7 tips for optimizing embedded software

Jacob Beningo - September 26, 2016

In my previous post, I discussed key embedded system characteristics that in many cases require optimization. These characteristics include system timing, code size, RAM usage, and energy consumption. While optimizing each characteristic typically requires its own methods and techniques, there are several general tips developers can follow when optimizing their embedded software.

Tips #1 – Always create a baseline for comparison

Creating a baseline to compare optimization results against is obvious, and yet it would surprise you how many times I encounter teams in a hurry that jump straight into optimizing without any baseline. A baseline measurement is important because there are diminishing returns with each optimization cycle. For example, a first pass at optimizing for energy may result in a 20% improvement. The second, 10%, then 5%, and so on. Developers need to be able to see this trend and quantify the improvement they are seeing in the system as a function of their time input.

Tips #2 – Set an optimization target

Each optimization pass requires more and more time to squeeze just the smallest improvement from the system. Teams need to carefully balance their time investment and determine if the resultant improvement is worth the time. Simply trying to go as low as you can go can become addicting. Before one realizes it, one can be spending weeks optimizing a system that no longer needs to be optimized. Before optimization begins, then, teams should set a target level that -- once achieved -- is good enough for the current application. Achieving that target indicates that the optimization process has completed.

Tips #3 – Get the right tools to make measurements

Optimizing a system can be extremely difficult without the right measurements tools. Optimizing for energy, for instance, can’t be done if there is not an accurate means to measure the system and microcontroller energy draws. In many cases I’ll see teams fail to separate these two different energy measurements and attempt to minimize microcontroller energy when it can’t go any lower. Developers interested in optimizing for performance might find the Seggar System View Utility, that I wrote about in Hands-on: Seggar System View Utility, to be critical to understand which functions are monopolizing the CPU. Without tools that can accurately measure or allow a developer to peer into the system behavior, there is no point in attempting to optimize a system.

Tips #4 – Use optimization tools

There are many areas of embedded software that can be optimized in order to decrease code size or improve performance. In some circumstances there are separate or add-on toolchains that can
perform the optimizations. An example optimization tool is the Somnium DRT optimizer that can be used with GCC to optimize code size, energy usage, and performance.

Sometimes external tools may not be necessary, though. Just selection of the right toolchain may be sufficient. I recently wrote about [Open Source versus Commercial Compilers](http://example.com) where I explored the fact that in Coremark tests, under the same test conditions with the same microcontrollers, commercial compilers generated higher scores when compared with open source compilers such as GCC.

**Tips #5 – Use compiler attributes and #pragma**

In general, I strongly dislike using #pragmas or compiler attributes. Attributes and #pragma are generally non-portable and changing compilers can result in software bugs. However, when it comes to fine tuning embedded software, developers will usually not have a choice. Using attributes and #pragma can improve speed, selectively apply optimization to a single function, and so forth. So for those reasons, developers intent on optimizing software should get familiar with attributes, but also read [Writing Portable Optimizations in C](http://example.com) so that they can understand how to write optimizations that are still portable and won’t come back to bite them.

**Tips #6 – Don’t hesitate to experiment**

There is no set-in-stone practice for optimizing a system and developers shouldn’t feel constrained to follow any specific technique. Sometimes the best way to learn and optimize a system is to put together experiments and just see what happens. When I first started optimizing systems for low power, there was a lot of trial and error associated with it. However, through experimentation and recording results, I was able to figure out what works, what doesn’t, and what is a waste of resources and time. A simple example is given in how to [Make the most of printf](http://example.com). Through trying different driver models there are ways to drastically improve the real-time performance a developer gets when using printf, which is usually assumed to be far better than it really is.

**Tips #7 – Dig into the compiler generated instructions** In applications that are extremely resource constrained, there comes a time when developers just need to roll up their sleeves and dig into the compiler generated instructions. Selecting the ternary operator over an if/else could potentially be the difference between three or four extra instructions being executed, which causes the application to implode. While languages such as C are standard, each compiler optimizes and generates the machine instructions slightly differently. The only real way to know what your compiler is doing is to review the assembly. (Check-out [Ternary Operator versus if/else](http://example.com) If you are interested in a low level look).

**Conclusion**

Applications will drastically vary on their optimization needs. Some applications that are in low volume production may require no optimization at all. Others, where every clock cycle or nano-amp matters, a vast amount of time may be spent trying to squeeze every last drop of performance or energy from the system. While each system is different, these tips provide developers and teams with a starting point that can get them on the road to a more efficient system.

*Jacob Beningo is an embedded software consultant who currently works with clients in more than a dozen countries to dramatically transform their businesses by improving product quality, cost and time to market. He has published more than 200 articles on embedded software development techniques, is a sought-after speaker and technical trainer and holds three degrees which include a Masters of Engineering from the University of Michigan. Feel free to contact him at...*
jacob@beningo.com, at his website www.beningo.com, and sign-up for his monthly Embedded Bytes Newsletter here.

Also see:

- 5 embedded system characteristics to keep an eye on
- Make the most out of printf
- Most embedded to become IoT
- Why, When and How: The basics of embedded systems prototyping