



Designing and implementing your own CPU or System-on-Chip brings benefits to thousands of researchers and forward-looking businesses, and is being adopted by a growing number of hobbyists. **Richard Smedley** finds freedom in configurable silicon.

For some years (the need for a few binary blobs in the kernel excepted) many readers have run an entirely Free Software stack on their servers, laptops, desktops, and even tablets and phones. But at the silicon level it's another story, with open source hardware limited to a few embedded boards like the Arduino. The good news is that not only are there open source designs for CPUs and Systems-on-Chip (SoC) nowadays, but that it's not too hard to learn to design and make your own. Indeed, there are projects designed to get you started doing just this.

One such of these is OpenCores, which bills itself as "the #1 community within open source hardware IP-cores", backing the claim with a statistic of more than 200,000 registered users. It hosts projects ranging from relatively simple UARTs (universal asynchronous receiver/transmitter) and Ethernet MAC (Media Access Control) LAN implementations right up to the complexity of full OpenRISC chips.

That reference to "IP-cores", rather than CPU cores is an abbreviation for so-called "intellectual property", and is a telling reflection of the proprietary nature of

Space RISC

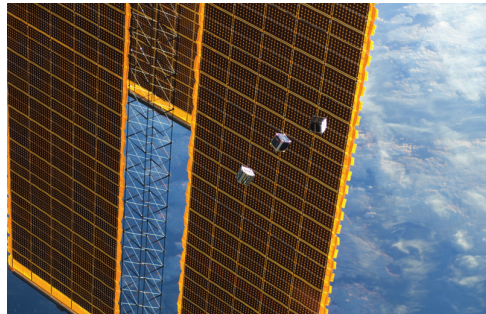
OpenCore has gone beyond earth-bound applications, after students at San Jose State University – funded by NASA's Ames Research Center – designed a 1U satellite, TechEdSat, to evaluate AAC Microtec's implementation of OpenRISC, and perform communications experiments.

The satellite, which was deployed from the International Space Station in 2012, cost less than US\$30,000 to build thanks to the combination of OpenRISC and off-the-shelf hardware selected to be rugged enough for space use.

According to engineers from AAC Microtec, the standard OpenRISC design was modified with fault-tolerant features and toolchain modifications invisible to the end-user software as different from standard OpenRISC spec. The great thing about using an open specification is that these modifications have no barrier in terms of licensing or configuration information, while the flexibility of FPGAs makes prototyping quick and (relatively) easy.

At these prices it's now conceivable that with savvy sponsorship, even schools could launch a satellite with their own custom CPU. However, don't forget you can send a

Raspberry Pi to near-space from your school for 1% of this cost, as David Akerman did when he launched his Pi and a camera on a balloon into the skies over Berkshire.



OpenCores in space: the OpenRISC powered TechEdSat is deployed from the International Space Station.

most work cast into silicon. The fast growth of OpenCores shows that there's enthusiasm and a business need for a more open alternative. Naturally, the opportunity that OpenRISC presents to gives playing with the design of a full-blown modern microprocessor makes OpenRISC useful in universities, and the freedom to explore means another field opened to hobbyists. But what's really driving development is a number of businesses taking advantage of a flexible, cost-effective route to specialist markets.

RISCing it

So, why pursue open CPU architecture, and why go the RISC route? The latter question is the simplest to answer. The case for RISC (Reduced Instruction Set Computing) was well made by IBM researchers in the late 1970s, and producers of the first RISC1 chip at the University of California, Berkeley 30 years ago. Reducing the operation code instructions in silicon (by a factor of 10 at the time), not only simplifies design but frees up space for more registers and cache. Efficient compiler design of the time took away the need for most instruction operation code, and the situation is unchanged today.

OpenRISC is a family of 32- and 64-bit processors with optional floating point and vector processing support. It's a free, open source RISC architecture with DSP (digital signal processor) features and a complete set of free, open source software development tools, libraries, operating systems and applications. The reference design, the snappily titled OR1K (OpenRISC 1000) is implemented as OpenRISC 1200 (OR1200), a synthesisable CPU core released under the GNU Lesser General Public Licence (LGPL).

Writing your own design (for an OpenRISC chip) consists of using a Hardware Description Language (such as Verilog) to describe the chip at the most basic level. Then comes synthesis – conversion to the

list of logic gates and connections used in your chosen FPGA. This latest acronym is a Field Programmable Gate Array, which is a kind of chip that isn't yet set in stone. One FPGA costs a lot more to make than the equivalent processor, but the extra flexibility means that if the design doesn't work the way you want it to, you can simply change it (that's the Field Programmable part).

Free as in almost

The netlist produced is a gate level description, which then usually uses the chip manufacturer's proprietary software to produce the programmed FPGA. For anyone wanting 100% Free and Open Source hardware and design there doesn't seem to be a way around this at the moment. As Embecosm founder and OpenCores stalwart Dr Jeremy Bennett told Linux Voice: "The back-end tools are proprietary to the FPGA manufacturers. Since these tools depend on intimate

Dr Jeremy Bennett of Embecosm showing the OpenRISC SoC implementation on FPGA at an Open Source Hardware Users Group meeting in 2011.

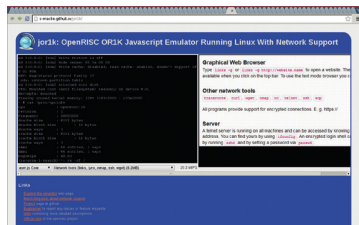


Browsing the hardware

While the idea of building your own CPU appeals to many of us, perhaps you are looking for a way of testing the waters without all the kit. Sebastian Macke of simulationcorner.net has written `lor1k` – the JavaScript OpenRisc 1000 emulator – which gives you the chance to try out open hardware design in that most familiar and comfortable environment, your web browser. `lor1k` works with Firefox and Chrome, though if running locally with the latter you need to run the browser with the command `--disable-web-security`.

The emulated OpenRisc CPU is around 1000 lines of code – a neat introduction to emulation, the OpenRisc architecture, and JavaScript programming all in one! It's also a handy sandbox to test OpenRisc ports, and you could try modifying the emulator to test out ideas for modifying OpenRisc away from the standard implementation.

The project's GitHub pages – <https://github.com/s-macke/lor1k/> – include a wiki with useful and interesting notes on some of the JavaScript optimisations used in the code, as well as speed differences between browsers and a list of the many demonstrations available in the Linux image on the emulator.



If your emulated OpenRisc goes wrong you can just scrap it and start again.

knowledge of the device, it is hard to see how there could be a free and open source implementation, unless the manufacturer chose to do so."

Given the growth of understanding in the advantages of open source methodology, this is not an impossible wish. Meanwhile, we accept that we live in an imperfect world, and continue to make it better – or at least more interesting – to the best of our abilities. At least the Linux-compatibility of the tools is good.

Fabulous Fabless

Designing and fabricating semiconductors is an expensive business. You don't get many opportunities to create prototypes of designs that have tens of millions of transistors in them, and this has led to notable bugs such as the Pentium FDIV bug, which caused the processor to return incorrect results in floating point calculations (Intel eventually had to

recall the chip, but not before considerable damage to its reputation). With even giants like Intel having rationalised its range of offerings in the last decade to concentrate resources on the most profitable lines, OpenRisc is a disruptive

technology, enabling semiconductor companies to develop chips for embedded markets like network devices, personal entertainment hardware, and niche industrial applications – without having to spend money on operating their own factories.

Much of the active development on OpenRisc comes from companies like Swedish design house ORSoC, which also sponsors the OpenRisc project

directly. Many other small companies make chips and boards based on OR1K, including AAC Microtec, which has had its product put into orbit. The fast development offered by open hardware also makes it great for larger companies playing in fast-moving markets: Samsung ships OpenRisc chips in the system-on-chips used in its digital TVs.

Any curious hacker or maker can experiment with FPGAs and OpenRisc. Delving into chip design enables you to grapple with all sorts of tasty problems involving Fused Multiple Accumulator (FMAC) arithmetic, bus design, and optimal register numbers. If you've ever programmed at a low level, and cursed the decisions made by chip designers at Intel, now is your chance to show the world a better way!

Anyone wanting to join in the fun will find many resources online, but also meetings and chances to learn the process of programming your own FPGA through the Open Source Hardware Users Group (OSHUG), which conducts meetings in and around London but also ventured north for last year's Open Source Hardware Camp at the Wuthering Bytes festival in Hebden Bridge.

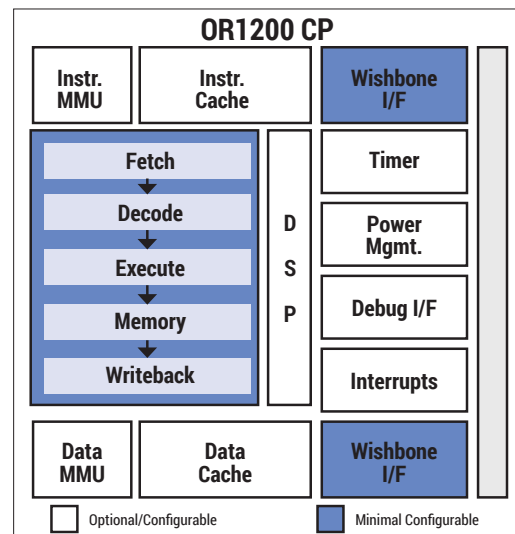
Open to all

Working with OpenCore designs is challenging but rewarding. "Inexperienced users should be warned that the OpenRisc processor is quite a difficult processor," warns Patrick Pelgrims of the Belgian De Nayer Instituut, in his tutorial on designing and implementing an OpenRisc-based embedded system. But we don't want that to put you off – the reference design is a good place to start, and as with learning programming through playing with existing, working code, so with hardware.

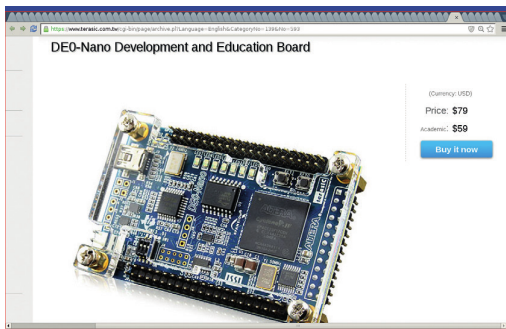
We asked Dr Bennett about the difficulties involved. He pointed out that it's "a relatively simple and well documented architecture. It has a pipeline (more difficult), but only a five-stage pipeline in the standard

"It is well within the grasp of a competent hobbyist. And of course modifying an existing design is always easier than designing one from scratch!"

Dr Jeremy Bennett.



Inside the OpenRisc 1200 CPU – configuration at the silicon level, with Free and Open Source Hardware.



You'll need an FPGA development board to get started which, while not cost-free, is orders of magnitude less expensive than building a CPU plant!

implementation (so not that difficult)." He summed it up as: "more complicated than some, but a lot less complicated than many. The bottom line is that processor design is not trivial. On the other hand it is well within the grasp of a competent hobbyist. And of course modifying an existing design is always easier than designing one from scratch!"

Chip Hack & getting involved

OSHUG runs an annual event called Chip Hack, which is a weekend of learning to create embedded hardware, building and making, and taking home your own OpenRISC SoC:

If you can't get to Chip Hack you can still give it a try yourself: you could use a browser-based emulator to explore OpenRISC (see boxout, above-left), but getting the toolchain installed on your PC to get started can be as simple as:

```
$ git clone git://openrisc.net/jonas/toolchain
```

```
$ cd toolchain
```

```
$ git submodule update --init
```

```
$ make -j3 PREFIX=~/.openrisc/toolchain
```

```
$ export PATH=PREFIX=~/.openrisc/toolchain/bin:$PATH
```

For an optimum `make -j` value, double your number of CPU cores, and add one. The destination directory can be anywhere you have permission to put it. After setting up your cross-compile environment and building a Linux kernel (see <http://openrisc.net/>

```
richard@rivendell:~/Downloads$ git clone git://openrisc.net/jonas/toolchain
Cloning into 'toolchain'...
remote: Counting objects: 93, done.
remote: Compressing objects: 100% (77/77), done.
remote: Total 93 (delta 52), reused 32 (delta 16)
Receiving objects: 100% (93/93), 13.83 KiB | 0 bytes/s, done.
Resolving deltas: 100% (52/52), done.
Checking connectivity... done
richard@rivendell:~/Downloads$ cd toolchain/
richard@rivendell:~/Downloads/toolchain$
BRANCH: master (f986c82)
$ git submodule update --init
Submodule 'binutils' (git://openrisc.net/jonas/binutils-svn) registered for path 'binut
ils'
Submodule 'gcc' (git://openrisc.net/jonas/gcc-svn) registered for path 'gcc'
Submodule 'linux' (git://openrisc.net/jonas/linux) registered for path 'linux'
Submodule 'uclibc' (git://openrisc.net/jonas/uclibc) registered for path 'uclibc'
Cloning into 'binutils'...
remote: Counting objects: 10268, done.
remote: Compressing objects: 100% (8585/8585), done.
remote: Total 10268 (delta 1298), reused 10261 (delta 1296)
Receiving objects: 100% (10268/10268), 23.04 MiB | 3.46 MiB/s, done.
Resolving deltas: 100% (1298/1298), done.
Checking connectivity... done
Submodule path 'binutils': checked out 'd4889c03191ec5cbe8cb2168d517977d9b72184'
Cloning into 'gcc'...
remote: Counting objects: 42573, done.
remote: Compressing objects: 100% (31484/31484), done.
Receiving objects: 0% (418/42573), 4.93 MiB | 1.55 MiB/s
```


[toolchain-build.html](#)) you can test-run your OpenRISC environment in a VM with:

```
or1ksim -f arch/openrisc/or1ksim.cfg vmlinux
```

Next, you'll need an FPGA development board. This year's Chip Hack event will use the DEO-Nano board, but there are plenty of others listed on the **OpenCores.org** website, including some recent developments. As noted earlier, you will need proprietary Quartus software from Altera installed to turn the Verilog HDL file into something that can be loaded onto the FPGA. Before that you'll need the Verilog file itself – the OpenRISC site has an OpenRISC Reference Platform System-on-Chip in the flavour you need.

Environmentally friendly

Low power consumption has always been an important selling point for RISC chips, enabling them to quietly conquer the embedded space in the 1990s, and thus be the big winners in the rise of the mobile device. Given the huge power consumption of data centres on a worldwide scale, it's no surprise to find OpenCore developers at UK-based Embecosm, which did much of the work on GCC and the GNU toolchain for ORSoC. Adapteva (developers of "a revolutionary many-core embedded computing platform for applications requiring ultra high floating-point performance with minimal power consumption"), is also doing work funded by the UK Technology Strategy Board (a UK government innovation agency), to optimise GCC for compiling binaries with a lower power draw.

As we go to press, the Chip Hack Cambridge event, providing an introduction to FPGA programming, is already sold out "but a key part of the idea is that he resources are open, so others can run the course themselves," Dr Bennett tells us. Get on the Chip Hack mailing list, and you should get early news of other events and training opportunities. 

You could build an entire OpenRISC toolchain yourself, but as the hard work's already been done, just grab it with Git and get on with the fun.

DIY chips on the web

- Introduction to FPGA programming event. <http://chiphack.org>
- Good beginners' introduction. www.rte.se/blogg/modesty-corex/openrisc-1200-soft-processor
- Julius Baxter's Masters Thesis on the OpenRISC Project. <http://juliusbaxter.net>
- Paper covering all the chips from OpenSPARC to the European Space Agency's LEON project: <http://ur1.ca/gytic>
- jor1k – OpenRisc 1000 in your browser. <http://s-macke.github.com/jor1k>
- Open Source Hardware User Group <http://oshug.org>
Also, there's a supportive community on the #opencores channel on freenode IRC.

"Low power consumption has always been an important selling point for RISC chips."