How to Become an Embedded Geek

Jack G. Ganssle
jack@ganssle.com

The Ganssle Group
PO Box 38346
Baltimore, MD 21231
(410) 496-3647
fax (647) 439-1454
I’m writing this while at anchor in Bermuda. There’s no net access aboard, so once or twice a week I head for an Internet Café ashore and dig through the email avalanche. For some inscrutable reason lately I’ve been drowning in emails from embedded system wannabees.

“Dear Jack: I lately learned Visual C++ and now want to start a career in firmware. But no one wants to hire me as I have no experience. What do I do? How to I learn about firmware?”

Perhaps my experience was atypical. I helped midwife the embedded business, learning while building products using the very first microprocessors. Like mastering the mysteries of the birds and the bees, I ran experiments, checked the results, talked to friends, and iterated till achieving some level of mastery.

Times are different now. Then, we were all amateurs. Today expectations are higher, competition for available positions brutal. Lately the staggering economy spawns few new job opportunities; those that surface are more often taken by experienced engineers than newbies.

So for people making the transition from college to real life I recommend hiding out for a year or two, if you can afford it. Consider getting an MS degree. If your BS is in Computer Science, take EE classes. Since the job market is so depressed it makes sense to optimize your skills to compete better when good times return.

And they will return. It seems we hit a bad economic patch in the beginning of each decade. Each one feels overwhelming, but they all pass. In the early 70s engineering collapsed with the loss of the Apollo program. Inflation and other woes caused a big contraction around 1980. The recession of the early 90s killed the elder Bush’s reelection bid. Our current problems, too, will pass, fading into an ugly but almost forgotten memory.

The Long and Winding Road

Too many of my email correspondents are looking for shortcuts. “How do I convince a potential boss to hire me?” “What book can I read to teach me firmware?” Sorry – there’s no easy path, no way to pass Go and collect $200. Though the bookstores have plenty of titles like “Learn to Program in 21 days”, I never expect to see an equivalent book for embedded systems.

Perhaps there is one easy way: get an EE degree. All other approaches will be harder. The degree gives you instant credibility in the marketplace, though a newly-minted sheepskin commands bottom-of-the rung salaries.

I wonder if this search for shortcuts is a quintessentially American characteristic. We’re so anxious to do things today! As a parent I’m constantly astonished to find how kids need to go
through so many experiences themselves. They can’t learn from my hard-won insights. There’s something intrinsic to humans about learning by doing.

Maybe you’re a master of C++, a whiz at programming in Windows using MFC and the wealth of tools we expect of any desktop computer. That’s a fantastic, valuable skill. It does not translate to an “in” into this field. C or C++ are base level skills for any firmware developer, but are merely a subset of the required expertise.

For there’s a huge gulf between the resource-rich environment of a desktop machine and a typical embedded system. It’s tough to generalize about firmware, because some projects run on 4 bit micros in 100 words of code, while others boot complete Windows or Linux operating systems. But I believe there are some skills shared by all of the best firmware developers.

First is the ability to work with limited resources. ROM and RAM may be very costly in high volume or low power products. Where on a desktop heaps and stacks are seemingly infinite, we firmware folks sometimes trim each to a razor’s edge, too often with catastrophic results. The Windows developer knows how to speed transcendental math using lookup tables, but his embedded counterpart looks askance at the sometimes staggering memory requirements. We use C subsets on minimal processors, where sometimes the pseudo-C is just a cut above assembly. Processors with poor stack architectures invariably spawn compilers that play complex games with automatic variables, games that can and will bite back when used by the unwary developer.

It’s not uncommon for time to be in short supply as well. There are limits to how fast a small processor can move data around; Moore’s Law does not bring the embedded developer a faster CPU every few months. When the system is performance-bound, embedded engineers re-design code, tune routines, and even at times change the hardware design.

So the accomplished firmware developer is a master of cramming more into less: more features into less memory, more performance into fewer CPU cycles. Assembly language always lurks, if only to understand what the compiler generates.

Embedded systems interact in complex and strange ways with the system’s peripherals. We’re not downloading drivers from some vendor’s web site, or relying on a vast infrastructure of OS support. Design a simple data acquisition system and odds are you’ll have to initiate A/D conversions, suck in data, and scale and normalize it. Working with a serial channel expect to write your own circular queue handlers.

We firmware folks are responsible for even the most basic of all functions. On most processors we must set up chip select pseudo-peripherals to determine the location and extent of all memory devices, as well as the number of wait states required.
Peculiar devices challenge even the most experience of developers. Pulse width modulated outputs aren’t uncommon yet defy many people’s understanding. Log amp compression circuits scale inputs in confusing ways. Even the straightforward switch behaves very strangely, it’s output bouncing for many milliseconds after it’s depressed.

Beyond simple I/O, though, the realities of our systems means we must be masters of interrupts and DMA transfers. Data arrives from a plethora of asynchronous sources and must get routed out as needed. Embedded systems people are expected to be competent at writing ISRs, and must understand how to create reentrant code.

Many firmware applications multitask, generally employing some sort of a real time OS. None of these offer the depth of support common to desktop systems; though some of the commercial OSes give a very complete framework for embedded work, they all look remarkably austere compared to Unix or Windows. Even today many embedded apps don’t and can’t use an RTOS, but the well-rounded developer must be versed at multitasking.

Debugging is especially difficult in the embedded world. If you’ve been spoiled by Microsoft’s debugger, expect culture shock when trying to peer into the workings of your firmware. In the best of cases there’s dam little visibility into the workings of our code. Sometimes we’re required to amend the hardware and software design just to make some sort of debugging possible – even be it so humble as wiggling pins and monitoring their states on an oscilloscope. If you can’t debug, you can’t make your stuff work, so plan to understand ICEs, BDMs, scopes and logic analyzers.

Embedded apps are – or should be – much more reliable than their desktop counterparts. A Word or Excel crash doesn’t compare in litigation possibilities to an avionics problem that kills hundreds. Less dramatic failures are just as serious. Having to stop every 20 miles to reboot your car controller is simply unacceptable. Few PC applications run for more than a few hours at a time, so memory leaks can often go undetected by the average user who turns the machine off each night. By contrast, an embedded product might have to run for years without cycling power.

To build reliable code we must understand and practice more extensive design than is common in other software projects. Failure Mode Analysis is required for some products. Extensive exception handling is a must. Code coverage tests are mandated for high-rel projects.

Become an expert C/C++ programmer. Gain competency at assembly language. Master all of the above. That gives you the basic skills needed for firmware development.
Changing Careers

An embedded occupation can be lots of fun, personally satisfying, a creative outlet, and reasonably financially rewarding. A lot of folks see these desirable traits after embarking on other vocations and search for ways to make a mid-life change. For most of these people, various family responsibilities make going back to college for an EE or CE degree impossible. In this case you have to design your own curriculum and advance your own career strategy.

First, read as much as you can. Here’s a few suggestions.

- *Bebop to the Boolean Boogie*, Clive Maxfield - A fun and interesting digital design book aimed at folks wanting to understand the hardware.

- *C Programming for Embedded Systems*, Kirk Zurell – A good introduction to working with small systems like the 6805, 6508 and PIC.

- *Embedded Systems Building Blocks*, Jean LaBrosse – a great intro into writing peripheral handlers. It also includes his firmware standard, a wonderful model for writing code in a consistent manner.

- *Embedded Systems Design*, Arnold Berger - A nice intro to the embedded world, with a focus on tools. Also has good hints on selecting processors.

- *An Embedded Software Primer*, David Simon – This is the best introductory book available. Extremely highly recommended.

- *Guidelines for the Use of the C Language in Vehicle Based Software*, by MISRA - This is a list of dos and don't dos for writing reliable C code. Not a book per se, but a hundred page list of rules. All will make you think.

- *High Speed Digital Design*, Howard Johnson and Martin Graham - The best book available about high-speed issues. The focus is entirely on hardware issues in fast systems.


- *Serial Port Complete*, Jan Axelson - A very complete reference to serial communications. Handling serial data is a basic skill for every developer.

Reviews of these books, plus many others, are at http://www.ganssle.com/bkreviews.htm.

Read embedded.com and chipcenter.com regularly. Scan every issue of Dr Dobb’s Journal, Embedded Systems Programming, and EE Times magazine. Clip and study ESP’s Beginner’s Corner, which is mirrored on embedded.com.

Read code, too, to see how experienced developers actually make things work. There’s plenty scattered around the web, on sites like 8051.com, chipcenter.com, microchip.com, and the like. I especially recommend reading the source to ecos (redhat.com), an open-source RTOS. Even better, read the source to LaBrosse’s UC/OS, available in the previously-mentioned book and on ucos-ii.com. Both of these operating systems are the very model of how we must write code – beautiful, well documented, clear, concise, and extensible.

Book knowledge is crucial but complement it with practical experience. Do projects. Build things. Make them work. Expect problems, but find solutions. Don’t abandon a project because it’s too hard or you’re confused. Most real development efforts are plagued by what initially appears as insurmountable problems, that the boss demands we overcome.

Spend some money on hardware for these projects. Consider, for example, Parallax’s Stamps In Class curriculum (www.stampsinclass.com). It’s a underpowered machine that offers enough performance for early learning.

Check out the inexpensive PIC and AVR boards at www.dontronics.com, and follow his newbie advice at http://www.dontronics.com/auto.html.

Move from that to a C-based system, like Rabbit’s evaluation boards (rabbitseminc.com) or an x86 board (svs-embedded.de/ssv/pc104/p169.htm). Another fun board is the 68HC11EVM (Motorola.com).

Start with simple projects, like LED blinkers, but keep challenging yourself. Add a timer ISR to manage the blink rate. Read real-world data from an A/D, process it, and display the results in an LCD. Send data back to a PC over a serial link. Build a command handler so the PC can issue various instructions back to your embedded system.

You’ll be on the way to mastery when the programs become large, not from lousy implementations, but due to the demanding nature of the project. I figure that a 1000 line project will teach a lot, but by the time the code reaches 5-10,000 lines of code it starts to resemble a simple but real-world app.
Conclusion

Without an appropriate degree, expect to work for a time as an intern or apprentice. Your salary will drop till you can acquire and demonstrate your competence. I suspect few people can avoid this painful reality.

Don’t be afraid to ask lots of questions… and be determined to move ahead.

Don’t be afraid to advertise your career dreams; let your current boss know you want to get into the embedded side of the business (if there is one). These days few companies are hiring, so it’s easier to make a lateral move from within the organization.

This sounds like a huge amount of work, and it is. If it’s too much, maybe you’re not cut out for the embedded industry. I suspect that most great developers succeed because they love doing the work. Indeed, various salary surveys show that, for engineers, money is one of the least important motivating factors. Doing cool projects inevitably ranks first.

Is it fun all of the time? Of course not. We pay for the thrills by wading through mind-numbing technical articles and putting up with unenlightened bosses. But if you love technical challenges, fighting really tough problems that span the range from hardware to software to even the basic science of some devices, embedded is the field for you. Be tough, be determined, think long-term… and have fun.
Flooded with cries for help from developers mired in late and buggy projects, Jack Ganssle has developed a video that will teach you new ways to build higher quality products in half the time. Well known embedded systems guru, writer and lecturer, Jack demonstrates the process of developing firmware in a disciplined manner... without the cost and overhead associated with a heavyweight software methodology.

The first of its kind, this high quality production video will show you proven techniques that will accelerate your product to market, while drastically improving its quality. This video is the only non-vendor training tool that shows you practical solutions that you can implement immediately.

His recommendations are practical, useful today, and tightly focused on embedded system development. You won’t see another clever but ultimately discarded software methodology.

Have you wanted to take Jack’s highly successful “Better Firmware Faster” seminar but can’t travel, or worse yet, can’t find the time, or have a limited budget? This video is made for you! This new video excerpts the seminar’s process and productivity information, and adds new content to show you how to cut development times in half.

A Must-See Video for Developers on a Tight Schedule
Video Highlights

Do You Deserve a Raise?
• Competing with Outsourcing
• The ugly truth about methodologies
• Defeating measurement madness

Partitioning Embedded Systems
• Learn how productivity crashes as program sizes grow. You’ll get numbers culled from thousands of projects showing how normal approaches are doomed to failure.
• Transforming the depressing productivity reality to a new opportunity; how to reengineer the project and use people in different ways to maximize the productivity.

Managing Features and Reuse
• Manage features . . . or miss the schedule
• Overcoming the feature/quality/schedule paradox.
• The truth about reuse . . . and how to overcome the obstacle that usually doom reuse efforts.

Finding a Process that Works
• How to institute a stealth process, one that flies beneath management’s radar screen
• The truth behind XP . . . and the important lessons we must learn from it.
• How the use of the right development approaches can reduce the cost of developing safety-critical code.

Using People Productively
• Minor environmental changes triple software productivity.
• Cubicles are the biggest productivity-busters of all . . . and how to turn that liability into an asset.
• Where to use the very best people.

How to Learn from Failures . . . And Successes.
• Embedded disasters, and what we can and must learn.
• Using postmortems to accelerate the product delivery.
• A seven step plan to firmware success.

Stamp Out Bugs!
• Using standards to improve the code quality
• The best approach to testing
• How to track bugs during the development process.

Contact and Ordering Info:

3 easy ways to take advantage of this offer:
• Visit Jack’s website at www.ganssle.com
• Email us at info@ganssle.com or
• Call us at 410-496-3647

With each video you will receive 2 VHS tapes or DVDs and an 80 page workbook. Your cost is $349.00 plus shipping and handling. Discounts available for quantity orders.

Do you consistently produce quality firmware on schedule? If not . . . what are you doing about it?

Develop Firmware in Half the Time