

Re: How to develop a random number generation device

Source: <http://sci.tech-archive.net/Archive/sci.electronics.design/2007-09/msg03768.html>

- *From:* JosephKK <joseph_barrett@xxxxxxxxxxxxxx>
 - *Date:* Thu, 20 Sep 2007 03:00:30 GMT
-

MooseFET kensmith@xxxxxxxx posted to sci.electronics.design:

On Sep 19, 7:01 am, John Larkin
<jjlar...@xx> wrote:

On Wed, 19 Sep 2007 05:04:34 -0700, Martin Brown

But in proper synchronous design, controlled by state machines, immensely complex stuff just works. It's sort of ironic that in a big logic design, 100K gates and maybe 100 state machines, everything happens all at once, every clock, across the entire chip, and it works. Whereas with software, there's only one PC, only one thing happens, at a single location, at a time, and usually nobody can predict the actual paths, or write truly reliable code.

4G of RAM * 8 Bits is a lot more bits than a 100K gates. You need to keep your sizes equal if you want to make a fair comparison.

On Sep 19, 7:01 am, John Larkin
<jjlar...@xx> wrote:

On Wed, 19 Sep 2007 05:04:34 -0700, Martin Brown

<|||newspam...@xxxxxxxxxxxxxxxxxxxxxxxx> wrote:

Re: How to develop a random number generation device

On 18 Sep, 17:12, John Larkin

<jjlar...@xx> wrote:

On Tue, 18 Sep 2007 07:05:25 -0700,
Martin Brown

<|||newspam...@xxxxxxxxxxxxxxxxxxxxxxxx>
wrote:

On Sep 18, 2:30 pm,
MooseFET
<kensm...@xxxxxxxx>
wrote:

On Sep 17,
7:55 pm,
John
Larkin<jjlar...@xx>
wrote:

[...]

Programmers
have
pretty
much
proven
that
they
cannot
write
bug-free
large
systems.

In every
other area,
humans
make
mistakes
and yet we
seem
surprised

Re: How to develop a random number generation device

that
programmers
do too.

In most other areas of
endeavour small tolerance
errors do not
so often lead to disaster.
Boolean logic is less
forgiving. And
fence

Software programming
hasn't really had the true
transition to a
hard engineering discipline
yet. There hasn't been
enough
standardisation

Compare a software system to an FPGA.
Both are complex, full of
state machines (implicit or explicit!), both
are usually
programmed in a heirarichal language (C++
or VHDL) that has a
library of available modules, but the FPGAs
rarely have bugs
that get to the field, whereas most software
rarely is ever
fully debugged.

I think that hardware engineers get a better grounding in
logical
design (although I haven't looked at modern CS syllabuses so
I may
be out of date).

Hardware can be spaghetti too, and can be buggy and nasty, if one
does asynchronous design. But in proper synchronous design,
controlled by state machines, immensely complex stuff just works.
It's sort of ironic that in a big logic design, 100K gates and

Re: How to develop a random number generation device

maybe 100 state machines, everything happens all at once, every clock, across the entire chip, and it works. Whereas with software, there's only one PC, only one thing happens, at a single location, at a time, and usually nobody can predict the actual paths, or write truly reliable code.

But it is mostly a cultural thing. Software houses view minimum time to market and first mover advantage to gain maximum market share as more important than correct functionality. And it seems they are right. Just look at Microsoft Windows vs IBMs OS/2 a triumph of superb marketing over technical excellence!

And I have bought my fair share of hardware that made it onto the market bugs and all too. My new fax machine caught fire. Early V90 modems that only half work etc.

So, computers should use more hardware and less software to manage resources. In fact, the "OS kernel" of my multiple-CPU chip could be entirely hardware. Should be, in fact.

You are treating the symptoms and not the disease. Strongly typed languages already exist that would make most of the classical errors of C/C++ programmers go away. Better tools would help in software development, but until the true cost of delivering faulty software is driven home the suits will always go for the quick buck.

Re: How to develop a random number generation device

No, I am making the true observation that complex digital logic designs are usually bug-free, simple software systems have a chance of being so, and complex software systems never are.

John

May i introduce you to a concept called cyclomatic complexity. The cyclomatic complexity of 100's of interacting state machines is on the order of 10^5 to 10^6 . A memory array of regular blocks of storage accessed by a regular decoder has a cyclomatic complexity of on the order of 10 to 10^2 . In the memory there is much self-similarity across several orders of magnitude in size.

.