LISP Hardware revisited

16. March 2006, Hans Hübner

Breakout Group Proposal to the 3rd European Lisp Workshop

LISP is not a perfect match for today's computers. In order to make stock hardware execute LISP code, complex compilers have to be used that make standard computers emulate LIPSs' model of computation. Progress in execution speed of standard processors have made this approach feasible, but this approach also carries a lot of complexity which makes it harder for a developer to fully understand the system that she is working on.

This is not much of a problem if the LISP programmer is in a team of many, allowing her to leave understanding the gory details of compiler and hardware technology to the respective specialists – But if the LISP developer is an individual thriving to control all aspects of her computer-implemented invention, the complexity of the environment surrounding contemporary LISP implementations is indeed a problem. Additionally, current computer systems follow an execution model that is radically different from LISPs'. In order to fully grasp what is going on inside her program, the LISP developer must not only understand LISP itself, but also the traditional von Neumann execution model. Finally, LISP environments are too large to be deployed in restricted environments like embedded systems due to the fact that their computation model must be emulated by sequential programs.

It is in fact possible to directly execute LISP in hardware, and it is more a matter of historic circumstance that LISP hardware is not available in the stores today. Companies like Symbolics, LMI and TI have shown that specialized LISP machines can be actually built – Even though those systems really did not natively execute LISP, they were specially built to support LISP systems. The SCHEME-78 and SCHEME-79 chips natively interpreted SCHEME in a binary form and have shown that symbolic processing can be done directly, without the need for a von Neumann system emulating the behavior. Given symbolic hardware that executes LISP directly, the traditional sequential model is no longer a prerequisite to understanding how to program, and in fact the expressive power of symbolic processing will make it completely unnecessary for many developers to even try understanding sequential systems at all.

The history of LISP hardware is generally being seen as a failure, but it seems to be common understanding that the failure was not due to the technology being inferior to the competition. Quite to the contrary, many of those who have used the Symbolics Lisp Machine seem to feel that even current environments often don't even come close to what Genera had to offer – Let alone the fact that all current LISP environments run on systems like Windows or Unix, which are totally foreign to LISPs' inherent concepts and need quite some wizardry to be understood.

One key aspect of why LISP machines failed is the fact that, at the time, developing custom hardware was a very expensive process and not only required the development of logic functions, but also the development of chip-design tools in order to put the designed logic onto silicon. Advanced CAD tools were unavailable, design rule checking was a manual and error prone process and turnaround times from design to chip were high and very expensive. At the same time, integration levels were low, with one million transistor functions on one chip being state of the art.

LISP systems have always been off-mainstream, seeking their applications in "advanced" fields like artificial intelligence and higher-order mathematics. Standard applications like operating systems, tabular databases, word processing and networks have been dominated by software written in sequential languages like C, Pascal or Fortran. This is why the execution model of these languages dominated development of new hardware in the past decades. In fact, being able to run GCC has been one of the first fitness tests of a new CPU developed in the 1990ies. Unix has been the predominant operating system, and getting Unix to run has been one of the prime tasks scheduled early in the development cycle of a CPU.

Also, the fact that recursion and symbolic processing is harder to understand for someone who has been exposed to the more traditional, sequential way of building computers, may have been influential on the mainstream development towards computers systems optimized to execute sequential code.

Then again, times have changed. The advent of cheap reconfigurable hardware allows us to re-think the feasibility of building LISP machines. To build one, we will not have to invent new CAD tools. We will not have to invent the basic mechanisms needed to map Lisp to hardware. All that it takes today is some ancient papers describing the architecture of a hardware LISP interpreter, a free synthesis toolkit for a FPGA, an evaluation board and…. Lots of free time.

Breakout Group Agenda

This breakout group will be used to discuss how the goal, a LISP computer executing LISP on hardware circuits, can be achieved. A list of topics to discuss includes:

- Can a full CL implementation be implemented, or does one need to constrain the system to SCHEME?
- What gate count will be necessary, is a one-chip implementation for the CPU feasible?
- How can the task be partitioned? What is the project's time frame?
- How can standard IP cores be interfaced to the LISP system?

- Should the task start where SCHEME-79 finished, or have there been other developments that can be built upon?

- What would be the architecture for a multimedia-able LISP machine?

Assorted links

http://opencores.org/	Free open source IP cores and chip design
http://repository.readscheme.org/ftp/papers/ai-lab-pubs/AIM-559.pdf	The Scheme-79 CPU
http://xilinx.com/	Producer of reconfigurable logic chips
http://altera.com/	Another producer of reconfigurable logic chips
http://shop.trenz-electronic.de/catalog/default.php?cPath=1	Ready-to-go FPGA development kits
http://members.optushome.com.au/jekent/FPGA.htm	John's FPGA page