The Embedded Muse 30

Editor: Jack Ganssle (jack@ganssle.com) December 15, 1998

Engineering Insights

This issue's theme is contributed by Robert Herold, and is used with his kind permission. Thanks also to Don Jackson who brought this to my attention and who keyed it in.

ENGINEERING INSIGHTS: Making Life Easier
By Robert Herold

Usually during the course of a day, I encounter something that is bothersome or just not quite right. The radio in my car is too close to the cupholder, so whenever I put down my coffee, the station gets switched from Howard Stern to Rush Limbaugh. Loading a new bottle into the office water cooler inevitably spills a few drops on the carpet. The sandwich guy asks my views of mustard versus mayonnaise despite having heard the full details just seconds earlier.

Porting operating systems to different hardware architectures is no different. Every system from the Rockwell AIM-65 through the Power Computing PowerTower Pro dual 225 MHz 604e has quirks that must be dealt with. Perhaps someday we'll design a system that's perfect, fast, reliable, and easy to write software for. If whoever builds it needs some unsolicited advice, I have a list...

LOGIC DESIGN

* Put in a software-readable version number. For chips, put in a read-only register. For circuit boards, dedicate a few I/O bits somewhere and put in some pull-up/pull-down resistors to make a unique identification possible. For I/O devices, put it wherever it's convenient. Whenever a change is made, even one that doesn't affect the operation of the software, change the version. Someday you'll be glad you did, bit it for manufacturing test, field service, consumer recall, hardware debug, user interface, or geek fascination with frivolous detail.

* Avoid write-only bits in registers -- let software read back what was written earlier. Avoid bits that mean different things when read and written -- chew up a little decode space instead and put them in separate registers.

* Design registers that can have individual bits modified in a single operation. This really helps when there's more than one bus master mucking about the system, in that register updates don't need to be protected with a software semaphore lock. My favorite example is the 6522 VIA registers -- the most significant bit controls whether you're setting or clearing, and the rest of the bits have a one wherever a bit is to be modified and a zero wherever a bit is to remain unchanged.

* Document what you design, preferably in a way that someone besides you can understand. Describe not only what it is, but also what it's for. Anyone who's taken a tour through a data book for a modern VGA graphics controller can appreciate this. Better yet, include some real (that is, actually compiled, run, and debugged) source code that uses your design. Keep the documentation available even after you stop selling the part. Make the documentation easy to get -- and free.

* Avoid timing dependencies. Any synchronous protocol or continuous input data stream is an exception of course, but even there you can help by providing a bit of buffering in your hardware. Putting a 5-million transistor 225 MHz screamer into an interrupts-disabled spin loop waiting for a few microseconds to pass before writing to the floppy controller again due to some empirically observed resistance to being written faster is not the best use of available MIPS.

SYSTEM DESIGN

* Make physical RAM contiguous. It's just easier that way.

* For multiprocessor systems, it's nice to be able to direct I/O interrupts to different processors to distribute the load. Also, it's nice to be able to turn off all I/O interrupts to all processors in a single operation and to turn them back on later, while still allowing interprocessor interrupts.

* For CPUs, it would be nice to separate I/O interrupts from interprocessor interrupts; having a different pin, a different interrupt vector, and a different interrupt enable is ideal.

* Resources that are duplicated across different parts of the system should have support for synchronizing their state. For instance, the PowerPC has a time base register and a pin for enabling it. Without that pin (or without any means of actually controlling the pin), it's impossible to ensure the time base register has the same value on all processors.

* For debugging, provide a means of doing a nonmaskable interrupt to all processors, preferably from an optional switch, and also from a software-accessible register. If this is a recoverable interrupt in the CPU, all the better. This will make OS designers everywhere think of you fondly every time the system locks up and they can push a button and get to a kernel debugger.

* Noncoherent caches aren't a good idea. A simple OS may be able to deal with caches that aren't coherent with DMA accesses, but anything beyond DOS or the maces will have major conniptions.

* Plan for bugs. Leave a couple of I/O pins around to let you control a PAL later, so you can do a quick turn on your design to fix the bug in the new improved gizmo that replaces the no-longer-manufactured old gizmo.

* Let the OS designer have one bit of visible I/O somewhere. A place for a two-pin header where an LED can be hooked up will do. Alternatively, provide a place to hook up a logic analyzer.

* Expansion buses should be self configuring (for example, NuBus) or software configurable (for example, PCI). For an example of how not to do this, see ISA, SCSI, or IDE.

PHYSICAL DESIGN

* Make the case easy to open. See the Mac IIci for an example.

* Put upgradable stuff where you can get at it. For an example of how not to do this, try adding RAM to a Power Macintosh 8500.

* Use the circuit board silk-screen for user documentation. If you've made the mistake of requiring jumpers, describe the jumper options right on the board.

* If you must have jumpers, put an extra one or two on unused pins. Not everyone lives ten minutes from Fry's Electronics.

* Use the circuit board silk-screen to make it easier to debug or probe. Label the chips with real names instead of "U24." Then tag every tenth pin with its pin number. This helps anyone poking around with a scope probe, and besides, its cool to personalize what that black plastic behemoth is doing millions of times a second. While you're at it, label some test points for clocks and interesting logic lines.

* If you must use screws, use the same kind everywhere. If they must be different lengths, at least make them the same thread. Make sure your design holds together if any one given screw is mission, because that screw always falls off the desk and take a four-hour vacation somewhere.

* Make it quiet. Otherwise, you can't use it while your spouse is sleeping.

* Avoid sharp edges. I hate bleeding on my motherboard.

* Connectors should be keyed. If not, the WILL be plugged in backwards. Lay out pins in connectors so that when they're plugged in backwards, smoke doesn't appear.

* Why is it that whenever I plug in the little twisted wire power LED, it's always the wrong polarity? Label the pins, preferably with the wire color that they're supposed to take. Alternatively, given the vagaries of wire color selection by LED vendors, label the pins minus and plus, and I'll just assume that the black wire goes to minus.

* If it's different, use different connector. I'm sure many parallel printers have been plugged into the SCSI ports on the back of Macintoshes everywhere. And in ancient times, 9-pin monitor cables were plugged into serial ports.

* If it's different, make it obvious that it's different. Can you tell the difference between a 3.5-volt DIMM and a 5-volt DIMM? Why aren't they labeled? For that matter, why aren't SIMMs and DIMMs labeled in the first place? Not everyone knows how to read DRAM part numbers.

PEOPLE IN GLASS HOUSES SHOULDN'T THROW STONES

Hardware designers who read this article may react defensively and start writing their own list about operating system design. They're right to do so -- we're by no means perfect.

Thought for the Week

Two mathematicians were having dinner in a restaurant, arguing about the average mathematical knowledge of the American public. One mathematician claimed that this average was woefully inadequate, the other maintained that it was surprisingly high.

"I'll tell you what," said the cynic, "ask that waitress a simple math question. If she gets it right, I'll pick up dinner. If not, you do." He then excused himself to visit the men's room, and the other called the waitress over.

"When my friend comes back," he told her, "I'm going to ask you a question, and I want you to respond 'one third x cubed.' There's twenty bucks in it for you." She agreed.

The cynic returned from the bathroom and called the waitress over. "The food was wonderful, thank you," the mathematician started. Incidentally, do you know what the integral of x squared is?"

The waitress looked pensive; almost pained. She looked around the room, at her feet, made gurgling noises, and finally said, "Um, one third x cubed?"

So the cynic paid the check. The waitress wheeled around, walked a few paces away, looked back at the two men, and muttered under her breath, " ..plus a constant."

About The Embedded Muse

The Embedded Muse is an occasional newsletter sent via email by Jack Ganssle. Send complaints, comments, and contributions to him at jack@ganssle.com.

To subscribe, send a message to majordomo@ganssle.com, with the words “subscribe embedded *your-email-address*” in the body. To unsubscribe, change the message to “unsubscribe embedded *your-email-address*”.

The Embedded Muse is supported by The Ganssle Group, whose mission is to help embedded folks get better products to market faster. We offer seminars at your site offering hard-hitting ideas - and action - you can take now to ***improve firmware quality and decrease development time***. Contact us at info@ganssle.com for more information.