

# Re: How to develop a random number generation device

---

*Source:* <http://sci.tech-archive.net/Archive/sci.electronics.design/2007-09/msg03265.html>

---

- *From:* David Brown <[david.brown@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:david.brown@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx)>
  - *Date:* Mon, 17 Sep 2007 23:17:33 +0200
- 

John Larkin wrote:

On Mon, 17 Sep 2007 14:54:38 -0500, Vladimir Vassilevsky  
<[antispam\\_bogus@xxxxxxxxxxx](mailto:antispam_bogus@xxxxxxxxxxx)> wrote:

John Larkin wrote:

My point is that large numbers of CPU cores \*will\* become common and cheap, and we need a new type of OS to take advantage of this new reality. Done right, it could be simple and astoundingly secure and reliable.

Dear John,

Try developing a perfect OS of your own. I did. That was a very enlightening experience of why certain things have to be done by the certain ways. Particular questions welcome.

I did write three RTOS's, one for the 6800, one for the PDP-11, one for the LSI-11. As far as I know, they were perfect, in that they ran damned fast and had no bugs. The 6800 version included a token ring LAN thing, which I invented independently in about 1974.

Well, I remember 64-bit static rams, and 256-bit DRAMS. I can't see

## Re: How to develop a random number generation device

any reason we couldn't have 256 or 1024 cpu's on a chip,  
especially if  
a lot of them are simple integer RISC machines.

1024 CPUs = 1048576 software interfaces and a hell of the bus arbitration.

No worse a software interface than if each process was running on a single shared CPU; much less, in fact, since irrelevant interrupts, swapping, and context switches aren't going on. Each process absolutely owns a CPU and only interacts with other processes when *\*it\** needs to, probably through shared memory and semaphores.

A shared memory interface for 1024 cpus? That's going to be absolutely vast, or have terrible latency.

I still don't understand why you think that interrupts or context switches are a reliability issue – processors don't have problems with them.

And I'd love to hear you explain to customers that while their web server has a load average of a couple of percent, they need to buy a second processor chip just to run an extra cron job. A single cpu per process will *\*never\** be realistic.

As far as bus arbitration goes, they all just share a central cache on the chip, with a single bus going out to dram. Cache coherence becomes trivial.

"Just share a central cache?" It might sound easy to you, but I suspect it would be *\*slightly\** more challenging to implement.

I'd be happy to waste a little silicon if I could have an OS that doesn't crash and that doesn't go to sleep for seconds at a time for no obvious reason.

The weak link is a developer. It is obviously more difficult to develop multicore stuff; hence it is a higher probability of flaws.

Putting a few hundred RISC cores on a chip, connecting to a central cache, is easy. You only have to get it right once. In our world, incredibly complex hardware just works, and modestly complex software is usually a bag of worms. Clearly we need the hardware to help the

## Re: How to develop a random number generation device

software.

You are too used to solid, reliable, \*simple\* cores like the cpu32. Complex hardware is like complex software – it \*is\* complex software, written in design languages then "compiled" to silicon. Like software, big and complex hardware has bugs.

Multiple cores gives absolutely no benefits in terms of reliability or stability – indeed, it opens all sorts of possibilities for hard-to-debug race conditions.

Especially if you remember about the 50–page silicon erratas for pretty much any modern CPU.

Intel, maybe. Are any of the RISC machines that bad? But my PC doesn't have hardware problems, it has software problems.

Yes, many RISC machines have substantial errata. The more complex you make the design, the more bugs you get.

What you seem to be missing is that although the cores on your 1K cpu chip are simple (and can therefore be expected to be reliable, if designed well), they don't exist alone. If you want them to support general purpose computing tasks, rather than a massive SIMD system, then you have a huge infrastructure around them to feed them with instruction streams and data, and enormous complications trying to keep memory consistent.

They don't if you insist on running a copy of a bloated OS on each. A system designed, from scratch, to run on a pool of cheap CPUs could be incredibly reliable.

What do you think in particular would be better for a typical desktop applications?

Oh, 256 CPUs and, say, 32 FPUs should be plenty.

My desktop machine might well run more than 256 processes. How does that fit in your device? But most of the time, there are only 2 or 3 processes doing much work – often there will be 1 process which should run as fast as possible, as single–thread performance is the main bottleneck for desktop cpus.

Re: How to develop a random number generation device

It's gonna happen.

You have to listen to the screams of the SEL software developers...

Of course lots of software people won't like this. Well, they had their chance and blew it.

John