The Embedded Muse 84

Editor: Jack Ganssle (jack@ganssle.com) September 22, 2003

EDITOR: Jack Ganssle, jack@ganssle.com

CONTENTS:

Editor's Notes

After an embarrassingly long hiatus, the result of getting married last May, extended sailing, and work on two books, the Embedded Muse is back. My apologies for the silence, and thanks to all who wrote wondering what was up.

One of those books is The Embedded Systems Dictionary (CMP Books, ISBN 1-57820-120-9), co-authored with Michael Barr (Editor-in-Chief of Embedded Systems Programming magazine). It's now available at Amazon and all the usual suppliers. We were both rather surprised at the sheer number of terms used in this field! The book has some 4500 entries, and is targeted at developers, their managers, and administrative support folks.

Jim Haflinger wrote in response to Embedded Muse 83 about testing systems through the power-up cycle with a very cool idea: "A test I have used for power cycling a unit is to have the control line actually turn off the power. All you need is a small circuit that keeps the power on unless a signal is forced from the unit. That way if the unit doesn't power up correctly, it will stay in that state for you to check it out."

Software WILL Fail

I had the honor of giving the keynote speech at last week's Embedded Systems Conference in Boston. The talk covered software disasters and the lessons we can learn.

But while working on the presentation I wondered why we refuse to acknowledge that software is indeed prone to failure. Developers generally have a subconscious attitude that the software will work, and there will be no unexpected inputs or odd operating conditions. Witness the plethora of buffer overrun conditions that plague many OSes, email and browser programs.

Software is funny stuff. Software engineering even funnier. No other branch of engineering is so intolerant of failure. We can beef up a strut to handle unexpected loads, or use a bigger conductor to handle possible electrical surges. But a single software error brings down the entire application. There are no fuses or weak links we can install to protect a system under stress.

Or are there?

A watchdog timer is a hardware kick-start in case the software crashes. Like a fuse it's extra hardware we add to the system (unless it's part of the processor's silicon), which sits outside the app, ready to intervene if the worst happens.

WDTs are a well-known and essential ingredient for reliable embedded systems. Yet an astonishing number of systems don't have a WDT. The Clementine spacecraft was lost when the code crashed and dumped all of the satellite's fuel. The team didn't "have time" to write code to use the WDT hardware that existed in the system.

Exception handlers are another tool we can and must use to capture unexpected problems. Yet unimplemented handlers are a common theme in the lore of disaster stories. If we've bothered to write the handler, in many, many cases that code is poorly tested. The system throws an exception, and the minimally tested handler, meant to save the device, causes a crash.

Memory management units are one of the best defenses against crashes. Run each task in its own partition; the MMU hardware traps errant accesses. The system can process the error and recover... or at least leave debugging breadcrumbs behind. If your system doesn't have an MMU, we can program spare chip selects to tag code that wanders into unused code regions. Yet we don't.

We think of software as being inherently brittle. Maybe it's not; maybe the code is fragile only because we refuse to install these sorts of fuses and weak links, as is done in every other branch of engineering. While writing this a good friend called to discuss some C code, and he very casually mentioned that all of his apps rewrite the output ports every second or so in case ESD flips their state. His code always includes a stack monitor to flag excessive pushing.

Why aren't we all doing this?

There's two reasons: laziness/schedule pressure, and lack of knowledge about techniques we can use. The former is problematic and possibly not solvable. We can deal with the latter by inventing cool tricks, and sharing those tricks widely.

A few techniques:
- Add an MMU, as discussed
- Always build in an awesome WDT
- Capture every possible exception – and TEST each handler thoroughly
- Monitor stack growth
- Rewrite output ports often
- Use ASSERTS to throw exceptions in any case where an error can possibly occur.
- Apply sanity checks to all inputs, whether from a user or from hardware (like an A/D).

Can you think of others? How else can we build fault-tolerant software, code that is robust enough to survive hardware glitches and outright bugs buried in the firmware? Let me know and I'll pass them along to the readers of the Muse.

Jobs!

I've been extremely reluctant to pass along job leads from recruiters. Most of us prefer direct contact with the company doing hiring and find working through these intermediaries frustrating. But the market is so awful, with too many decent developers desperate for work that I can't think of a good reason to not pass along what sounds like good opportunities. Here are some job openings in the Minneapolis/St. Paul area. Let me know if you're looking for embedded people and I'll mention it in the Muse.

SW Program/Project Manager - Full life-cycle embedded software development in mature/formal environments (CMM, SEI, FDA, DOD, ISOetc). Additional experience in GUI and mission/safety critical environments and communications a plus. Should have 2+ years Management or Project Management experience. BS (CS/EE/CE) required; MS/PhD preferred.

Senior V & V Engineers - Requires ability to learn software-intensive systems functionality, performance and operational needs. This position requires knowledge of embedded software testing principles as well as web based business system testing principles. BSCS/EE or equivalent required, prefer advanced degree. A minimum 5 years of experience developing state of the art software controlled products. Object oriented analysis and design is required. Use and discipline in system decomposition down to subsystem level. Experience with both real-time, reactive systems and Internet business systems; writing system requirements, designing subsystem interface

documents, system design, requirements allocation; IS processes and procedures; web based testing techniques and automated test tools. Good communicator both written and oral.

Contact Ken Rogers, RWKCONSULT@aol.com, 702-345-2089.

Joke for the Week

Embedded Muse 73 defined the Ultimate Tool Kit: WD-40 and Duct Tape. If it doesn't move, and it's supposed to use the WD-40. If it moves, and it's not supposed to, use the Duct Tape.

Michael Kercsmar wrote about an upgrade to this kit. It includes:

For things that shouldn't move: SuperGlue

For things that should move: A hammer

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.