The Embedded Muse 15

Editor: Jack Ganssle (jack@ganssle.com) January 30, 1998

Magic and Delays

Miguel Flores contributed a "Dumb Mistakes" piece (following) about the perils of using "magic numbers", in this case in delay routines. It's scary how many times people get into trouble with delays, and also with constants tuned to "make the damn thing work". In his case both conditions combine to form a disaster.

If you've been in this business for a while you've surely seen the perils of delay routines. It's almost amusing, in a bittersweet way, to watch programmers wrestle with delays, tuning them to get the system working.

Magic numbers are all too often the shortcuts we use to dodge deep understanding of a problem. We tune a delay, or a constant, coming up with a bit of what is truly magic just to get it out the door.

If we, the people developing the system, don't have a deep understanding for the reasons behind EVERY DESIGN DECISION WE MAKE then we're surely invoking magic, just as much as the necromancer of old. Magic numbers are as effective as Tarot cards.

Remember the old adage: problems that magically go away have a habit of magically reappearing.

Understand before coding. Magic has no place in embedded systems.

More Dumb Mistakes

From Miguel Flores

Here is one I had to solve recently that has a good lesson. The boards I mentioned above have an ISA bus interface with the host PC. We have a message protocol for sending messages between the host and the board's microcontroller. The traffic across the ISA bus is controlled by a hardware register containing status bits for words coming and going, and a ready bit. The ready bit is supposed to indicate that the board is ready to do ISA bus data transfers and is tied to an I/O pin on the microcontroller. The board has an interrupt for data coming, and one for data going.

The host PC uses a Unix OS, and when the system is booted, a driver for our boards will perform a hardware discovery routine to learn what hardware is installed on the ISA bus. This routine consists of resetting the board, waiting a *magic* delay, then trying to send a message to the board over the ISA bus. In theory, the ISA status register on the board eliminates any timing or execution speed dependencies.

That's the theory. In practice, the I/O pins from the microcontroller are bi-directional. After reset the I/O pins are tri-stated and act as inputs. This includes the ISA ready bit I/O. It is pulled up to inactive so at reset it goes inactive (not ready for ISA I/O). The original developer (in fairness, under considerable schedule pressure) got this whole scheme to work with a little extra (read: magic) delay in the Unix driver after it resets the board. Fine, ship it.

So, next, we make a new version of one of these boards, and eliminate a 15 MHz clock circuit used to drive the microcontroller. Instead, we use an external 10 MHz clock which is used elsewhere on the board anyway. The microcontroller runs slower, but it is still fast enough to do its work. This should not affect the ISA bus messaging since it uses that ISA bus status register. Well, by now you can guess that the Unix driver could not find the slower board after reset.

Puzzled, I first supplied the board with a 15 MHz clock where the 10 MHz should go. This would make the board run like it used to when it worked. OK, binary search. Try 12.5 MHz. Works. Try 11.5 MHz. Works. Try 10.5 MHz. Fails. Try 10.6 MHz. Works. Oh no, a timing dependency. In comes the o-scope and some instrumented firmware and printed copies of I/O drivers.

To make a long story short, the board's ISA ready register was being set within the first couple dozen instructions after reset (about 50 microseconds), even though the firmware was not ready until 5 milliseconds later (what with interrupt vectors and all). The magic delay in the Unix driver was not long enough for the slower microcontroller clock speed, so, the first word on the ISA bus after reset goes into the bit bucket. The board runs, but Unix can't talk to it.

The assembly language start up routine for the firmware, copied from other projects, assumed it knew the I/O configuration of the microcontroller, and set the direction of the I/O pins without first initializing the data that would be driven on the output pins. This included the ISA ready bit in the wrong state. This explained the need for the magic Unix driver delay after reset.

The lesson here is to not make or impose any unnecessary assumptions about the hardware or software. The start up firmware has nothing to do with any of the microcontroller I/O's, so don't mess with them. And further, make sure the I/O data has

been initialized before driving the output pins. Good thing this equipment is not connected to any mechanical stuff. Finally, if you need magic to make your system work (like our magic delay in the Unix driver), then you don't fully understand how or why it works.

Thought for the Week

Write in C (Sing to the Beatle's tune "Let it Be")

When I find my code in tons of trouble,
Friends and colleagues come to me,
Speaking words of wisdom:
"Write in C."

As the deadline fast approaches,
And bugs are all that I can see,
Somewhere, someone whispers:
"Write in C."

Write in C, Write in C,
Write in C, oh, Write in C.
LOGO's dead and buried,
Write in C.

I used to write a lot of FORTRAN,
For science it worked flawlessly.
Try using it for graphics!
Write in C.

If you've just spent nearly 30 hours,
Debugging some assembly,
Soon you will be glad to
Write in C.

Write in C, Write in C,
Write in C, yeah, Write in C.
BASIC's not the answer.
Write in C.

Write in C, Write in C
Write in C, oh, Write in C.

Pascal won't quite cut it.
Write in C.

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.