



The compiler advantage

CHOOSING THE RIGHT C COMPILER for your embedded-system projects may be a bigger challenge than you expect. The embedded-system-industry trend

is moving from using assembly to using the higher level languages. Assembly language gives you more control over your code size and execution speeds. However, embedded C is easy to understand and portable from platform to platform. Another benefit is that developing code in C generally improves your product time to market. These features are advantageous if you expect your firmware to have longevity or if your staff changes periodically.

A variety of embedded-C compilers is available in the market today. Embedded-C compilers are available as shareware (for example, the Gnu Compiler Collection, <http://gcc.gnu.org>); from controller- or processor-IC vendors; or from high-end, reputable software companies whose selling prices often reflect the sophistication of the software. Some compiler-software companies have great advertisements and competitive prices but inferior products. So, how do you choose your compiler software for now and for the future?

There are many issues to consider when choosing your compiler. The most important are ANSI compliance, size and speed optimizations, portability, and supporting tools.

Adherence to the ANSI standard for the C language is critical for about 80 to 85% of code. This feature facilitates portability from part to part, as well as from software platform to software platform. ANSI-compliant software also integrates easy-to-understand code into your library. This type of code—if it is well-written—will have a shelf life like a good red wine. Even if portability is not an issue, it is important to understand how the compiler differs from ANSI and whether those differences

will impact your development.

If the compiler of interest is ANSI-compliant, you should then examine the software's machine-code-generation capability. You might ask whether the compiler output code is optimized for small code (because memory is precious in controllers or processors) or whether the application executes in a timely or time-accurate manner. These features are somewhat mutually exclusive, but the compiler software should be able to optimize one or the other.

Portability between platforms is critical in larger companies that have a diverse range of products. However, it can also be convenient in smaller companies that may want the flexibility to quickly change controllers during development. Portability allows you to change quickly from one controller to another during application development. It also makes your code libraries more valuable over time.

If you want to easily complete your programming projects, the compiler cannot stand on its own. Every compiler needs an integrated suite of other tools. This tool suite should have, at a minimum, an integrated development environment that comes complete with a simulator and a hardware-debugging interface. Sometimes, you can mix and match these tools from other vendors, but use the vendor for your compiler as a first source. It can be difficult and frustrating to mix and then match tools from various vendors.

Some other characteristics that make up a good compiler are the availability of code libraries and library support. This concept does not fit into organizations with the NIH (not-invented-here) philosophy. An NIH organization deserves

everything it gets (or doesn't get). Libraries with code that implements useful functions help expedite your projects to completion. Many times, these libraries have gone through evaluations and debugging to ensure their value.

Some of the final details in your compiler evaluation have to do with customer service. You need to ask whether your compiler company supports your questions and helps with its software bugs. You will find this information in the company's application-notes library, discussion forums, and software-release history. Although frequent releases can be a sign of responsiveness, they can also be an indication of serious stability concerns. The ideal compiler vendor has more support than you'll ever need.

In the past, most firmware designers would write embedded-firmware programs in assembly language. The embedded-C compiler transforms your source code into machine code. The C compiler you use determines the size as well as the execution time of your code. Many ways exist for determining these factors, so running your own test programs is also important. Don't depend on the results from a compiler-software company that runs its own tests. Many times, these companies can consciously or unconsciously skew the results. □

REFERENCES

1. Ganssle, Jack, *The Firmware Handbook*, Elsevier, 2000, ISBN 0-7506-7606-X.
2. Hyde, Randall, "Why Learning Assembly Language is Still a Good Idea," May 6, 2004, www.onlamp.com/pub/a/onlamp/2004/05/06/writegreatcode.html.

Bonnie Baker is the analog/mixed-signal-applications engineering manager for Microchip Technology's Microperipherals Division. You can reach her at Bonnie.Baker@microchip.com.