EMBECOSM™

Open Source Software Meets Open Source Hardware: The OpenRISC 1000

Jeremy Bennett, Embecosm

Abstract

This paper presents the OpenRISC 1200, an open source implementation of the OpenRISC 1000 architecture, verified using open source tools. The OpenRISC 1000 is supported by a modern GNU tool chain and is capable of running Linux as well as many real-time operating systems.

1 Introducing the OpenRISC 1000 Architecture

The OpenRISC 1000 architecture defines a family of 32 and 64-bit RISC processors with a Harvard architecture [9]. The instruction set architecture (ISA) is similar to that of MIPS or DLX, offering 32 general purpose registers. The processor offers WishBone bus interfaces for instruction and memory access with IEEE 1149.1 JTAG as a debugging interface. Memory management units (MMU) and caches may optionally be included.

The core instruction set features the common arithmetic/logic and control flow instructions. Optional additional instructions allow for hardware multiply/divide, additional logical instructions, floating point and vector operations. The ALU is a 4/5 stage pipeline, similar to that in early MIPS designs.

A hardware debug unit provides access to all registers and main memory and allows external stall/unstall and reset of the processor via JTAG. This interface can be used to provide software debug access via the GDB remote serial protocol.

One particularly useful feature is the parameterized NOP instruction, **`l.nop`**. The 16-bit immediate operand is ignored by hardware, but can be used by simulators to provide useful side-effects, such as host I/O and tracing.

The design is completely open source, licensed under the *GNU Lesser General Public License* (LGPL), this means it can be included as an IP block in larger designs, without requiring that the rest of the design be open source. Although there have been ASIC implementations, the majority of uses are with f*ield programmable gate arrays* (FPGAs).

The OpenRISC 1200 (OR1200) was the first implementation to follow the OpenRISC 1000 architecture. It is a 32-bit implementation incorporating optional MMUs and caches, a tick timer, programmable interrupt controller (PIC) and power management. The implementation is approximately 32,000 lines of Verilog.

The overall design of the OR1200 is shown in Figure 1.

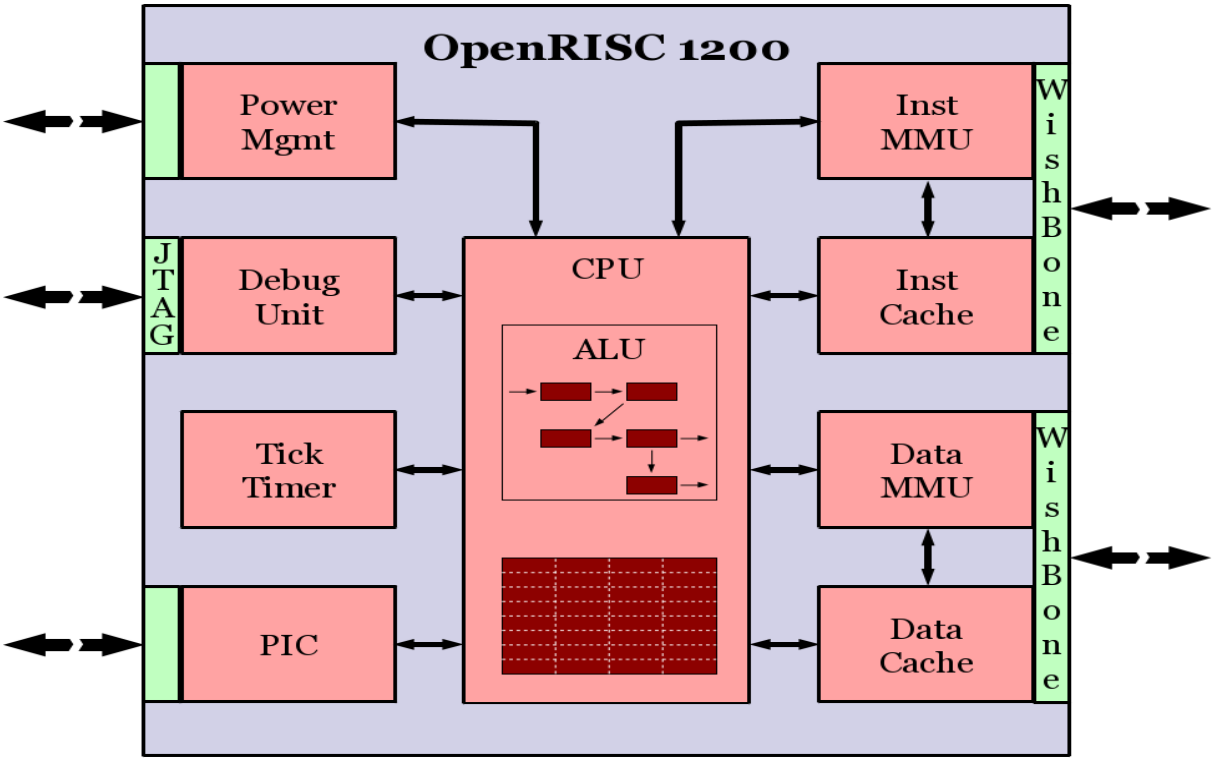

Figure 1: Overall design of the OpenRISC 1200.

1.1 History of the OpenRISC 1000

The OpenRISC 1000 was conceived in 1999 by Damjan Lampret, then at Flextronics, with the earliest public report in 2001 [7]. The objective was to use an open source approach to silicon intellectual property (IP) development, which at that time was almost exclusively closed source. The concept was wider than just the OpenRISC processor, and a website for open source IP of all types, opencores.org, was set up to host contributions.

Modern chip design looks (superficially) very like software design. Chip specifications are written using formal languages (typically Verilog or VHDL), which look like programming languages. They may be compiled for execution (as simulations of the chip), or compiled into low level logic gates, and ultimately silicon layout masks. As with software, it is amenable to open source development, particularly in a FPGA environment, where the marginal cost of change is zero.

For the first eight years, the development of opencores.org was supported by Flextronics, with most contributors being coming from their European development teams. The result included a reference ASIC implementation with approximately 150,000 gates and 17 memory blocks.

Since 2007, opencores.org has been run by ORSoC AB, a Swedish hardware design house. Under their stewardship the website has grown to around 120,000 registered users (at the time or writing). Discussions are under way now to set up a completely independent foundation, with the community now mature enough to take control of its own destiny.

The OpenRISC processor has been adopted in a number of commercial applications. Beyond Semiconductor is a design house, supplying commercially hardened derivatives of the OpenRISC processor. Jennic (now part of NXP) was an early

adopter of the Beyond Semiconductor designs for their Zigbee chips. Cadence use OpenRISC as a reference architecture to demonstrate their various EDA design flows.

2 Software Models

All silicon IP relies on modeling to prove the design before fabrication, even when the design is intended for FPGAs (as is the case with OpenRISC). A number of open source models are available for the OpenRISC 1000 architecture. It is worth noting that the main language for hardware modeling is now SystemC, itself an open source template library for C++.

2.1 Reference Golden Model

Or1ksim, the OpenRISC architectural simulator, is the golden reference for the OpenRISC 1000 architecture [2]. Originally written in C, it now also offers a SystemC TLM 2.0 interface. It is a traditional interpreting *instruction set simulator* (ISS), which typically executes at 2-5 MIPS on a modern workstation. It is licensed under the *GNU General Public License* (GPL).

Or1ksim also includes models of common peripherals components (UART, keyboard, VGA, Ethernet etc). It can be used standalone, as a GNU Debugger target (of which more later) or as a library within other programs. The modeling of Ethernet includes a TUN/TAP interface, meaning the model can appear on a LAN as an actual device.

Or1ksim has comprehensive tracing facilities available, which are available not only through configuration, but by external signal (SIGUSR1) or within programs using **`l.nop`**. This last is particularly useful, since it allows applications to turn trace on and off, allowing short traces of trouble spots to be collected.

2.1.1 Verifying the Golden Reference Model

A golden reference model must be thoroughly verified, and this verification is in turn is open source. Verification makes use of DejaGnu [8], a POSIX 1003.3 compliant test framework. The tests are either native C programs which link to the library form of Or1ksim (to test the library interface) or C/assembler programs compiled with the OR1200 tool chain to exercise the functionality of the ISS.

A total of 2,275 tests of the ISS functionality and 264 tests of the library interface are included in Or1ksim at the time of writing (release 0.5.1rc1). These test the functionality of both the core ISS and peripherals. For floating point verification we make use of John Hauser's *TestFloat* software [5].

Although the Or1ksim test suite has expanded considerably in the past few years (it was originally just 20 tests), no effort has yet been put into ensuring all target functionality is exercised. Furthermore no coverage metrics have been defined, nor coverage measured, so no assessment can be made of the quality of the coverage.

Thus verification of Or1ksim must be regarded as a work in progress.

2.2 Event Driven Simulation of the OR1200 using Icarus Verilog

Icarus Verilog [6] is an open source event driven simulator after the fashion of Verilog XL (the original commercial simulator). It runs between 20 and 50 times slower than modern commercial simulators such as Synopsys VCS, Cadence NC or

Mentor Graphics ModelSim. It is capable of running the OpenRISC 1200 Verilog at 1.4kHz on a 2GHz Core2 Duo E2180.

Being derived from the actual Verilog implementation, the simulation model can be used to check the design is correct before fabrication or synthesis for an FPGA.

Originally the hardware test bench comprised just 13 target programs, which were compiled and executed using the simulated design. This was completely inadequate. The tests were not exhaustive, there were no coverage metrics and the testing was not consistent with that of Or1ksim.

More recently, for his MSc dissertation [1], Waqas Ahmed implemented an OVM testing regime for the OR1200. Using Or1ksim as golden reference he aimed to verify against 5 criteria, generating appropriate coverage metrics.

1. Does the PC update correctly?
2. Does the status register update correctly?
3. Do exceptions save context correctly?
4. Is data stored to the correct memory address?
5. Are results stored correctly in registers?

Although Ahmed used a commercial simulator for his work, he was able to make use of the SystemC interface to Or1ksim to implement a DPI SystemVerilog wrapper. The resulting test bench allowed comparative testing of Or1ksim against the Verilog implementation.

Constrained random test generation was used to create a set of tests to maximize coverage. Testing uncovered numerous errors where the Verilog and Or1ksim disagreed, which fell into three categories.

1. Discrepancies due to ambiguities in the architectural definition. An example being the handling of unaligned addresses by **`l.jr`** and **`l.jalr`**.
2. Instructions incorrectly implemented or missing in the Verilog.. Examples being **`l.addic`** and **`l.lws`**.
3. Instructions incorrectly implemented or missing in Or1ksim. Examples being **`l.ror`**, **`l.rori`** and **`l.macrc`**, although these are all optional instructions. However they are implemented in the OR1200 Verilog.

In total 20 instructions had errors of some sort. This made for limitations in the coverage that could be achieved, since Ahmed did not have the opportunity to fix the Verilog during the period of this project. However he was able to show that for many instruction set coverage criteria, he had achieved as full coverage as possible, while for others he had achieved significant coverage. There remain however a set of coverage criteria with 0% result, since all instructions in these cases had errors.

As a result of this work, the architectural specification has been updated, Or1ksim has been fixed, and changes to the Verilog are in progress.

2.3 Cycle Accurate Modeling of the OpenRISC 1000

Event driven simulation is highly detailed, allowing the engineer to probe not just clocked signal logic levels, but to explore signals which are undefined, or undriven and to model behavior of the logic between clock edges.

However the downside is low performance (1.4kHz compared to several MHz for the architectural simulator).

For many purposes, such as low-level software development or compiler optimization, this level of detail is not necessary. However the architectural simulator is also not suitable. It offers only functional simulation of the instructions, and cannot see the details of, for example, pipeline operation.

The solution is to use a cycle accurate model of the design. This provides simulation of just the logic levels at clock edges. It is faster than event driven simulation, but more accurate than the architectural simulator.

For OpenRISC, the open source tool, *Verilator* [10], is used to generate a cycle accurate model from the Verilog. The performance (130kHz on a 2GHz Core2 Duo E2180) is adequate for low-level software development and testing. Indeed it is possible to run the entire GNU tool chain regression suite using such a model.

3 Open Source Tool Chain Support for the OpenRISC 1000

The OpenRISC 1000 has a full GNU tool chain implementation, based on binutils 2.20.1, gcc 4.5.1 and gdb 7.2. Both C and C++ are supported. There is a long term objective to bring the tool chain into the GNU mainline, something that has already been achieved for binutils.

At present only static libraries and linking are supported. The OpenRISC 1000 ABI did not have a definition for Position Independent Code (PIC). However a new project aims to remedy that omission, providing PIC support, and with it shared object libraries and dynamic linking.

3.1 Library Support

For bare metal operation, version 1.18.0 of the *newlib* library is implemented. This is a minimal implementation, offering just console or UART I/O. This was originally adopted to support GNU tool chain regression testing.

However an objective of the OpenRISC project has always been to support Linux. For this, the *uClibc* library has been implemented, providing full C/C++ library compliance for Linux applications.

There are thus two variants of the OpenRISC tool chain. The **`or32-elf`** tool chain is for bare metal applications using *newlib*. the **`or32-linux`** tool chain is for Linux applications using *uClibc*.

3.2 Software Debugging

Software debugging is through the GNU Debugger (GDB). A range of remote serial protocol servers are implemented, allowing debug through the JTAG [4] interface and the hardware debug unit. This interface is supported on all simulation models, as well as the hardware interface itself.

The debug unit includes optional support for hardware breakpoints and watchpoints. At present these are implemented in Or1ksim, but not available in hardware.

A notable omission is lack of any hardware trace support.

For Linux applications, direct debugging of the processor is not appropriate. Instead the *gdbserver* application is implemented, allowing GDB to connect to applications running in user space, making use of the Linux *ptrace* interface.

There is also an experimental implementation of KDB and KGDB for kernel debug, but at the time of writing, this has not been incorporated into the standard Linux distribution.

3.3 Tool Chain Testing

The GNU tool chains are supplied with very large regression test suites. During development of the tools these are run using Or1ksim as the target. At the time of writing, binutils and GCC are robustly passing regression for both newlib and uClibc targets. There are a handful of failures in C++ library regression for uClibc, due to limitations in the uClibc thread handling. GDB has a more substantial body of failures, and represents something of a work in progress.

It is also feasible to run the regression tests against the Verilator cycle accurate model derived from the Verilog implementation. This leads to an interesting approach to hardware verification, in which the results of testing using the two different targets are compared. Any differences are likely to indicate a hardware error, or at least an error in the reference golden model specification [3].

4 Operating System Support

4.1 RTOS

Many modern RTOS are open source, or at least have open source variants, and a wide range of these have been implemented on OpenRISC over the years. Examples include RTEMS, eCOS and FreeRTOS.

4.2 Linux

Linux is supported through a kernel implementation, supporting a BusyBox interface, using *uClibc* as the application library. At the time of writing Linux kernel 2.6.39 is supported, and forms the basis of a submission for adoption into the mainline Linux kernel.

5 Summary

The OpenRISC 1000 architecture and its main implementation, the OpenRISC 1200 have grown into a powerful open source embedded processor system. The hardware is supported by a robust open source tool chain and range of operating systems, including Linux.

The OpenCores community welcomes new members. Find us at www.opencores.org, or via IRC on freenode.net, channel #opencores.

6 References

1 Waqas Ahmed. *Implementation and Verification of a CPU Subsystem for Multimode RF Transceivers*. MSc dissertation, Royal Institute of Technology (KTH). May 2010.

2 Jeremy Bennett. *Or1ksim User Guide*. www.opencores.org.

3 Jeremy Bennett. *Processor Verification using Open Source Tools and the GCC Regression Test Suite: A Case Study*. Presented to the Design Verification Club, Bristol, 20 September 2010.

4 Jeremy Bennett. *Using SystemC Processor Models with the GNU Debugger.* 22nd meeting of the European SystemC Users Group, Southampton 14 September 2010.

5 John Hauser. *TestFloat*. www.jhauser.us/arithmetic/TestFloat.html.

6 *Icarus Verilog*. www.icarus.com/eda/verilog.

7 John G Spooner. *Open-source credo moves to chip design.* CNET News, 27 March 2001.

8 Rob Savoye. *DejaGnu: The GNU Testing Framework*. Free Software Foundation, 2004. Available at www.gnu.org/software/dejagnu/manual.

9 *The OpenRISC 1000 project*. opencores.org/openrisc,overview.

10 *Verilator*. www.veripool.org/wiki/verilator.

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.

Licensing

A shorter version of this article appeared in the BCS Open Source Specialist Group Newsletter #1 in July 2011.

- For any reuse or distribution, you must make clear to others the license terms of this work;
- Any of these conditions can be waived if you get permission from the copyright holder, Embecosm; and
- Nothing in this license impairs or restricts the author's moral rights.