

Re: relative complexity of hardware and software (was: pick 'n' place machines

multithreading bugs that exist in software, also exist in hardware. (Put it this way: I'm sure I could write a threading bug in hardware if I tried.)

Perhaps the differences have more to do with how concisely small functional blocks can be fully described (in software, even something as basic as a scrolling listbox is hard to fully specify concisely enough to be useful to a programmer), and with the number of layers of functional composition (>10 for even a simple desktop app, versus I imagine many fewer for a hardware logic system)?

I don't design hardware logic so I'm just guessing. (I do design software, so I'm all too familiar with the processes that lead to bugs on that side.)

I reckon its down to the ease of upgradeability. its **easy** to change a piece of software on a PC. type away for a few minutes and hey presto, its changed. IMO this has led to a dreadful culture of coders whereby they dont design, they just code, and are almost deliberately sloppy, because hey, its ISP right.

its much harder to change analogue electronics, so its more important to get it right first time. or at least close enough that {insert name of best tech here} can easily rework it. So designers think about transient states, startup, shutdown, fault conditions, emissions, immunity, temperature, humidity, PCB flexure etc **before** building anything.

oops, I think I just said many coders are lazy and poorly schooled. that would explain why so much code is crap. Revenge of the bell curve. I'm sure it has nothing whatsoever to do with the fact that its a lot cheaper to train a student to play with a PC than to be an engineer, but they pay similar fees...

programmable logic does indeed allow for ratshit programming. I think the trick is to not hire ratshit programmers.

look at the number of screw-ups intel makes, versus micro\$oft. The hardware guys are winning hands down.

Cheers
Terry

.