EMBECOSM™

ESL in the Future:
The Impact of Open Source

Jeremy Bennett, Chief Executive, Embecosm
jeremy.bennett@embecosm.com

Introduction

You will notice I am not using any PowerPoint slides. Well I want to tell you a story, and for a story, words are better than pictures.

My story is about why SystemC was an important turning point for the EDA industry. Not for its technology, but because of the way that technology was made available. It was the first major standardization effort in EDA to make an open source implementation central to its work. The successes and failures of SystemC can tell us a lot about where EDA might go in the future and how we should prepare for that future.

But before I tell you that story, we need to understand some history, some law and some commercial realities.

Hacker Culture

The modern *free and open source software* movement, often known by its acronym, FOSS, can trace its origins back to model railway engineering. The MIT *Tech Model Railroad Club*, founded just after the second world war, specialized in automation of their layout. When an IBM 704 computer was installed on campus in 1959, a group of friends from the model railroad club would sneak inside the computer room after hours to attempt to program the machine.

In US slang, the term “hack” means a student prank, and these early unofficial programmers became known as “hackers”. Their work was characterized by informality and openness, with an emphasis on exploration and stretching the programmer's abilities. While MIT may have been first, students at Universities with early access to computers, were quick to join in the fun.

The GNU Manifesto

By the end of the 1970s, the world of software was changing. Computers, if not yet personal, had become mainstream. There were well paid jobs to develop challenging software and explore the limits of computer engineering. You didn't have to do it as a hobby in your spare time.

But the hacker culture was not dead. In 1983, a young MIT programmer, Richard Stallman, published his “GNU Manifesto”. What is GNU? Gnu's Not Unix. A self referential acronym capturing the goal of a collaborative project to develop a free replacement for Unix. Coinciding with the early days of what we now call the Internet, the project grew rapidly. In 1985, Stallman set up the Free Software Foundation, a non-profit foundation to support the work of GNU and free software

in general. By 1991, with incorporation of the Linux kernel, the goal of a free replacement for Unix had been achieved.

What Does "Free" Mean?

The GNU Manifesto talked sloppily of providing "free" software. However the word "free" has two distinct meanings. It can mean that something does not have to be paid for, as in, "Jeremy can I get you a free beer after your talk". However it can also mean free in the sense of "freedom".

It is this latter sense of "freedom" which was intended in the GNU Manifesto. That the user should be free to do what they wish with the software. In particular it does *not* mean that you are not allowed to make money from the software.

With this in mind, two of the engineers behind Netscape, Bruce Perens and Eric Raymond set up the *Open Source Initiative* in 1998 to help market the concept of free software. The OSI advocated the use of the term "open source software" as less frightening to the business community, a term which is now generally adopted.

The Open Source Definition

The Open Software Initiative is based around a set of criteria, know as the Open Source Definition. There are 10 criteria in all, but they can be summarized into just three principles.

1. The most important is the principle of free distribution. There must be no restriction on any party giving away the software freely, either standalone or as part of another program.
2. Closely allied to this is the right of access to the source code, and this is important because it allows anyone to make what are known as "derivative works". These may be bug patches, or they may be completely new programs, but they are central to the true value of open source software.
3. Finally there is the principle of non-discrimination. By this is meant discrimination against who may use the software, discrimination against the technology on which this software may run, discrimination against what the software can be used for, and discrimination over whether you can use the software commercially.

Legal Frameworks

A common misconception is that open source software has no license or is in the "public domain". Nothing could be further from the truth. Open source software relies on copyright law and licensing to enforce its requirements for openness.

Copyright law has the merit of being largely the same around the world. It is usually free, and in most countries automatically granted to an author. Most importantly the courts have for many years held that software is a creative act covered by copyright law. Open source software works by granting the user a license to the copyrighted software, in exchange for which the user must follow the rules of open source software.

And, in case you are wondering, those licenses do get enforced. The courts may impose punitive damages, payable to the copyright owner, and they may order a

violating product to be withdrawn. So the clear message is: if you use open source software, follow the rules.

Type of Open Source License

Open source licenses can be grouped into two broad categories. The first group are known as “viral” licenses. These licenses grant you a right to use the software with its source and modify it, so long as you pass those same license rights on with any software you derive from the original. Most importantly if you incorporate the open source software into a larger program, you create what is known as a *derivative work*. In this case, *t*he license applies to the *whole* derivative program, not just the original open source software. In this sense the license *infects* any software of which it becomes part, hence the name.

The second broad category are the “non-viral” licenses. These are not so restrictive. They may require you to pass on the rights to the software, but only to the part of the program that was originally open source. Some are even less restrictive, and amount to little more than “do what you like, just so long as you never sue me”.

By far the commonest open source license is the *GNU General Public License*, or GPL. Now on its 3rd version, this is used by around two thirds of all open source projects. It is a truly viral license, and over the past 25 years, a great deal of legal effort has gone into defining what is a derivative work and hence would be covered by the GPL.

The majority of other licenses are non-viral. They range from the GNU Lesser General Public License, which imposes strict obligations on distributing modifications to the original code, through to the MIT and BSD licenses which come into the “do what you like, just so long as you never sue me” category. Two large programs covered by this sort of license are the Apache web server and the Eclipse IDE, both allowing proprietary developments from their code base.

Business Models

Today IBM, holder of the world's largest patent portfolio, make more money from open source software than they do from patents. Red Hat have a turnover of three billion dollars. MySQL was sold for one billion dollars, making a fortune for its VC investors. Last year the Standish market research group published a report suggesting that open source accounted for a drop of sixty billion dollars in revenue for makers of proprietary software. Bad news for the proprietary software makers, but good news for open source businesses and excellent news for customers.

Open source is all around you. 70% of the world's web servers are based on Apache. The majority of front-end EDA design tools are built around Eclipse. Most of you in this audience rely on Linux workstations for your design and verification work.

All these businesses work for two reasons.

For the user, the freedom of open source means no supplier lock in, alternative providers of support and bug fixes, and a marketplace for new features. These are always risks on a project, and it is this risk reduction, rather than the absence of a license fee, that is the true value to the customer.

For the supplier, the Internet makes the marginal cost of distributing software nil. I can afford to give away hundreds or thousands of free copies in order to attract one paying customer.

There are numerous business models for making money from open source. The MySQL approach is to have an open source personal version of the software, and a proprietary enterprise version. Websites like SourceForge host open source projects to drive advertising traffic and revenue. Perhaps the commonest approach is to sell services supporting open source software. This is how IBM, Red Hat, and my own company, Embecosm, make their money.

Fear, Uncertainty and Doubt

There are some myths which surround open source software and some pitfalls for the unwary. Now is a good time to dispel those myths and warn of the pitfalls.

The first myth is that open source software is full of bugs. But any significant open source project will have far more eyes looking for bugs and their fixes than a commercial product. Apache wouldn't have obtained its dominant position in the web server market if it was not extremely robust.

The second myth is that open source software is unsupported. But how many in this audience can say that every bug, or feature request they have submitted to a proprietary tool provider has been dealt with promptly and accurately? With an open source product, your first line of support is the entire community. If they can't help, there are plenty of companies like my own, who will sell you a support service. And with open source you always have the final option of fixing it yourself.

The third myth is that free in the sense of “unpaid for” software is the same as open source software. Not so. If you can't get the source code, you are still at the mercy of the supplier for support and feature provision. If they go bust, get acquired or just change their strategic direction, you can do nothing. All your investment in learning and using that tool is gone for nothing.

The pitfalls are all to do with licensing.

As a user, check what you are getting. A supplier may advertise a tool as being open source and based on Eclipse. However the Eclipse license is non-viral, and all too often you'll find the interesting parts are proprietary and secret. As a user your best friend is the viral license. If your tool comes with the GPL you are safe.

The biggest danger for developers is when incorporating virally licensed code in their products. Those products will themselves then become open source. That is a good thing if it is what you intended. However it is essential that engineering processes keep development of open source and proprietary software separate, to avoid accidental contamination of software that is intended to remain proprietary.

The other pitfall for developers concerns choice of licenses. It is quite usual to construct open source programs by bringing together open source components from various sources. However not all licenses are compatible. For example you can combine GPL and BSD licensed code, but only if the combined code is licensed under the GPL. This is something to be planned right at the start of the project.

The Cathedral and the Bazaar

So back to the main story. Before turning to SystemC, let's look at how open source engineering works in practice. With open source and the Internet comes a different way of working on software projects. It is feasible to have large numbers of programmers in multiple locations all working on the same project. But how do you manage such a project?

Eric Raymond captured the two principle approaches in his 1997 essay “The Cathedral and the Bazaar”.

The first approach suggests that complex software needs to be built like cathedrals, carefully crafted by individual wizards or small bands of mages working in splendid isolation, with no beta to be released before its time. The software is still released with its source code, but between releases only a small band of developers have access to the work in progress, giving very close control.

The second approach seems to resemble a great babbling bazaar of differing agendas and approaches, where anyone can contribute between releases. Out of this a coherent and stable system can emerge only by a succession of miracles. Software is released early, and is released often.

The astonishing thing is that the second approach not only works, but is actually much more productive than the first approach. Indeed almost all major open source projects now operate on the “bazaar” model.

Raymond lists 19 reasons why this should be so, but only a few of them matter. Most importantly the large number of participants mean you are engaged with large numbers of your customers, essential for finding bugs. Secondly those large numbers of eyeballs on the code mean that when a bug is found, someone will see what the fix is and finally the more people that are involved, the better chance you have of great ideas being contributed.

Which is not to say that open source projects don't need leadership. They absolutely do, but it must be guidance of the masses, not dictatorship of the few.

What Projects are good for Open Source

Eric Raymond also tried to identify the main drivers behind successful open source projects.

Perhaps the most important is leadership. Early on this means someone with a software “itch” to scratch. Someone who cares passionately about doing something in a better way. Later on, when the project is bigger, it needs leadership that can ensure the community continues to contribute effectively.

There has to be a big enough community. Enough users, who will try the product and give feedback so it can be debugged and improved. Enough users, that in time some will become developers as well. Eventually, once a project has ten to fifteen active contributors, it will take on a life of its own and essentially live forever.

The majority of open source projects never reach this stage. They continue as the personal hobby of one or two individuals, but nevertheless many represent useful contributions to the open software world.

So what are the commonest itches being scratched. There are certainly some projects that started from scratch. They are the first of their kind, and there is no commercial alternative. The software for Network Time Protocol is a good example in this category.

However most commonly, the desire is to do a better version of a mature piece of commercial software that has become bloated, expensive and is still buggy. Right at the start GNU Emacs replaced the Unix proprietary Emacs editor and the GNU C compiler replaced the Unix proprietary C compiler and a number of commercial

competitors. More recently Linux replaced Unix, and Open Office (on which incidentally this talk was prepared) is starting to replace Microsoft Windows.

SystemC and Open Source

So how does this apply to SystemC and the world of EDA?

SystemC represents a key step for the world of EDA, because of the decision to implement a reference simulator, and to make that simulator open source. It wasn't the first open source EDA software, but it was the first time major industry players agreed to develop open source software.

SystemC comes with its own open source license. It is a non-viral license, essentially of the "do what you like, just so long as you never sue me" variety. That has allowed SystemC to be widely incorporated into commercial offerings from all the major EDA players, none of whom have open sourced their derivatives.

The SystemC license has one quirk. It requires all users to agree to help defend the SystemC trademark. In that sense it is not an open source license – there is a cost to adopting the software. That is why SystemC is not found in any major Linux distribution. The failure to use a standard license was a big mistake.

The SystemC project was, and still is, built as a "Cathedral" and not a "Bazaar". You can only contribute to SystemC if you are a paid member of the Open SystemC Initiative. Releases are infrequent, and so bugs take months or years before they are fixed. Indeed it it hard for ordinary users to even report bugs. As a consequence the latest version will not build on modern versions of Linux or Windows.

Open Source EDA Tools

SystemC is not the only open source EDA tool. Many open source versions of front end tools are available. Rich Porter will talk in detail about some of these later, but here are some examples.

Icarus Verilog and GHDL are open source event driven simulators for Verilog and VHDL respectively. If you want more speed there is Verilator which will convert Verilog into a fast cycle accurate SystemC model. For wave trace analysis there is GTKWave.

There is even a GNU EDA project providing an umbrella for a wide range of free and open source EDA tools, covering both digital and analog design.

Perhaps most significant is the Fedora Electronics Lab, part of the latest release of Fedora Linux. That includes not just front-end tools, but synthesis and layout tools and cell libraries for ASIC development. In fact the only thing missing is SystemC, because of the failure to resolve the licensing problem I mentioned earlier.

Open Source Hardware

What about open source for hardware? There is plenty out there. The OpenCores website hosts dozens of projects and has over forty thousand users. But commercially open source hardware is around 15 years behind open source software.

One of the key issues is licensing. Unlike software, hardware is manufactured, not published. So it is covered by patent law and the law of contract, not copyright and

licenses. This remains a legal minefield for the commercial development of open source hardware, but an interesting one for lawyers.

Indeed so interesting that the *Journal of Information Law and Technology* is running a special issue on this subject. This is an open access journal, and if any of you would like to contribute, then I should be very happy to speak with you afterwards.

The Future of Open Source in EDA

Earlier I spoke of the sort of project that is well suited to open source. Replacements for mature products that are bloated, costly and still have bugs. That could well describe the state of many EDA front end tools and technologies. So perhaps it is not surprising that we see the emergence of tools like Icarus Verilog, GHDL, Verilator and GTKWave.

But the problem is that all these projects are still in the category of "personal itch being scratched". Verilator is largely the work of Wilson Snyder, with just a handful of other occasional contributors. Icarus Verilog is almost all written by just two engineers. GTKWave has one main contributor. None of them have reached that critical mass of 10-15 active contributors which means the project gains a life of its own.

That reflects the small community in which EDA operates. There are perhaps sixty thousand chip designers out there, compared with the millions of software engineers from which projects like GNU and Linux draw their contributors.

Maybe chip design houses should look at their software counterparts, and consider funding development of open source tools. Just think of the difference to Icarus Verilog and GHDL if each major design house contributed just one engineer. Some of you in this audience are the managers who could make this happen. Think of the potential savings in both license fee and risk reduction if this were successful.

The Software Factor

There is another factor driving the future of EDA front-end tools. That is the balance of power in engineering teams. 15 years ago, the cost of developing a new chip was completely dominated by the hardware design team and its tools. Since we reached 90 nanometer, that balance has been reversed. It is not unknown for a hardware design team of 10 engineers to be complemented by a software teams of hundreds. It is now the software engineering manager who is calling the budget shots.

These are managers who are used to Eclipse and Linux and GNU Tool Chains. They don't pay license fees for tools, because the best ones are free, they just pay modest fees for support of those tools. Now they are starting to look at their hardware colleagues and asking why they are spending so much.

These engineers are also becoming customers for EDA technology. Why use a general ISS, when I could have a proper model of an entire SoC to work with. But if I have 500 engineers, I am not going to pay EDA prices. I'll use an open source tool instead. My company, Embecosm, is already doing that with our clients, and as you will hear later this afternoon, we are not the only company in this business.

So open source is coming to EDA and hardware design, whether we like it or not. We can either ignore it while our competitors move ahead, or we can embrace it and be early beneficiaries of all the advantages I have discussed earlier.

The Future for SystemC

This the end of my story so far. SystemC represents EDA's first big corporate attempt in the Open Source world. But what of its future? Will it be, as has been promised for so long, the salvation of front-end chip design.

Well, not as it stands. SystemC has four key problems.

The first is purely technical. SystemC introduces some radical concepts. It was a mistake to try to implement if with the syntactic sugar of C++, a language that is semantically incompatible. Software engineers don't mind good new languages. They'll program in C++ one day, C the next and then on to Java. And in between they'll be writing scripts in Perl, Ruby, Python and TCL. SystemC missed the opportunity to be an elegant new language for digital discrete event simulation.

The second is that all the SystemC tools are intended for software engineers, but are designed by hardware engineers. I have seen some wonderful virtual platform tools from gifted engineers at great EDA companies. They incorporate wonderful tools for debugging the software. But they are for debugging software in the way that hardware engineers think it should be done. Not one uses the GNU debugger running under Eclipse. Yet that is the work horse of every system engineer, and should have been the first tool provided.

The third problem is that SystemC messed up as an open source project. By using the “cathedral approach” with a small group of industry insiders the project has stagnated and fallen behind the times. By using a proprietary license, it has been impossible to incorporate in standard Linux distributions. As we have seen, there are few enough engineers in EDA as it is. If only SystemC had acted as a “bazaar”, drawing on all the talent available, engaging with all the users, releasing early and often and becoming part of mainstream Linux.

But the final problem is the critical one. The software engineers are coming. They now control the budgets. They are by far the biggest potential customer base for SystemC. The biggest SystemC companies turn over a few tens of millions of dollars. Embedded companies like Wind River and Green Hills are 10 times bigger, and far more experienced with managing open source products.

But SystemC has not been a failure. The project and the name may not survive, but the ideas will. There will be a software technology for efficient discrete event simulation of digital hardware, and it will be open source. And future historians will be able to point back to SystemC and its open source reference simulator as the point when it all began.

Companies like Wind River, Green Hills, Code Sourcery, Red Hat and Embecosm know how to get the best out of Open Source. Expect to see organizations like this pick up SystemC, chew it up and spit out a completely new open source technology that will better solve the problems addressed by SystemC.

SystemC was a brave experiment, it established many new ideas. But like so many brave pioneers, it will not ultimately be the winner. The real legacy of SystemC is its introduction of open source into mainstream EDA. Future EDA front end tools will be open source, heavily oriented to, and driven by the software community. Costs will be lower, quality will be higher and risks will be reduced.

And for that we will thank the pioneers behind SystemC.

About the Author

Dr Jeremy Bennett is Chief Executive of Embecosm Limited (www.embecosm.com). We want our users to develop embedded software seamlessly, using standard open source tools, whether the target is an early model of the architecture or final silicon.

Embecosm's services include:

- Comprehensive GNU tool chain porting and optimization for embedded processors.
- Standards based cycle accurate and transaction level hardware modeling, including OSCI SystemC TLM 2.0 compliance.
- Seamless, unified SoC firmware development and debugging from initial model to final silicon.
- Support, consultancy, tutorials and training throughout the product life cycle

Jeremy Bennett is an active contributor to the OpenCores project (www.opencores.org). Contact him at jeremy.bennett@embecosm.com.

This talk was given on 24 September 2009 at the National Microelectronics Institute meeting on Electronic System Level Design in Bristol, UK.

References

The GNU Manifesto, Richard Stallman, 1983. Free Software Foundation (www.gnu.org/gnu/manifesto.html).

The Free Software Foundation. www.fsf.org.

The Open Software Initiative. www.opensource.org.

The Open Software Definition. www.opensource.org/docs/osd.

The Cathedral and the Bazaar, Eric S Raymond, 1999 (first presented May 1997), O'Reilly, ISBN 1-56592-724-9. www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar.

Journal of Information Law and Technology. Editor Prof Philip Leith, Queens University, Belfast. www2.warwick.ac.uk/fac/soc/law/elj/jilt.