The Embedded Muse 154

Editor: Jack Ganssle (jack@ganssle.com) January 28, 2008

EDITOR: Jack Ganssle, jack@ganssle.com

CONTENTS:

Editor's Notes

Did you know it IS possible to create accurate schedules? Or that most projects consume 50% of the development time in debug and test, and that it's not hard to slash that number drastically? Or that we know how to manage the quantitative relationship between complexity and bugs? Learn this and far more at my Better Firmware Faster class, presented at your facility. See http://www.ganssle.com/classes.htm .

For the first time, I'll be conducting a public version of this class in Denmark in April. For more information see http://www.ganssleindk.dk/ .

Are you in the Chicago or Denver area? I'll present a public version of the Better Firmware Faster class in Chicago and Denver, on April 23rd and 25th. Sign up before March 23 and receive a $50.00 discount. Registration and other info here: http://www.ganssle.com/classes.htm . You'll earn 0.7 Continuing Education Units, learn a lot, and have more than a little fun.

And I'll be teaching a public version of this class in London, UK, May 19. See http://www.ganssle.com/classes.htm .

Michael J. Pont is working on a new book entitled "Rapid Development of Reliable Embedded Systems," and is releasing draft chapters for free, in hopes of getting useful feedback. Check it out here: http://www.tte-systems.com/books.php .

Tools and Tips

The "insert symbol" process in Microsoft Word is slow and cumbersome, yet in technical docs we often need to do that a lot. Dave Kellogg sent along a simple tip that converts a two character sequence into a single mathematical character. It has screenshots that can't be included in this text-only newsletter, so see http://ganssle.com/misc/insertmathsymbols.pdf .

Nuggets From Recent Publications

One useful software engineering publications is Crosstalk, a freebie available from http://www.stsc.hill.af.mil/crosstalk/about.html (even the print version is free). Though most of the articles are dreary there are enough gems to make it worth reading.

In the December 2007 issue Capers Jones has an article called "Geriatric Issues of Aging Software." Pretty much everything he writes is worth reading. In this piece he lists some interesting data:

- Defect removal efficiency is about 85% for most organizations, and it shows no sign of getting better. About 5 bugs per function point (around 120 lines of C) get injected during development, or about 42 per KLOC. Fix 85% and you're still shipping with 6 bugs per KLOC. About a third of those will be serious enough to stop the program from running, or will create incorrect outputs.

- About 7 percent of bug fixes will inject another bug. That number grows significantly for complex functions.

- Around 5% of the modules contain around 50% of the system's bugs, though CMM level 3 and above organizations usually don't see this effect.

The December 17, 2007 copy of EE Times has two articles about engineering education in India. Apparently there just are not enough professors for the student body; most schools are understaffed by 30% since the profs are flocking to higher salaries in industry.

In 2006 Indian schools granted some 237,000 engineering BS degrees, compared with 74,200 in the US, 98,400 in Japan and a whopping 351,500 in China. US schools conferred 8,400 PhD degrees, eight times as many as in India and twice Japan's and China's.

IEEE Computer, another one of those magazines whose content is too-often soporific, devoted the October 2007 issue to embedded systems. One article does a good job explaining the challenges of designing systems for the automotive industry, and another investigates software-defined radios. Apparently software-defined radios consume more die space than a traditional radio, but once more than about four standards must be supported they offer a clear die-size advantage.

The Communications of the ACM's January, 2008 issue is their 50th anniversary. A piece titled "Digging CACM" extracts some historical information from their half-century of publication. In it Phillip Armour explains Grosch's Law, which said that the cost of a computer will rise as the square root of its power. At the time the IBM 3081 offered 10 MIPS and cost $3.7m. Today's Xbox hits 6,400 MIPS, so should cost $93m instead of $300. Moore's Law trumped Grosch's Law by a factor of 310,000. Today the rule of thumb is that the processor power rises with the square root of the number of transistors.

The January 2008 issue of IEEE Spectrum has an on-line calculator that shows the economics of the tech industry since 2001 (see http://spectrum.ieee.org/rndcalc). R&D spending has gone up by third over that period, sales (for the listed companies) shot up 40%, but employment increased a dismal 6%.

A War Story

I find stories from other peoples' experiences to be interesting, and even vital. Learning from others is much more efficient than making the same sorts of mistakes ourselves. Tamir Michael sent this one in:

This is a war story. It is a short story about a series of mistakes that could have led to a massive failure. It is about my smugness. I hope you have time to read on – whether you are a developer or a manager. The technical details (I tried to simplify them) are less important than the persistency used to find the failure and the lessons I learned from it. It is an important lesson, I think, which demonstrates the wonderful complexities that are sometimes involved in our work.

It all starts with a click. It sounded like a relay shifting, actually. Every time I pressed on the user interface button to display the error archive, I heard a click. I am using hardware that was utilized for another project with a different I/O setup so I ignored it, since all seemed to work normally. The man in charge once asked me for the reason for that click. I told him what I knew at the time: “I don’t know. Nothing, probably”. I was wrong.

Trying to finalize the product, months later, I was asked to add CAN logging to the program. No problem, I thought: all I had to do was to modify the logging functions. I did that, tested the CAN traffic to my satisfaction and was ready to finalize. But before checking-in everything back to SVN, I ran a general test, as I usually do, just to be sure. To my dismay, attempting to display the error archive generated not a single click but a series of clicks! I knew this phenomenon too well. That was the power supply relay going crazy, which meant my microcontroller was dead.

What happened? I don’t have a debugger, but I knew that I did nothing significant except of removing several functions that were not referred to by any other component. Could this then be the reason for the system failure?

I started running additional tests which left me even more puzzled. It seemed as if removing parts of the suspicious functions didn’t cause the microcontroller to crash, but removing all the contents did kill the microcontroller, even though the functions were not used in any way. I used a blackboard to summarize the possible causes for the system failure, but none of them seemed to be a plausible explanation to the behavior I experienced.

At this stage I turned to an automatically generated file that shows how system memory is organized and utilized by the microcontroller. It almost escaped my eye, but then I saw it: The culprit functions were mapped outside of the valid address space of the microcontroller! That would indeed make any attempt to use them or even their mere presence (or even their removal, as vital code would replace them) a potential disaster. However, the tool chain’s linker didn’t generate any warning or error to alert me of this dangerous situation.

Now that I knew what the problem was, all I had to do was to figure out how to command the linker to relocate the functions to a safe location.

The crash disappeared. But something else happened as well.

The click.

What can we learn from this?

First of all, this failure was built into the system for a long time without causing a system failure, and only a special chain of events exposed it. I have no idea how it crept into my program or why the functions were mapped in the wrong way.

This is not the first time I encounter this problem (the previous time involved another part of the program) but then I didn't find the cause and used a work-around. Essentially, I relied on a wrong theory without clear evidence to explain the erroneous behavior. That was a big mistake. On the other hand, I could not have known the actual cause without looking at the file that the linker generated. But since the linker didn't report any warnings or errors, I had no reason to do so.

For me, this incident was a reminder of hard-learned lessons that have managed to dim slightly lately. I got confident because everything seemed to work fine, but as too often occurs in the software business – it does not matter. It does not matter that it works. A program can contain deadly potential failures, sometimes so obscure that it takes the eye of an expert to detect them (this is not one of then, though), and to function normally most of the time. We just cannot be oblivious to detail.

ESD is 15? No, It's 20 Years Old

Embedded Systems Design is now 20 years old. They asked me to write a retrospective of those 20 years, which is here: http://embedded.com/columns/breakpoint/205203787 . But it was just yesterday, as I recall, that the magazine turned 15. At that time I wrote a personal history of the embedded landscape which follows. If anyone else, especially old-timers, want to share their perspectives, please do. If you love the history of this field as I do, and are ever in Mountain View, CA, be sure to visit the Computer History Museum (http://www.computerhistory.org).

If there's a dweeb gene I got a double dose. Computers fascinated me from early childhood. Yet in the 60s none of us had access to these horrendously expensive machines. Dweebs built ham radios and vacuum tube "hi-fi" gear instead. I had a tiny lab in the basement filled with surplus electronic equipment that was constantly being reconstituted into new devices. One "personal computer" existed: Heathkit's EC-1 (picture at http://www.heathkit-museum.com/computers/ec-1.shtml) analog computer was a $200 monster comprising just 9 vacuum tube op amps. Designed for simulating differential equations, users programmed it by wiring the amplifiers together using patchcords, resistors and capacitors. My lust for it remained unsated due the impossible cost.

At age 13 I was expected to spend Saturdays doing free janitorial duties for my dad's underfunded startup. The usual grumbling ceased when I discovered the other half of the deal: components swept from the engineering lab's floor were mine. Even better, those

engineers were slobs who dribbled parts faster than infants lose their rattles. Resistors, transistors, and even digital ICs started filling the parts bins at home, and far too many hours spent wiring all sorts of circuits taught me the nature of each device.

A lucky break at 16 landed a $1.60/hour job as an electronics technician. While Neal Armstrong frolicked across the Sea of Tranquility we built ground support equipment for Apollo and other programs. Exciting? You bet! Feeling flush with cash I paid $4 an hour for dial-up access to a Honeywell mainframe, my first exposure to any sort of computer. An ASR-33 Teletype at a friend's school gave us time-shared Fortran at 110 baud.

But surely there was a way to get my own computer! All attempts to build one failed, doomed by little money and less knowledge. So little that most of these designs accepted simplified Fortran as machine language since I'd never heard of assembly, let along machine, code. And any sort of memory was simply not available to a high school kid.

Senior year two of us hitchhiked from DC to Boston to visit an electronics surplus store. There I bought a 13,000 bit core memory box. No drivers, no electronics, just 26 planes of 512 bits arrayed in X-Y matrices.

Was it our long hair? Maybe the three unheeded warnings to get off the New Jersey Turnpike contributed. Gary and I found ourselves in a New Jersey jail cell, nailed for hitching. The police were surprised to discover the core instead of drugs in our backpacks. "What's this?" the chief growled. I timidly tried to convince him it was computer memory, the last thing he expected to find on a pair of hippie-freaks. They eventually let us go, me still clutching the core box. 37 years later it sits on my desk, a reminder of both long-lost youth and the breathless pace of technology. Picture here: http://www.ganssle.com/misc/core.htm .

I became a reluctant college student, reluctant only till I discovered the university's Univac 1108, a $10 million mainframe packing about as much power as one of today's ubiquitous calculators. Class attendance became a last option before exams, pushed to the bottom of the priority list after work and all night computing. As I learned the secrets of assembly language and operating system vulnerabilities the limits of the usual $50/semester account were easily transcended.

The 1108 ran at 1.3 MHz, had 768K (not megs) of core memory, and used 2 ton six foot long spinning drums each storing 45 Mb. Punched cards were the GUI of that age. The machine had a ridiculous instruction set lacking even a stack, an awkward 36-bit word, and the annoying habit of crashing more often than Windows 2.0, especially when final projects were due. But I loved that computer, and wasted absurd amounts of time developing tools and applications for it.

To stay abreast of the latest in electronics I wrangled a free subscription to EDN, probably offered due to considerable exaggeration about my title. Pathetically it was more fascinating to me than the Playboys my friends hoarded. In 1971 the magazine announced the creation of a computer on a chip. Chips never had more than a few hundred transistors – was this a hoax?

The "computer on a chip" was more marketing hype than reality, as the 4004 required a tremendous amount of external support circuitry. But it was a huge advance in computing. Since no dreamer wanted a 4-bitter as a home PC the only market it targeted was as the as yet unnamed and almost non-existent one of embedded systems. Intel invented not only the microprocessor, but for all intents and purposes the entire notion of cheap embedded systems. Yet according to The Microprocessor: A Biography (Michael S. Malone, 1995, Springer-Verlag, NY ISBN 0-387-94342-0) the company didn't at first understand the implications of this new innovation – they were afraid the yearly microprocessor market was a mere 2,000 chips!

Meanwhile I'd finally figured out the secrets of computers and built a 12-bit machine that actually worked. It used hundreds of TTL ICs – no microprocessor - wired on vectorboard, using brightly colored telephone cable soldered directly to each chip's pins. I couldn't afford sockets or wire-wrap wire. My Heathkit scope helped troubleshoot the logic, but since the computer was fully static it could even run at a fraction of a hertz. A simple voltmeter could follow bits flipping. Once the logic worked I cranked up the clock to 100 KHz or even 1 MHz, depending on how adventurous I felt. Memory was 768 words of semiconductor RAM (36 of Intel's 1101 256 bit static RAMs) and 256 words of 1702A EPROM.

$50 procured a World War II vintage ASR-15 teletypewriter capable of 50 baud communications (pictures of this type of machine are at http://www.railroad-signaling.com/tty/tty.html). Powered by a half-horse motor it must have weighed 200 pounds. The noise that beast made was indescribable. The neighbors in the apartment below sure learned to hate it. It spoke BAUDOT, sort of a 5-bit version of ASCII. Eventually I managed to get a bootloader working in EPROM (using a bit-banging UART), and then wrote a monitor program that was loaded from paper tape.

By the time the machine worked it was utterly obsolete. For in 1972, Intel released the 8008, the first 8 bit microprocessor. Like its predecessor, this part was at sea unless surrounded by lots of support circuitry. But the device, in an 18-pin package, offered what seemed like a vast 16k address space. It used three power supplies (+12, +5, and -9) and two twelve-volt clocks.

Panic set in at the company where I still worked as a technician. The 8008 made decent amounts of computing available for a reasonable price. It meant we could build a new kind of device, a machine that analyzed oil coatings on nylon, which needed far more

intelligence than possible via hardwired logic. The problem? None of the engineers know how to program.

A consultant saved the day, writing thousands of lines of PL/M and creating one of the very first production embedded systems. But he was expensive and the company broke. Work had started on another product, one that used infrared light to measure the amount of protein in wheat. Someone figured out that I knew assembly language, so was suddenly promoted to engineer. Classes, on those rare occasions I showed up, seemed utterly irrelevant compared to the thrill of messing with the machines.

The first of these grain analyzers used an 8008 with 4k of 1702A EPROMs. 4k doesn't sound like much, but required sixteen – sixteen! - 256 byte chips occupying an entire PCB. The CPU ran at a blistering 800 KHz, the device's max rate, and just about the same speed as the by now forgotten Univac 1108. Which was still at the same school noisily reminding me about upcoming finals.

Intel provided a rudimentary development system called the Intellec 8 (picture at http://online.sfsu.edu/~hl/c.Intellec8.html), which had a single hardware breakpoint set by front-panel switches. Its only I/O device was an ASR-33 TTY (picture at http://www.columbia.edu/acis/history/teletype.html), a 10 character per second unit incorporating a paper tape reader and punch. No disks, no mass storage, no nuthin'.

To boot the Intellec 8 we'd manually load a jump instruction into memory via front panel switches and press "run". A bootloader then read the editor in from paper tape. A crude very GUI-less editor let us enter our source code and correct mistyping (lots - the ASR-33 was hardly finger-friendly). The final edited source was punched to paper tape. These tools were all buggy; once it output my hours of typing completely reversed, printing the last character first and the first last. I started keeping a bottle of Mylanta at hand.

The cruddy tools gave us plenty of incentive to modularize, so even a small program comprised many separate source tapes.

The 10 CPS tape reader needed a couple of hours to load the assembler, which then read each source tape three times, once for each of its passes. Often – oh how often – the mechanical reader missed a zero or one (hanging chad isn't new) so one of the three reads wouldn't be identical to the others. Phase error – start all over! If fate smiled and all the stars of the Zodiac aligned the machine punched a binary relocatable tape.

Next load the linker (another hour or two), feed the binary tapes twice each, and if everything worked perfectly it spat out a final absolute binary image of our program.

Our 4k program probably consisted of 5000 source lines. A complete build took three days.

Needless to say we rarely reassembled. Each morning we'd load the current binary image and start debugging. Errors were fixed by changing the machine code of the offending instructions. Sometimes the fix was shorter than the code in memory so we'd drop new code in place of the old, padding unused bytes with NOPs. Longer patches went into unused RAM, accessed via a JMP plopped on top of the offending code. Careful notes on the listing logged each change so a later edit could make the source match the code. A day of debugging might result in quite a few patches, preserved by punching a tape of the current binary image. The next day we'd load that tape and continue debugging.

That 4k of code did very sophisticated floating point math, including noise reduction algorithms, least squares curve fits, and much more. It was a huge success that led to many more embedded products.

Intel's 1974 introduction of the 8080 wasn't much of a surprise. The shocker was an article in Popular Electronics about a home computer called the Altair 8800 (picture at http://wwwcsif.cs.ucdavis.edu/~csclub/museum/items/mits_altair_8800.html), an 8080 machine that sold for $400 in kit form. Not much memory came with it for that price, but since the processor alone sold for $400 engineers couldn't imagine how MITS pulled off this miracle. At the time we imagined they used reject parts, but learned later the company bought CPUs in high volume for $75 each.

The next generation of our product used an 8080, so we bought a pair of Altairs as development platforms. With two machines we could cannibalize boards to keep one working… most of the time. The Altair was designed to meet a price, evidenced by poor PCB layout and unreliable DRAM circuitry. Constant crashes and data loss were the norm. Tagemet became available about this time and prescriptions for the drug littered the lab benches.

After much complaining and more loss of time the boss relented and spent $20,000 (equivalent to $79k today) on Intel's new MDS-800. It was a general purpose 8080 computer that included an editor, assembler, linker, various other software tools, and the very first in-circuit emulator. A huge device, it became familiarly known in the industry as the "Blue Box". Mass storage consisted of two 8" floppy disks holding about 80Kb each.

Though our productivity soared development times grew longer. The 8080's 64k address space removed program size constraints, so customers and marketing started demanding ever-more features. The programs swelled to 16k, 32k and beyond. EPROM sizes grew in step so there were no barriers to code bloat. Except engineering time, but even then managers chanted the mantra "it's only a software change".

Everything was still written in assembly language. We experimented with Basic from a tiny outfit named Microsoft, and their later Fortran. Neither was adequate for real-time embedded work. No Cs existed for micros at the time. I created an Algol-like pseudo code implemented in assembly macros, but the MDS-800 took over an hour to translate a little 100-line program.

But assembly language was fun, if we could find programmers. There were never enough. For now the company was in a growth spurt and every product used micros. As is normal in times of turmoil the Peter Principle kicked in and I found myself in charge of all digital design and firmware. A growing staff barely kept pace with the demand for new products. We had our own offices and lab; those and our youth, the music, and odd hours divided us from the older analog folks. At least they seemed old; in retrospect none could have passed 30.

Processors were still very expensive. CPU chips cost hundreds of dollars. In 1975 MOS Technology introduced the 6502 at the astonishing price of $25. That sparked the Jobs/Wozniak whiz kids to develop the Apple computer. I bought one of these chips thinking to make a home machine. But for over a year home was a VW microbus, usually parked outside the office or on a rest stop on Route 95. After a Canadian vacation I reentered the US at a remote Maine town, hoping to take in the sights of the great North Woods. In an incident eerily echoing the NJ Turnpike bust customs agents, seeing the long hair and the microbus, stripped the van. They found the 6502 in the glove compartment. Here too, the officers were unaware of the pending microprocessor revolution, and were disbelieving about my story about a computer on a chip. Sure, kid. They didn't know what that 40 pin DIP was, but it sure looked like contraband.

We followed the evolution of technology, moving to the 8085 when it came out, and then, looking for much more horsepower to run the graphical displays our customers demanded, designed a system using AMD's 2901 bit-slice components. These were 4 bit elements strung together to create processors of arbitrary word length with custom instruction sets. A Signetics 8X300, a wacky DSP-like processor using 16 bit instructions but 8 bit data words, sequenced interactions between the bit-slice device and a Nova minicomputer.

We still flirted with minicomputers, searching on some products for more horsepower than possible with a micro. And some of these devices had to run Nova (picture at http://www.simulogics.com/museum/N1200_1.JPG) legacy code. This Data General 16 bitter offered decent performance for much less than the more popular and wonderfully orthogonal PDP-11.

The original Nova 1200 moved data through a single 4-bit ALU, using four cycles to do any arithmetic operation. It's hard to imagine in this day of free transistors but there was a time when hardware was expensive. Later models used full 16 bit data paths.

All had non-volatile core memory. Data General was very slow to provide ROMed boot code, so users were expected to enter the loader via the front panel switches. We regularly left the Nova's boot loader in a small section of core. My fingers are still callused from flipping those toggle switches tens of thousands of times, jamming the same 30 instructions into core whenever a program crashed so badly it overwrote the loader. At first all of the engineers got a kick out of starting up the Nova, flipping switches like the captain of some exotic space ship. Later we learned to hate that front panel. It's time to enter those instructions AGAIN! Acid indigestion gave way to a full ulcer.

We put in insane hours.100 hour weeks for 40 hours pay wasn't unusual, but there was little complaining. The technology was so fascinating! But by 1976 I was living on an old wooden sailboat and was rich – or so it seemed, with the amazing sum of $1500 in the bank. I quit to sail around the world.

And sank a year later. Back to the same job, but now I'd felt freedom and was itching for something more exciting. I resigned and started a consulting outfit with a good friend. For two years we built custom embedded systems for a variety of customers. A few stick out in memory – like the security system for the White House, which used over a hundred tightly-coupled 8-bit CPUs. When the contract ended I lost my White House pass the same day as Ollie North, though with much less fanfare.

We built a variety of deep ocean probes that measured O2, temperature, salinity, currents, and other parameters. These had to run for months to years on small batteries, so used RCA's 1802, at the time the only CMOS processor. It was a terrible chip lacking even a call instruction, but sipped so sparingly from Vcc that it hardly needed a power switch. Later we built a system that also used an 1802 to measure how fruit ripens while being shipped across oceans. That project was doubly rewarding when stevedores in Rotterdam dropped a shipping container on the device. The replacement job paid the bills for another month or two.

A 12-ton gauge that moved on railroad tracks as it measured the thickness of white-hot steel used a PDP-11 minicomputer interfaced to various 8-bit microprocessors. The plant's house-sized main motor reversed direction every few seconds to run the steel back and forth under rollers, tossing staggering amounts of RFI into the air. Poorly designed cabling could quite literally explode from coupled EMF. We learned all about shielding, differential transmission, and building smart software to ignore transients.

In those two years we starved, never having learned the art of properly estimating the cost of a job. Both of us agreed the friendship was more important than the company, so we sold the assets and each started our own outfits. That proved wise as we're still very close, and today our sons are best friends.

I'd had it with consulting. With every project the consultant more or less starts from scratch. A product, though – that seemed the ticket. Design it once and sell the same thing forever. But cash was scarce so I consulted during the day and wrote proprietary software at night.

The result was MTBASIC, a Basic compiler for the Z80 that supported multitasking. For a development platform I built a Z80 CP/M machine using a 40 character-wide TV monitor and a single floppy disk whose controller was a half-PCB of discrete logic rather than the fancy but expensive FDC chips of the time.

The compiler, targeted at embedded apps, was interactive like an interpreter yet produced native compiled code that could be ROMed. Compile times were nearly instantaneous and the generated code even faster. Built-in windowing and a host of other features drove the source to over 30k lines of assembly. But this compiler was the cutest code I ever wrote. Working out of the house I managed to sell some 10,000 copies for $30 each over the next few years.

In 1981, IBM introduced the PC. Using a 4.77 MHz 8088 and limited to 640k of RAM, this machine caused the entire world to take notice of the microprocessor industry. Though plenty of "personal computers" already existed, these were mostly CP/M based Z80 models that required enormous techie competence and patience. The one exception was the Apple][, but that only slowly made its way into the business world.

Though a very healthy and dynamic embedded industry existed, then as now it was mostly invisible to the average Joe. Few smart consumer products existed, so Joe's perception of computers was still the whirling tape drives in sci-fi programs or the ominous evil of HAL in the movie "2001- A Space Odyssey". But the IBM PC brought computers into the mainstream. Normal people could own and master these machines. Or so they're still telling us.

I bought an early PC. Unbelievably, floppies were optional. Most customers used cassette tapes. My two floppy model with 256Kb RAM cost $7000. I ported MTBASIC to the PC, recoding it in 8088 assembly, and found a willing market.

The top floor of the house was devoted entirely to offices now, the main level for storage. We moved into the basement. Neighbors complained about daily delivery trucks. Ceiling-high stacks of manuals and pallets of shipping boxes in the living room made entertaining challenging, or at least quirky.

Despite brisk sales, advertising ate all the profits. Still consulting, a government customer needed a battery-operated data collection system. National's NSC800 was ideal, but since

tools didn't exist for the CPU it seemed natural to make a simple little ICE, which worked surprisingly well.

Eventually that Eureka moment hit – why not sell these emulators? Since the NSC800 was so similar to the Z80 and 8085 it was a snap to expand the product line.

Some customers built astonishing control systems with MTBASIC. But the C language slowly gained acceptance in the embedded space and Microsoft's various flavors of Basic ate away at our non-embedded market. The PC killed off CP/M, obsoleting that version of the compiler. As the product's sales slipped, though, ICE revenues more than compensated.

The hardware design was simple, using only 17 ICs. The emulation processor was also the ICE control CPU. A bit banging UART complexified the software but saved a chip or two. But the firmware – oh my god, the firmware was a nightmare. And more fun than you can imagine. For on a breakpoint the CPU had to store the entire context of the executing program. Since I'd made the ridiculous decision to use no target system resources, when transitioning through a breakpoint the hardware had to swap in local ICE RAM and turn off user memory. State-saving PUSHes stashed information wherever the user's stack pointer had been – anywhere in the ICE's address space. Hardware stripped off some of the address bits to insure the data went into the emulator's RAM, but the need to minimize chip count meant it was usual for the writes to wipe out local variables used by the ICE. Reconstructing the data while correctly preserving the target context was quite a challenge. It was a cool design, though probably should have used more hardware to simplify the code.

We sold the units for $595 each. Though parts and labor only ran about $100, advertising and overhead burned cash at a scary rate. But the business grew. Forced out of the house by space needs, we rented a facility, the first of many as growth demanded ever-more square footage.

Over time I learned the basic law of the embedded tool market: keep prices high. Every application is truly unique so customer support is hugely expensive. Support costs are about the same for a $600 or $6000 tool. That's why today a simple BDM, which might use only a few dollars of parts, can cost thousands. And why Linux is free but plenty of outfits will happily drain your fortunes to help get it going. So we developed much more powerful units for prices up to $10k, keeping hardware and production costs around $1k.

Those were the glory days of emulators, when chip companies funded new ICE designs and customer demand was high. Our product line grew to include many 8 and 16 bit processors. The emulators themselves became hugely complicated, stuffed with boards crammed with very high-speed logic, memory, FPGAs and PLDs. 4 MHz processors accelerated to 8, then 12, and to 40 MHz or more. Even a few nanosecond delay imposed

by a single chip ate an unacceptable 20% of the bus cycle, so more exotic technology placed closer to the customer's target CPU socket became the norm. Logic design morphed into high-speed RF work. Maxwell's laws, at first only vaguely remembered from those oft-skipped college electromagnetics classes, were now our divine guidance. Firmware content skyrocketed. We used plenty of C, yet the emulators needed vastly more assembly than most products, as plenty of very low level bit twiddling was required.

Worn down by 70-hour weeks and the toll on my personal life I sold that company in 1996. Yet in many ways the tool business is the best of the embedded world. I met so many fascinating developers and poked deeply into their intriguing projects. Some used 8-bit processors to control fleets of aircraft while others had 32 bitters loafing along handling very slow inputs. A few ran for years off two AAs while others sucked from a 5 volt firehose. Applications varied from the absurd to the most noble imaginable.

After 35 years in this industry there are times I despair for its future. How can mere humans cope with million-line-plus programs? Is firmware quality an oxymoron? Will engineering jobs migrate at light speed around the world in pursuit of lowest possible costs?

I've learned, though, that the embedded revolution is one of the greatest outcomes of a troubled 20th century. No industry is untouched by our work. A flake of silicon reduces power plant emissions by orders of magnitudes, smart pumps irrigate sustenance farms in Nepal, and electronics in an automatic external defibrillator turn a Good Samaritan into a veritable cardiac surgeon.

In those Dilbert moments when the Mylanta isn't strong enough, if therapy or kicking the dog seem the only hope of getting through another day, take pride in your profession. We have profoundly changed the world, mostly for the better.

And that's a pretty darn good legacy.

Jobs!

Let me know if you're hiring firmware or embedded designers. No recruiters please, and I reserve the right to edit ads to fit the format and intents of this newsletter.

As a Staff software engineer for Scientific Atlanta, A Cisco Company, you will be involved in the design, development, and maintenance of 2D/3D Graphics Driver software for PowerTV and Linux-based settop boxes targeted to the North American market. You will also work closely with other engineers at the same and remote locations

including Cupertino (CA), Atlanta (GA), and Chennai.

Years of Industry Experience Required: 9 - 14 years of relevant experience (Driver Development in a Multitasking OS Environment)

Minimum Required Education: BS Degree (EE, CS, Math) or equivalent; MS/MBA Degree is preferred

- 2D/3D Graphics Drivers Development in Multitasking OS Environment/ 5 yrs. Min
- C software development/ 9 yrs. Min
- Excellent written and verbal communication skills required

Preferred Experience:
-Hardware accelerated OpenGL or Direct3D driver development in an embedded environment
-Hardware accelerated 2D Display driver development in an embedded environment
- Linux driver development in a small memory footprint environment
- Driver development using PowerTV OS
- C++ software development
- Experience with broadband and cable settop technology such as:
-- Conditional Access / Cryptography
-- Graphics Drivers
-- MPEG Audio, Video and Transport
-- Digital Video Recorder
-- Firewire 1394 Interface
-- IP Networking Protocol
-- DAVIC

Please feel free to apply to opening # **3298** at:
http://www.scientificatlanta.com/contactus/employment.htm

Associate Embedded Program Manager entry level PM. Are you looking to make the move from software development into program management? The Associate Program Manager is responsible for the overseeing Jumpstarts, Packaged Services, Work for Hire contracts and other simple to moderately complex hardware and/or software projects.
Position Responsibilities:

- Ensures customer satisfaction by managing the details of project schedule, project financials and deliverables and providing regular status reports to the customer.

- Develops and maintains staffing plans, may monitors project team members' progress and makes adjustments as required, controls changes, triages bugs and provides regular project status reports to BSQUARE management.
- Works closely with Finance to insure that Customer invoices are accurate and complete.
- Leads the project team to success by clearly communicating tasks, schedule, goals and deliverables.
- Is the primary interface between the customer and project team once the project is underway
- Helps win repeat business by making every practical effort to ensure that the customer's experience with BSQUARE is a positive one.
- Contributes to program management processes, tools and documentation.
- Presents a positive, confident and can do attitude in all communication with the customer.
- Works to attain a billable percentage of 50%.

Required Skills

- Requires a four-year degree (technical degree preferred), and a minimum of two years experience in a software or hardware engineering environment.
- Must possess an understanding of contract or product specifications.
- Requires leadership and communication skills (written and verbal).
- Requires a thorough understanding of the software development life cycle.
- Requires a thorough understanding of the project management life cycle.
- Ability to work onsite at a customer's location.

Required Experience

- Previous experience developing products in the Windows CE environment.
- Training or certification in project management discipline such as PMI or equivalent is highly desirable.

*We are currently not accepting agency submissions for this position.
Job Location Bellevue, WA, US.
http://hostedjobs.openhire.com/epostings/jobs/submit.cfm?fuseaction=careeropps&startflag=0&company_id=15686&version=1&CFID=11946622&CFTOKEN=12252300

Embedded Software Engineer (Platform) openings in Bellevue, WA and Vancouver, BC. Are you a C/C++ guru? Do you like working on a small team in a company that values work-life balance? Job Description: Adaptations for Windows Embedded and Windows Mobile to various platforms. These platforms include Windows Mobile devices, industrial handheld devices, web appliances/webpads, video camera systems, thin clients, personal computing companions, set top box systems and much more. Develop low level software support for new platforms running Windows CE and Windows Mobile.
Required Skills

- Experience writing embedded software or kernel level code, including drivers or hardware adaptations
- Expert C/C++ development skills
- Ability to work in small team environment
- Strong desire to work on commercial products

We think it would be awesome and a plus if you had:

- Windows CE and Windows Mobile Platform Builder experience
- MMC/SD/SDIO protocols and driver experience
- USB protocols and driver experience

Required Experience

- Relevant development experience.

Please apply online at:
http://hostedjobs.openhire.com/epostings/jobs/submit.cfm?fuseaction=careeropps&startflag=0&company_id=15686&version=1&CFID=11946622&CFTOKEN=12252300

Joke for the Week

.
To refresh your memory, the joke in the last issue was:

"To be or not to be... that is the question."

The answer is 0xff since:

0x2b | ~0x2b = 0xff

Many readers replied that the = sign should be ==.

Brian Trial wrote: “A better answer is -1: it scales to 16 and 32 bit machines.”

Colin Walls wrote: “But there is also the possibility:
0x2b ^ ~0x2b == 0

Who knows whether Elizabethan's thought in terms of OR exclusivity

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*”. ". BUT - please use YOUR email address in place of “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.