am not a spammer" first and then click on or paste the URL again.)

[18] Eric Anderson's ideas are relevant to this discussion. See https://github.com/ewa/free-hdl-license/wiki/GPL-like-license and https://github.com/ewa/free-hdl-license/wiki/GNU-LGPL-like-License.