EMBECOSM®

Softcores for FPGA:
The Free and Open Source Alternatives

Jeremy Bennett and Simon Cook, Embecosm

Abstract

Open source soft cores have reached a degree of maturity. You can find them in products as diverse as a Samsung set-top box, NXP's ZigBee chips and NASA's TechEdSat.

In this article we'll look at some of the more widely used: the OpenRISC 1000 from OpenCores, Gaisler's LEON family, Lattice Semiconductor's LM32 and Oracle's OpenSPARC, as well as more bleeding edge research designs such as BERI and CHERI from Cambridge University Computer Laboratory.

We'll consider the technology, the business case, the engineering risks, and the licensing challenges of using such designs.

What do we mean by "free"

"Free" is an overloaded term in English. It can mean something you don't pay for, as in "this beer is free". But it can also mean freedom, as in "you are free to share this article". It is this latter sense that is central when we talk about free software or hardware designs. Of course such software or hardware designs may also be free, in the sense that you don't have to pay for them, but that is secondary.

"Free" *software* in this sense has been around for a very long time. Explicitly since 1993 and Richard Stallman's GNU Manifesto, but implicitly since the first software was written and shared with colleagues and friends. That freedom is achieved by making the source code available, so others can modify the program, hence the more recent alternative name "open source", but it is freedom that remains the key concept.

Free software is everywhere. You may be reading this article on a smartphone running Android, using a browser such as Firefox, with the article supplied by a webserver running Apache.

All this works, because software has a marginal cost of distribution that is effectively zero. It costs no more to supply one million copies of a program than a single copy. Instead we can make money from anything associated with that software, whether it is advertising on the site from which it is downloaded, hardware on which to run it, or services to support it. And typically there is much more money to be made in this way if there are a million copies of that software out there.

But hold on. Surely we are supposed to be discussing hardware, not software. Real hardware costs money, and even if it is cheap hardware, a million copies will be a lot of money to give away in order to reap profit from advertising or services.

What we are really talking about is not open source hardware, but open source hardware designs. Like software, designs also have a zero marginal cost of

distribution, so can be provided by the million. And if other things can be sold alongside those millions of copies, then “free” business models can still apply.

Softcores for FPGAs absolutely fall into this criteria. Indeed blasting a bitstream onto an FPGA is functionally little different from loading a program into a computer's memory. So there can be a business case for free softcores. We'll discuss this further later, but for now let us accept that people are motivated to make free softcore designs.

A word on terminology

The use of the word “free”, with its dual meanings has been a source of confusion for decades. The influential software engineer and writer, Eric S Raymond is generally credited with introducing the alternative term “open source”. However that term has also been criticized for not having anything to do with “freedom” at all. While proponents argue over which is best, in the real world, the two are put together in the phrase “free and open source software” or FOSS.

The French have no problem. Very sensibly they have two separate words, *gratuit* (free in the sense of not paying) and *libre* (free in the sense of freedom). So purists sometimes refer to “software libre”. The pragmatists cover all the bases with the phrase “free/libre open source software” or FLOSS.

It is about at this stage in discussions that we start to lose the will to live. We know what we mean and tend to use “free” and “open source” interchangeably. Which is what we shall do for the rest of this discussion. But throughout we mean designs that are both open and freely available.

Which free softcore?

We'll focus primarily on four widely used “free” (in the sense of “freedom”) softcores. Major FPGA providers offer “free” cores, such as Nios or MicroBlaze, but in the sense of “not paid for”, and we are not interested here in that sort of “free”. The four cores we shall consider are the OpenRISC 1000 architecture from the OpenCores community, the Lattice Semiconductor LM32, the LEON3 from Aeroflex Gaisler and the OpenSPARC family from Oracle. All are made available with their RTL and meet the criterion of being “free”. We'll describe the OpenRISC 1000 in most detail, partly because it has the most active community, and then describe the other processors largely by comparison.

The OpenRISC 1000

The OpenRISC 1000 architecture dates from an initiative to develop a freely available RISC architecture by Damjan Lampret and colleagues in 1999. For the first few years the project was backed by their employer, Flextronics, who produced an ASIC version of the design. From the start the project was led by an independent community, opencores.org. While opencores.org hosts well over 1000 projects today, the OpenRISC 1000 remains its flagship project.

The OpenRISC 1000 is not an actual chip design, but an architecture specification for a family of 32 and 64-bit processors. The specification includes options for floating point and vector processing support as well as multi-core implementations. Architecturally it draws heavily on the DLX design of David Patterson and John Hennessy, which itself is closely based on MIPS. There is a regular array of 32, 32-bit registers, with simple addressing modes, and all arithmetic register-to-register.

It is a Harvard architecture in the sense that it has separate instruction and data buses, but instructions and data share a unified memory address space.

The first implementation was the OpenRISC 1200, written in Verilog, which offered 32-bit integer functionality. Although always conceived as being suitable for FPGA use, this first design was also made into an ASIC by Flextronics, requiring around 150 thousand gates plus memory blocks.

A processor is only as good as the peripherals around it, and from an early stage, a reference system-on-chip was developed using free IP blocks, as a way to test the processor design. The OpenRISC Reference Platform System-on-Chip, ORPSoC, adds memory management units, a UART, flash and SRAM controllers, with GPIO, VGA, PS2, Ethernet and audio interfaces. The implementation uses a 4/5 stage pipeline, much like early MIPS designs.

A SoC needs a bus to connect components, and popular buses, such as AMBA are proprietary. OpenCores has developed its own bus architecture, *Wishbone*, which has gone on to be used by many other projects, including the LM32 (or which more later). Wishbone is intended as a "logic bus". It does not specify electrical information or the bus topology. Instead, the specification is written in terms of "signals", clock cycles, and high and low levels.

The Open RISC 1200 and ORPSoC were the mainstay of OpenRISC development for many years, with implementations available for common FPGA development boards.

However, being a freely available design, there is nothing to stop others coming up with their own implementations. For example Raul Fajardo, a research student at Hanover University developed a simpler SoC, *minsoc,* during 2011.

Increasing frustration with some aspects of the OR1200 implementation led Julius Baxter to write a new implementation of the OpenRISC 32-bit architecture, *mor1kx*. This design was intended to eliminate some of the bottlenecks that limited clock speed with the OR1200 design. It has now grown to 3 variants, with different pipeline implementations. “Cappuccino” provides a 6-stage pipeline with MMU and cache support aimed at top performance for designs that will run Linux. “Espresso” is a simpler 2-stage pipeline without cache or MMU support, intended for more deeply embedded applications. “Pronto Espresso” is the simplest of all, also with a 2-stage pipeline and eliminates the branch delay slot.

The original OpenRISC 1000 architecture specification did not conceive of removing the delay slot. However it is a living document, and continues to be refined and extended as required. In this case by allowing branches without delay slots.

The benefit of the reimplementation can be seen in the performance figures, with Cappuccino achieving around 100MHz on common FPGA boards, while the old OR1200 struggles to exceed 25 MHz.

AltOr32 is a stripped down implementation of the OpenRISC 1000 architecture, with just the barest minimum of ISA features, aimed at the very smallest FPGAs.

There are many other projects using OpenRISC, particularly in academic research. For example Stefan Walentowitz at the Technical University of Munich has developed a multi-core implementation. A large team led by Prof Luca Benini at the University of Bologna, and also involving researchers at ETH Zurich is currently developing an ultra-low power multi-core OpenRISC SoC, which will be fabricated by ST Microelectronics.

The reference SoC, ORPSoC, remains a key proving ground for all these processor implementations. Its third revision, released this year provides a much more configurable design flow, making it easy to add and remove peripherals as required. The design is now capable of fitting on sub-$100 FPGA development boards while running Linux.

The Lattice Semiconductor LM32

The LM32 is a configurable softcore, specifically optimized for Lattice Semiconductor FPGA devices. It is available under the Lattice Semiconductor free IP license.

Like the OpenRISC 1000, the LM32 is a RISC design, with 32 32-bit general purpose registers. It is also a Harvard architecture, with separate instruction and data buses but a unified address space. Arithmetic operations are always register-to-register.

Unlike OpenRISC, LM32 is not an architecture specification, but an actual chip design and implementation. It features a 6-stage pipeline which is fully bypassed and interlocked.

Lattice Semiconductor make available a range of free peripherals, allowing SoC implementations to be constructed. These include SDRAM, DDR and DMA controllers, a timer, and GPIO, I^2C and SPI interfaces.

The most-well known user of the LM32 is the MilkyMist project, which has developed an open source video synthesis system for live performance artists. At the heart of the hardware is an FPGA using the LM32 processor. Milkymist engineers have modified the LM32 design and contributed those modifications back to the wider community.

The LM32 is the youngest of the four free designs considered here, and while it is a comprehensive and well supported design, its youth means it has the smallest support community. The fact that it is a processor design, rather than an architecture specification, means the opportunity for reimplementation is somewhat limited.

The Aeroflex Gaisler LEON3

The LEON project was initiated by the European Space Agency, ESA, in late 1997 to study and develop a high-performance processor to be used in European space projects. The objectives for the project were to provide an open, portable and non-proprietary processor design, capable of meeting future requirements for performance, software compatibility and system cost.

Being aimed at space environments, another objective was to be able to manufacture in a Single event upset (SEU) sensitive semiconductor process. To maintain correct operation in the presence of SEUs, extensive error detection and error handling functions were needed. The goal was to be able detect and tolerate one error in any register without software intervention, and to suppress effects from Single Event Transient (SET) errors in combinatorial logic.

All LEON devices are implemented in VHDL. The first, LEON 1, was a test chip to prove the fault-tolerance concepts. LEON 2 was used in the commercially available Atmel AT697. Both LEON 1 and LEON 2 were developed by ESA. LEON 3 was developed commercially, although still as a free design, by Gaisler Research, Jiri

Gaisler having led the original LEON development. Gaisler Research is now Aeroflex Gaisler, and has recently announced the LEON 4 processor.

LEON is not a new processor architecture, but instead is based on the SPARC v8 RISC architecture. This has the advantage that a huge body of SPARC software can be reused. Although all LEON versions are fault-tolerant designs, smaller and simpler non-fault-tolerant versions are provided.

All LEON implementations are based on a 5-stage pipeline. The new LEON 4 adds branch prediction to the pipeline for increased performance. As with OpenRISC 1000 and LM32, free peripheral IP components are provided to build SoCs, but in the case of LEON, using the proprietary ARM AMBA AHB bus. The peripherals are marketed by Aeroflex Gaisler as GRLIB. While originally quite small in number, for LEON 3, the range of peripherals is extensive, with a large number of memory and bus controllers and a wide range of interface blocks.

While the OpenRISC 1000 and LM32 are both primarily aimed at FPGAs, LEON is intended to be used for ASIC. This is particularly true for space applications, where suitable ASIC processes are needed to achieve fault-tolerance. However LEON 3 has proved popular in other applications, not least because of its SPARC compatibility.

The Oracle OpenSPARC

OpenSPARC has the youngest open source pedigree of the four processors described here, but it is the oldest architecture. It arose from Sun Microsystems decision to open source the UltraSPARC T1 design in 2005. Like LEON, this follows the SPARC architecture, but is implemented in Verilog rather than VHDL. In late 2007 Sun also open sourced the UltraSPARC T2 design. Oracle have since acquired Sun Microsystems, and now have responsibility for the UltraSPARC project.

This is the only 64-bit free softcore, and the only true multi-core implementation, although there are research multi-core implementations of the OpenRISC 1000. Unlike the other three softcores, the design is aimed at desktop environments.

Other free softcores

There are a huge number of softcores available, many developed for academic research. The expiration of some of the early RISC patents has meant that free versions of early designs are now available.

Perhaps one of the most exciting is the Bluespec Extensible RISC Implementation, BERI, developed by Simon Moore and his team at Cambridge University. This is a 64-bit implementation of the MIPS ISA, using Bluespec. This project has also been extended to add capability hardware for security applications. The Capability Hardware Enhanced RISC Instructions (CHERI) design runs a version of FreeBSD with capsicum hybrid capability operating system extensions.

This is also an interesting project, in that while fully free and open source, it relies on a technology, Bluespec, for which only proprietary tools are available. However this has in turn spawned a project to develop free Bluespec tools.

Commercial applications using free softcores

Free and open source softcores have found widespread adoption in commercial products both in FPGA and in ASIC, including by some of the largest companies.

OpenRISC was first fabricated into a commercial standalone ASIC by Flextronics in 2003. More recently it has been used by Samsung in their set top box processors,

starting with the SDP -83 'B' series through to the SDP-1003 and SDP-1006 'E' series. A fault-tolerant version of OpenRISC was developed by the Swedish space and defense company ÅAC Microtec, and flew in NASA's TechEdSat last year. Beyond Semiconductor is a Slovenian chip design company founded by some of the original OpenRISC project team. One of its BA family of processors, derivatived from the original OpenRISC, was used by NXP in its JN5148 ultra-low power Zigbee transceiver chip.

As mentioned earlier the LM32 is a relatively new processor. By far its most high-profile use has been in the Milkymist video mixer, which while an open source community project, is intended to deliver a product for commercial deployment. Its primary commercial role is hower to be the free processor IP for Lattice Semi's FPGA products, and as such is most widely seen in their FPGA development boards.

LEON has been used for many space based projects by both the European Space Agency and NSA. The EADS-Astrium SCOC3 is a “spacecraft controller on a chip” developed using the LEON3-FT processor design.

OpenSPARC has found adoption as a reference platform for a number of FPGA system vendors. Xilinx, BEEcube and Digilent all provide FPGA systems with OpenSPARC as the compute resource.

EDA and Software support

Central to any processor's adoption is the availability of a robust software ecosystem. It is often forgotten that before ARM became famous for low-power designs, it was famous for being the first embedded processor with serious software support. The RealView tool suite made it easy for developers to use ARM within their designs, giving ARM that commercial edge from which they have since grown.

This applies just as much to free softcores.

EDA support

All free softcores are distributed as Verilog or VHDL RTL, and are generally well processed by all the major EDA and FPGA vendors tool chains. At the back end, these are proprietary and tied to a particular FPGA vendors technology.

At the front end, there are also free and open source alternatives, and in general softcore communities have been keen to ensure they work with such tools. It allows them to address small companies, and non-commercial entities, for whom commercial front-end tools are not always an option.

So for example we see that OpenRISC and its various SoC platforms can be simulated using Icarus Verilog, the free event driven simulator and is also very well suited to cycle accurate modeling with Verilator. Indeed Verilator models run at around 500kHz, allowing the design to be verified not just with standard verification approaches, but by running test software. This includes executing all 125 thousand GCC regression tests for C and C++, providing a very thorough exercise of the design.

For larger scale software development all four designs described here offer instruction set simulators. In the case of the OpenRISC 1000, its architectural simulator, *Or1ksim*, is also the reference for the ISA, so comparative testing is used as part of design verification. The degree of wider system support provided by such instruction set simulators varies. All provide a degree of modeling of peripherals, and in the case of Or1ksim this is sufficient for it to be able to boot and run Linux

in simulation with reasonable performance – approximately 15 seconds to boot to console prompt.

For those who want to simulate faster than the chip will actually run, the compiling simulator QEMU is available for all four designs, although in the case of LEON and OpenSPARC it is the generic SPARC version, rather than specific to the actual design.

Compiler tool chains

A processor must have a compiler, and all four designs offer the GNU tool chain. A challenge for commercial deployment is to develop not just a functional compiler, but one that is robustly accurate. LEON and OpenSPARC can rely on the standard SPARC compilers, although this has the disadvantage of not offering processor specific optimization. Aeroflex Gaissler has addressed this for LEON with their own variants of GCC, although as a consequence their tool chain is based on a very out-of-date version (GCC 4.4.2). Our company, Embecosm, developed a commercially robust release of GCC 4.5.1 for OpenRISC for the TechEdSat project. The compiler has since continued to be developed by the community, with GCC 4.8 being available. LM32's compiler is not only up-to-date, but has been adopted into the FSF mainline.There is one way that softcores differ from other free IP. Irrespective of the RTL, a softcore is useless without its programming manual. This is a textual document, and easily protected by copyright and existing free and open source licenses. A manufacturer can arguably take the RTL and make an ASIC without regard to the spirit of open source licensing. But the law is crystal clear that they cannot distribute the programming manual without following the license. So ensure the manual is properly protected with the right sort of license. This could be a creative commons license or the GNU Free Documentation License (GFDL).

The most popular licensing schemes for softcore RTL are based on the GNU General Public License (GPL) and the derivative GNU Lesser General Public License (LGPL). For software the difference is that if you use GPL licensed code in a system, its provisions for open sourcing apply to the entire system, not just the GPL licensed code that was incorporated. The LGPL applies the provisions just the the code itself, not the entire system.

FPGA designers have adopted this, assuming that GPL would mean using an IP block would cause the GPL to apply to the whole system, while LGPL would mean that only the IP block itself was covered. However that seems a very gross assumption to make. The difference in the license text is written in terms of linking software together, and IP reuse within a bigger system is definitely not software linking. The GPL documentation is clear it should not be used for hardware, and it seems that without a rewrite, the GPL/LGPL distinction does not apply to softcores.

The OpenRISC 1000 architecture specification is protected under the GNU General Public License. This is an example of poor license choice – for documentation the GNU Free Documentation License would have been appropriate. However changing a license in a document to which so many have contributed is almost impossible, since it would require them all to assent. So the specification is likely to remain GPL.

The original OpenRISC 1200 and the various ORPSoC implementations are all licensed under the GNU Lesser General Public License. As noted above, the choice between LGPL and GPL is probably irrelevant for RTL, but it does provide some open source enforcement, at least for FPGA use.

mor1kx is a completely new reimplementation of the OpenRISC 1000 architecture, and the designer, Julius Baxter, was acutely aware of the licensing issues. He has released his design with his own Open Hardware Description License. This is based on the well established, and permissive Mozilla license, modified to be more suited to softcore RTL.

Lattice Semiconductor's LM32 comes with their own “Free IP” license. In general the main free and open source licenses have been through an approvals process by the Open Source Initiative, which assesses them against 10 criteria which all such licenses must meet. The LM32 “Free IP” license is not such an approved license, and it is not clear if it meets strict criteria to qualify.

The drawback to such an approach is it engenders suspicion (which may be completely unfounded) in the wider community. This leads to a reluctance to contribute, just in case it turns out that this is not a truly free license, and actually allows a corporation to take private ownership of community work. The Open Source Initiative's approvals process is a powerful counter to such suspicions, and use of an approved license is much to be encouraged for a successful free softcore project.

OpenSPARC is very simply licensed under the GPL. This is in one sense heartening, in that, despite the concerns over suitability for softcores, Sun's corporate legal team were happy to use the GPL.

LEON 3 is perhaps the most interesting as all. LEON 1 and 2 were both licensed under the Lesser GNU General Public License, thus following the same approach as OpenRISC. However LEON 3 is licensed under the main GNU General Public License, with the clear intent that if you use LEON 3 in a design then the *whole* design becomes free and open.

The difference is that the entire IP for LEON3 is owned by Aeroflex Gaisler, so they have an option to issue alternative licenses if they so wish. Note that this is not the case for other projects, since they have multiple contributors over many years, all of whom would have to agree to a license change.

Aeroflex Gaisler are quite happy to provide you with a completely private license to LEON 3 – at a price. Indeed that is one way that they make money. So their decision to use GPL is not an altruistic desire to get as much silicon IP as possible opened up, but a clear goal to increase the value of a proprietary license.

We are quite supportive of this approach. Aeroflex Gaisler are being completely clear and giving customers a choice. Either commit fully to the principles of free development and open source everything. Or take it all private and pay for the privilege thereof. In our eyes this is explicitly avoiding the ambiguous “use the open source, but then don't give back approach” that others seem to think is acceptable.

Aeroflex Gaisler can do this because they own the entire IP. However it is an approach that open source communities, which can struggle for funding, might consider. It would require contributions to be contributed to the community, in the same way that the Free Software Foundation requires for its software projects. But the community could then offer restricted licenses for a fee that could be used to fund the community. We are not sure it would work – communities can be very philosophically driven, and might not take kindly to companies being able to buy themselves out of the commitment to free development. But it is a discussion worth having.

These are not the only compilers in town. Oracle encourages users to use their proprietary Sun Studio compiler. SPARC is supported by the generic LLVM compiler, while an experimental version of LLVM 3.3 for the OpenRISC 1000 has been developed by one of us (Simon Cook) and Stefan Kristiansson.

Unix

All four platforms offer Linux support. In the case of OpenRISC, it is part of the official Linux distribution, the latest 3.11 version is available, and commonly used with BusyBox on platforms. LM32 has also offered Linux, although it is not part of the official distribution and at the time of writing, its status and availability is unclear.

LEON3 has its own port of Linux, the Snapgear Embedded Linux. However it has a problem in that it uses very old versions of the kernel. For non-fault tolerant processors it is kenel 2.6.21.1, while the fault-tolerant version only offers an ancient 2.0.x based kernel.

OpenSPARC as befits a design aimed at desktop environments offers not just Linux kernels, but full Linux distributions. Both Gentoo and Ubuntu distributions are supported. OpenSPARC also supports other Unix variants, with both FreeBSD and OpenSolaris available.

RTOS

The OpenRISC 1000, LM32 and LEON3 are all aimed at embedded applications. For larger applications, Linux may be appropriate, but for more deeply embedded uses, a real-time operating system is needed.

OpenRISC is supported by RTEMS, FreeRTOS and eCos, although the status and quality of these ports depends on the activity and motivation of the particular project carrying out the work. For LM32 there is some commercial RTOS support with uC/OS-II, and it also supports the open source ulTRON RTOS.

LEON3 is the big leader in RTOS support, with implementations of many commercial RTOS, such as RTLinux, PikeOS, VxWorks and LynxOS. It also supports the open source RTEMS and eCos.

Licensing free softcores

Licensing of softcores is something of a gray area. In general developers have tended to regard RTL implementations as software. The design is then licensed using standard free and open source software licenses, with the RTL as source code and the bitstream as object code. While this is probably legally defensible for FPGAs, it is not clear that the provisions of such licenses would apply to ASIC manufacture. It is also problematic even with FPGA, because such licenses tend to refer to terminology, such as “linking”, “compiling” and “object code”, which are not directly translatable to FPGA synthesis.

In many cases commercial manufacturers have stuck to the spirit of the licensing, with for example Samsung making the RTL of the OR1200 available through their open source download sensor. Others have not been so forthcoming. Beyond Semi likes to proclaim their OpenRISC heritage until you request the RTL source, at which point they are keen to tell you that it is a completely new design. Similarly

the novel fault-tolerant features of the TechEdSat design have never been made publicly available.

This in part represents the relative immaturity of some of the players in open source softcore design. The same was the case in the early days of free and open source software. However in time, companies learn that if you take this approach, you will lose the community support on which you depend. You also lose the trust of end customers – if you are willing to betray the community which provided the original design, how will you treat your customers. It is thus perhaps not surprising that the largest companies, with past experience in open source software, have taken the most enlightened approach to free softcores.

There is one way that softcores differ from other free IP. Irrespective of the RTL, a softcore is useless without its programming manual. This is a textual document, and easily protected by copyright and existing free and open source licenses. A manufacturer can arguably take the RTL and make an ASIC without regard to the spirit of open source licensing. But the law is crystal clear that they cannot distribute the programming manual without following the license. So ensure the manual is properly protected with the right sort of license. This could be a creative commons license or the GNU Free Documentation License (GFDL).

The most popular licensing schemes for softcore RTL are based on the GNU General Public License (GPL) and the derivative GNU Lesser General Public License (LGPL). For software the difference is that if you use GPL licensed code in a system, its provisions for open sourcing apply to the entire system, not just the GPL licensed code that was incorporated. The LGPL applies the provisions just the the code itself, not the entire system.

FPGA designers have adopted this, assuming that GPL would mean using an IP block would cause the GPL to apply to the whole system, while LGPL would mean that only the IP block itself was covered. However that seems a very gross assumption to make. The difference in the license text is written in terms of linking software together, and IP reuse within a bigger system is definitely not software linking. The GPL documentation is clear it should not be used for hardware, and it seems that without a rewrite, the GPL/LGPL distinction does not apply to softcores.

The OpenRISC 1000 architecture specification is protected under the GNU General Public License. This is an example of poor license choice – for documentation the GNU Free Documentation License would have been appropriate. However changing a license in a document to which so many have contributed is almost impossible, since it would require them all to assent. So the specification is likely to remain GPL.

The original OpenRISC 1200 and the various ORPSoC implementations are all licensed under the GNU Lesser General Public License. As noted above, the choice between LGPL and GPL is probably irrelevant for RTL, but it does provide some open source enforcement, at least for FPGA use.

mor1kx is a completely new reimplementation of the OpenRISC 1000 architecture, and the designer, Julius Baxter, was acutely aware of the licensing issues. He has released his design with his own Open Hardware Description License. This is based on the well established, and permissive Mozilla license, modified to be more suited to softcore RTL.

Lattice Semiconductor's LM32 comes with their own “Free IP” license. In general the main free and open source licenses have been through an approvals process by the Open Source Initiative, which assesses them against 10 criteria which all such licenses must meet. The LM32 “Free IP” license is not such an approved license, and it is not clear if it meets strict criteria to qualify.

The drawback to such an approach is it engenders suspicion (which may be completely unfounded) in the wider community. This leads to a reluctance to contribute, just in case it turns out that this is not a truly free license, and actually allows a corporation to take private ownership of community work. The Open Source Initiative's approvals process is a powerful counter to such suspicions, and use of an approved license is much to be encouraged for a successful free softcore project.

OpenSPARC is very simply licensed under the GPL. This is in one sense heartening, in that, despite the concerns over suitability for softcores, Sun's corporate legal team were happy to use the GPL.

LEON 3 is perhaps the most interesting as all. LEON 1 and 2 were both licensed under the Lesser GNU General Public License, thus following the same approach as OpenRISC. However LEON 3 is licensed under the main GNU General Public License, with the clear intent that if you use LEON 3 in a design then the *whole* design becomes free and open.

The difference is that the entire IP for LEON3 is owned by Aeroflex Gaisler, so they have an option to issue alternative licenses if they so wish. Note that this is not the case for other projects, since they have multiple contributors over many years, all of whom would have to agree to a license change.

Aeroflex Gaisler are quite happy to provide you with a completely private license to LEON 3 – at a price. Indeed that is one way that they make money. So their decision to use GPL is not an altruistic desire to get as much silicon IP as possible opened up, but a clear goal to increase the value of a proprietary license.

We are quite supportive of this approach. Aeroflex Gaisler are being completely clear and giving customers a choice. Either commit fully to the principles of free development and open source everything. Or take it all private and pay for the privilege thereof. In our eyes this is explicitly avoiding the ambiguous “use the open source, but then don't give back approach” that others seem to think is acceptable.

Aeroflex Gaisler can do this because they own the entire IP. However it is an approach that open source communities, which can struggle for funding, might consider. It would require contributions to be contributed to the community, in the same way that the Free Software Foundation requires for its software projects. But the community could then offer restricted licenses for a fee that could be used to fund the community. We are not sure it would work – communities can be very philosophically driven, and might not take kindly to companies being able to buy themselves out of the commitment to free development. But it is a discussion worth having.

Modern licensing alternatives

We cannot conclude on licensing, without noting more recent work on licenses better suited to free and open source hardware development, and drawing attention to three new licenses. These are particularly suited to true open source hardware

development, while softcore development is something of a gray area between software and hardware, which probably needs further work.

The Solderpad Hardware License is based on the well established Apache software license. It is based on protecting the copyright in designs and using that to enforce open source requirements. It is a permissive license so the obligations to pass on benefits to others is modest.

The CERN Open Hardware License version 1.2 has been developed to support the open source hardware work being carried out at CERN and other particle physics laboratories. Like the Solderpad Hardware License it works by protecting the design copyright. It is what is know as a weak copyleft license. Not only does it provide provision for open sourcing between designer and user, it also requires many of those obligations to be passed on to subsequent users. Note that version 1.2 is a major rewrite of the design, and the version which should be used. It is also worth noting that the CERN group feel that this license is not suitable for protecting RTL designs.

The Tucson Amateur Packet Radio Open Hardware License was developed by John Ackerman, an American amateur radio enthusiast who is also a professional lawyer. His was the first serious attempt at writing a license to protect free hardware. It is a more strongly copyleft license than the CERN OHL and draw on both copyright and patent law. The downside of using patent law is that it is much more jurisdiction dependent.

There are two particular good pieces of news in connection with this. The first is that much of the work on the Solderpad and CERN licenses has been carried out by a British lawyer, Andrew Katz, so this is a field in which the UK has a leading position. The second piece of good news is that Andrew Katz and John Ackerman have worked together, and as a consequence the three licenses are compatible. So unlike many software licenses there are fewer issues when a design combines elements with the different licenses.

Business models

We now want to look at business models, both from the perspective of the developers of softcores and the users. It makes sense to start by looking at what has happened with free and open source software.

Today Open source software is a multi-billion dollar business. 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 market capitalization of eight billion dollars on revenues of 750 million dollars. MySQL was sold for one billion dollars, making a fortune for its VC investors.

Open source software is all around you. 60% of the world's web servers are based on Apache. Firefox holds more than a quarter of the browser market. Android based smartphones are hugely outselling the Apple iPhone.

It is worth considering how open source software is developed, because they give insights into how free softcores can be developed.

The Cathedral and the Bazaar

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 (early prototype) 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.

The beekeeper

Much modern open source development is by full-time professional engineers. James Dixon has developed an understanding of how such an approach can work, while still retaining the open source community around it.

He uses the analogy of a Beehive. The commercial organization wants some open source software developed (for example a compiler for their new chip), but does not have the staff in house to do it. They need to persuade the open source community to do that development.

The commercial organization is the beekeeper, the open source community the bees in the beehive, the program is the honey. Like programmers, the bees are free agents, able to fly where they will. However if you given them a nice beehive, and supply them with sugar in the winter, they will happily make honey, which the beekeeper can remove and sell.

So by analogy, the commercial organization must support the open source community. This may be a matter of providing just resources, but increasingly it involves actually funding programmers who contribute to that community.

How to develop a free softcore

We see cathedral, bazaar and beekeepers in use for softcore development. Projects like LM32, LEON and OpenSPARC started out as cathedrals, developed by small groups inside companies before being open sourced. OpenRISC 1000 is different, since it was developed as a bazaar from day 1, and We can personally attest to the chaotic nature of some of the development. However Lattice Semiconductor show the value of a beekeeper approach. Their support, particularly for Milkymist, has allowed their ecosystem to grow through community contribution.

Yet the power of the bazaar is seen with OpenRISC. It has by far the largest ecosystem and most independent variants. It is the only project large enough to run

its own conference each year. It is used in many commercial projects and continues to thrive and grow.

Business issues for users of a free softcore

For the user of a free softcore the advantages are fairly obvious. There are no license fees or royalties available and you have access to community effort. Community effort is a two-edged sword. It is free, but it is hard to direct to your commercial needs. However there are increasingly companies who can provide commercial support – our company provides compiler tool chain development for example.

There are two big counters to this. First you are responsible for your own verification. If the chip doesn't work, then there is no one to sue. For FPGAs you can at least fix the issue, but for ASIC there is a cost of a failed chip.

The second big negative issue is the risk of IP infringement. This is the case for all IP, but with open source IP, everyone can see the RTL, and so there is more likelihood infringement can be discovered. However to put this in perspective, the four main softcores we have described are all well established and either backed by major corporations – Lattice Semiconductor, Aeroflex Gaisler and Oracle – or have previously been used by major corporations – NXP an Samsung for OpenRISC. So the risk in these cases is pretty low.

The bottom line is that there is no real reason not to use a softcore. In general the processor is not the unique selling point of a SoC. So why attract license fees or royalties for it – choose a free version.

The Future

The free softcore communities continue to thrive, as can be seen by the rapid growth of projects on communities such as opencores.org. But perhaps the most interesting recent development is the emergence of crowd funding.

The problem with paying for open source development is that you are at risk of paying for something from which others (possibly competitors) will then benefit without having contributed. There are many models of crowd funding, but the Kickstarter approach, which is essentially about pre-paid sales is of most interest.

You may have seen very recently the Kickstarter project to develop an open source GPU. This will require many people to come together before the project can get off the ground. Unless everyone plays their part it won't happen. Player hoping to come along afterwards and get something for free, risk not having the project start at all.

We are not sure this particular project is going to succeed – we think they have their benefits (the pre-sales) poorly defined. However the general approach is something that may allow competitors to jointly support free and open source development that is to their mutual benefit.

Summary

In summary:

- there are a range of well established free softcores available;
- these free softcores have been deployed in many commercial projects;
- there is generally good tool, software and operating system support;

- there are strong communities supporting free softcores;
- commercial support is available where needed;
- the risks are well understood and mitigated; and finally
- there is no reason not to consider a free softcore for your next project.

Acknowledgments

OpenRISC is a community project, to which we are just two of the contributors. It is the cumulative result of 14 years work by a very large number of people.

About the Authors

Dr Jeremy Bennett is Chief Executive of Embecosm (www.embecosm.com) which provides open source services, tools and models to facilitate embedded software development with complex systems-on-chip. Contact him at jeremy.bennett@embecosm.com.

Simon Cook is a graduate of the University of Bristol, where he achieved joint First Class Honors in Computer Science and Electronics and leads Embecosm's work on the LLVM compiler. He is also lead engineer on the TSB funded MAGEEC project, developing a machine learning framework for compilers to improve the energy efficiency of compiled code.

This paper was presented by Simon Cook at the NMI meeting "Embedded Processors-you've got the power, but which to choose" at Engineers House, Bristol on 24 October 2013.