

EMBEDDED OPERATING-SYSTEM SOFTWARE— *build or buy?*

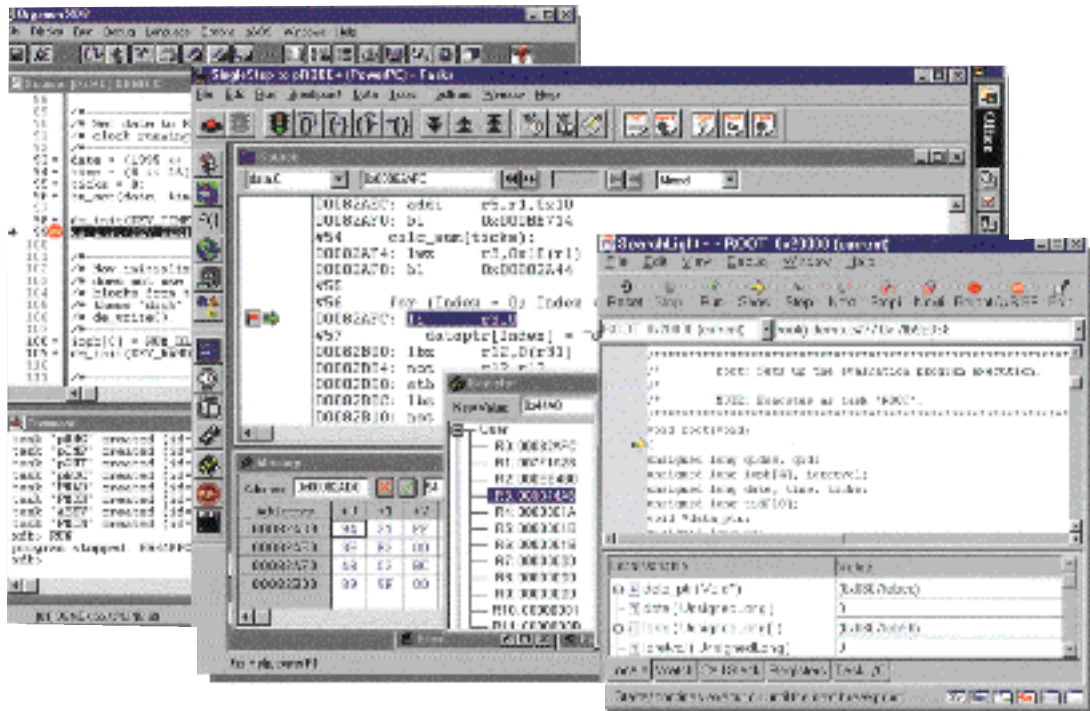
WARREN WEBB, TECHNICAL EDITOR

In spite of the industry's push to commercial off the shelf and the shortened time to market for new products, many embedded-system-development teams continue to write their own real-time operating software. Before you decide to roll your own RTOS, take a close look at your system requirements and develop a plan for the future.

Ask the average embedded-software developer if he would write his own dynamic, multitasking, pre-emptive, real-time operating system (RTOS), and the response would normally be "No, of course not." Yet, intentionally or not, embed-

ded-software developers have written most of today's RTOSs. Sometimes, a simple embedded design grows into a full-blown, real-time system as marketing and management push for more features. Underestimating the hardware and

Integrated Systems' pRISM+ integrated development environment gives fast access to your editor, compiler, code manager, or debugger.



RTOS: BUILD OR BUY

software complexity may also lead the software developer down the path of complete homegrown development. Developers cite reasons such as critical timing, legacy code, or low project funding to justify proprietary software. But before you launch into that next big software challenge, take the time to investigate commercial operating systems. You may bypass some software-development headaches and end up with a lower cost project.

Lack of need is the most common reason that software developers do not choose a commercial operating system. With only one task running at a time, designers feel that they can easily keep track of the required hardware interaction. Many embedded developments start with this mindset and then grow into disaster. Jim Ready, chief technical officer at Microtec (www.microtec.com), says that some designers define their application, grab a C compiler, start at main (), and write code until they are finished. However, embedded systems have a dynamic nature: They respond, control, measure, or interact with the outside world.

@ a glance

- c It's hard to write an RTOS that's faster or smaller than commercial versions without giving up features that you eventually have to add anyway.
- c Many programmers who write their own RTOSs run into subtle and difficult concurrency errors and shared-variable problems.
- c An application that has to handle two or three concurrent interrupts probably needs a commercial RTOS.
- c If you factor in a homegrown RTOS's hidden costs, even a \$15,000 to \$20,000 commercial

com), says that some designers define their application, grab a C compiler, start at main (), and write code until they are finished. However, embedded systems have a dynamic nature: They respond, control, measure, or interact with the outside world.

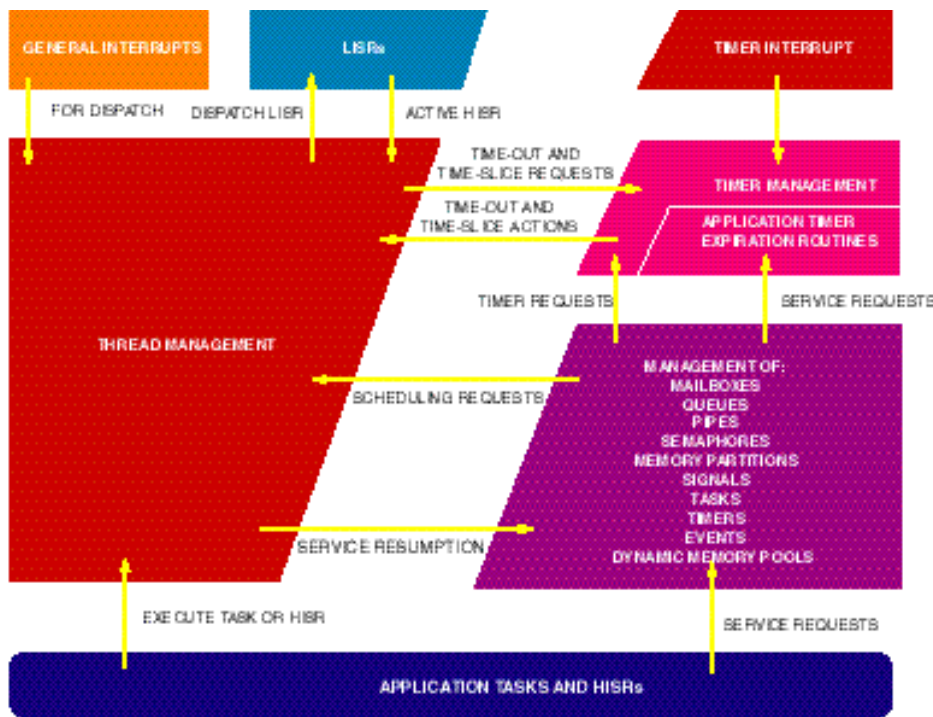
According to Ready, the fundamental embedded-software problem is one of I/O concurrency. You may need to service two or three devices "at the same time." Consider a simple scenario with just two interrupts: You are in the middle of handling the first I/O device, and the second one interrupts. You must save the status and set a flag to remind you that the first process was suspended. Now, you must acknowledge the second interrupt, decide which task has highest priority, re-enable the interrupt system, and resume the highest priority task. At the end of each task, you must check flags to see if any unfinished tasks remain.

Developer problems

Ready explains the problems that developers face. "What people don't realize is that they are building the elements of a small multitasking kernel and probably repeating all of the mistakes that we made trying to build these things in the first place. The types of errors that you get are very subtle. They are concurrency errors or shared-variable problems. It took [mathematics professor EW] Dijkstra quite some time to develop semaphores to solve the shared variable problem (Reference 1). And I will challenge most programmers to develop correctly functioning semaphores on their own."

Some embedded-system developers have situations that justify in-house software. "We rolled our own mini-OS," says Dan Powers, vice president of research at Heartstream (www.heartstream.com) regarding his latest portable-defibrillator product. The design objectives were to pro-

FIGURE 1



NOTES:
HISR=HIGH-LEVEL INTERRUPT SERVICE ROUTINE.
LISR=LOW-LEVEL INTERRUPT SERVICE ROUTINE.

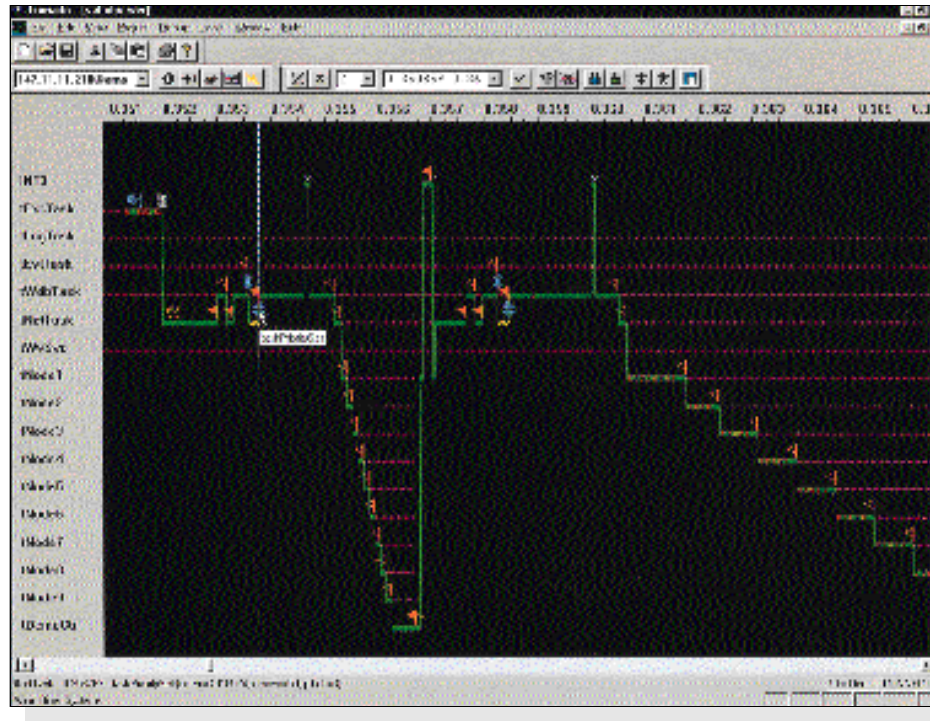
A real-time kernel contains interrupt routines, a task scheduler, memory management, and an interface to the user's application software (courtesy Accelerated Technology Inc).

RTOS: BUILD OR BUY

duce a low-cost and low-power product. Because the product has a one-year battery-standby life and conducts daily self-tests, it needs a low-power, low-MIPS processor. (Heartstream chose the Motorola 68HC16.) Time-dependent portions of the signal-analysis algorithm were hand-coded in assembly language. Safety-critical parts of the code include program flow monitoring to verify state-machine transitions. Design goals and safety rules forced Heartstream to live without the extra memory and processing power necessary to support a commercial RTOS.

Many developers claim that they can produce a smaller or faster kernel than the RTOS vendors. For example, Ready notes, "We have had instances where customers say that their internal kernel is faster. But their kernel does no error checking—or it works for a small number of tasks but slows down as you add more tasks." Commercial-RTOS features have evolved as sort of an average of customer demands. If you can live with a specialized subset of operating-system features or minimized error checking, you may be able to squeeze out a little more performance than from a commercial RTOS. However, do not count on big savings. Ready believes that it would be highly unlikely for an application programmer to use VRTX programmer's reference guide (the external specification) to produce a kernel with better performance or a smaller footprint.

RAM size is another issue. Designers favor popular processors with onboard RAM for small, embedded systems. Sometimes, a 1-kbyte RAM must support both the operating system and the application software. Because the standard fare in commercial operating systems requires a much larger data space, embedded designers choosing these processors have been forced to write their own operating systems. Recently, however, several RTOS vendors have gone after this small-footprint market by introducing their software in versions that have minimal RAM requirements. These versions include Integrated Systems' pSOSystem, Microtec's VRTXmc, and Embedded System Prod-



Wind River's Wind View 2.0 visualization tool allows developers to see the interactions among tasks, interrupts, and system events to simplify debugging.

ucts' RTX. Each version requires less than 512 bytes of RAM space in small configurations.

A good reason to develop your own operating systems is that your new project is an upgrade of a previous project and you wish to use as much legacy code as possible. Commercial-RTOS vendors do not have much of an argument for this reasoning. Sometimes, they can emulate the customer's runtime system to move the application code. However, this approach is probably a losing battle for the RTOS vendor because it pits the vendor against the engineer who wrote the original code and may be emotionally attached to his or her work. No matter who has the final say, management must rely on its in-house technical experts for recommendations.

The not-invented-here factor (NIH) is also a reason that many developers write their own operating systems. Installing software from a third party into a product that is a developer's pride and joy is like admitting that the developer is somehow not up to the task of

creating the software. Developers fear that bugs in code from an external vendor may cause product failure. In-house software developers may also feel that purchasing a large chunk of operating code from external vendors threatens their job security. Software engineers may not admit it, but it is a whole lot more fun and technically challenging to solve the entire problem themselves—even if it means reinventing a few wheels here or there. Once a company switches to a commercial RTOS, it will more than likely use the same RTOS for all future projects.

In addition to the NIH factor, another reason developers write their own programs is that they fear they will lose control over software modifications that compensate for hardware changes or correct bugs. The designer can easily adjust the order of execution or drop to assembly language to solve critical timing problems with in-house-developed software. With a commercial RTOS, the scheduler handles many of the timing issues, thus developers lose the perception of being in total control.

RTOS: BUILD OR BUY

RTOS literature lists hundreds of features and service calls available to handle a large number of user needs. These extensive lists lead the developers to conclude that commercial RTOSs are just too complicated for their needs. Developers see this literature as a lot of detail to learn. Although vendors are pushing the modularized, "take-what-you-need" approach, their literature claims every possible RTOS feature to meet their competition.

"Sticker shock" is certainly a big reason to write your own operating software. The initial license for a full commercial RTOS and associated tools can cost \$15,000 to \$20,000 for one development seat. There are also ongoing royalties for every unit shipped. Royalties are significant until your product volume exceeds 100,000 units. The royalty schedule for Integrated Systems' pSOS is \$500 per unit for one to 99 units, yet it drops to \$1 per unit for 100,000 units. The initial license and royalties can be a major hit to the budget of a low-volume development project.

RTOS vendors respond

Instead of trying to convince the customer that their product is better than the competition's product, RTOS vendors usually end up in the middle of the make-versus-buy decision. RTOS ven-

dors have heard all of the reasons potential customers write their own software, and they have countered with a number of convincing arguments. They promise cost and schedule savings, scalability, life-cycle support, and a more reliable embedded product.

Although a commercial RTOS is expensive, cost savings is an important reason to purchase an off-the-shelf product. Software has displaced hardware as the highest cost item in an embedded-development project. You should give careful consideration to purchasing software because it eliminates the coding, debugging, and documentation of the most complicated portion of your software. Software developers are scarce commodities. If you can reduce the size of the development team, you not only save money, but also reduce recruiting headaches.

Vendors promote product technical support as a major benefit of a commercial RTOS. They can provide continuous support for the operating-system portion of your software by spreading the cost over all customers. Maintaining and supporting in-house software may not always be as easy as it sounds. Sustaining problems or bugs may require pulling the software guru from a subsequent project or possibly rehiring him as a consultant.

A commercial RTOS is modular and scalable. Users can select only those portions or features of the operating system that they need. A subset of a commercial RTOS may reduce acquisition costs and the required-memory footprint. With constantly changing technology, designers want to be able to download new software to add features or revise to their embedded product. Futurists predict that even the simplest embedded products will soon connect to and send data over the Internet. A graphical user interface may also become standard in small, embedded systems—even if just for maintenance. These features are currently standard or optional in most commercial RTOSs but may be very expensive or impossible to add to a proprietary operating system.

An off-the-shelf RTOS also gives the user more options when applying legacy code to new designs or updates. You may select any processor that the RTOS vendor supports. Unless your software is hardware-independent, proprietary software often locks you into a code-compatible family of processors. If you adhere to the discipline of writing your application code to conform to the application-programming interface of one RTOS, you have a good chance of switching to a competing product if necessary.

FOR MORE INFORMATION...

For information on products such as those described in this article, circle the appropriate numbers on the Information Retrieval Service card or use EDN's Express Request service. When you contact any of the following vendors directly, please let them know you read about their products in EDN.

Accelerated Technology Inc

Mobile, AL
1-334-661-5770
www.atinuclous.com

Circle No. 307

Embedded System Products

Houston, TX
1-281-561-9990
www.rtxc.com

Circle No. 308

Green Hills Software

Santa Barbara, CA
1-805-965-6044
www.ghs.com

Circle No. 309

Heartstream Inc

Seattle, WA
1-206-443-7630
www.heartstream.com

Circle No. 310

Integrated Systems Inc

Santa Clara, CA
1-408-980-1500
www.isi.com

Circle No. 311

Microsoft Corp

Redmond, WA
1-206-882-8080
www.microsoft.com

Circle No. 312

Microtec

Santa Clara, CA
1-800-950-5554
www.microtec.com

Circle No. 313

QNX Software Systems Ltd

Kanata, ON, Canada
1-613-591-0931
www.qnx.com

Circle No. 314

Wind River Systems

Alameda, CA
1-510-748-4100
www.wrs.com

Circle No. 315

VOTE . . .

Please also use the Information Retrieval Service card to rate this article (circle one):

High Interest 598
Medium Interest 599
Low Interest 600

Super Circle Number

For more information on the products available from all of the vendors listed in this box, circle one number on the reader service card.

Circle No. 316

RTOS: BUILD OR BUY

"Because of the 'nuclear-arms-race' phenomenon there is great interoperability between RTOSs," explains Ready. In other words, you lose a deal because the competitor has a feature you don't, so you add the feature.

"Another thing that you are buying into from a RTOS vendor when you buy a kernel," says Steve Houtchens, director of new technologies at Integrated Systems, "is that all of the tools are integrated." Most vendors provide a full interactive-development environment, including the source-code editor, the code manager, links to the compiler and linker, software to download your code to the target platform, and one or more debuggers. Houtchens continues, "You can do either line-by-line source-level debugging or task-model debugging, which looks like a software logic analyzer. You're able to monitor which tasks are running at which times, and you can see the stream of data flow. You can also see when a task was interrupted by a

higher level priority item." RTOS vendors all agree that high-quality development tools can dramatically shorten your debugging time (Figure 1).

What about royalties?


The RTOS market has diverged into two camps. The traditional vendor sells a proprietary product, and the customer receives binary object modules, which customers link with their application software. The customer pays the RTOS vendor an initial license fee plus a royalty for each product shipped. Vendors in this area include Microtec, QNX, Wind River, Integrated Systems, and others. These are the largest, most popular RTOS vendors. In response to the often-heard gripes about royalties, a second camp has evolved. This camp licenses the source code for the RTOS and allows the customer to ship products royalty-free. The source may be used as-is or modified to suit the application. Accelerated Technology's Nucle-

us Plus and Embedded Systems Products offer source-code, royalty-free products. Green Hills Software takes a slightly different approach by including its royalty-free VelOSity RTOS with the purchase of its compiler and software-development environment tools.

A discussion of RTOSs would be incomplete without mentioning Windows CE (Reference 2). This year, Microsoft (www.microsoft.com) announced that it is extending Windows CE to include real-time features. Version 3.0 is due out in the first half of 1999 and promises nested interrupts, additional priority levels, semaphores, and a guaranteed response time of 50 msec. Although 50 msec is still a fairly slow response time for some hard real-time systems, Microsoft sees a huge embedded market with a range of timing requirements, plus a loyal group of programmers familiar with the Windows desktop products. At 300 kbytes to 4 Mbytes, depending on compo-

nents selected, Windows CE is one of the largest embedded operating systems.

Although you might get excited when you consider the challenge of developing an in-house operating system, the roll-your-own days may be fading away. You can buy 32-bit processors today for less than \$20, and memory prices are low, too. (A 32-bit RTOS would be quite a challenge.) There are many low-cost, desktop-software packages available to fit your application. Advancing technology dictates that your product be capable of periodic software updates as requirements change. You must also plan for transferring your product to the next-generation hardware platform. So, take the time to analyze your system before embarking on software development. Consider development schedule, software support, expandability, communications, scalability, and future growth. You may find that an off-the-

shelf commercial RTOS is in your future. 

References

1. Dijkstra, EW, "Cooperating sequential processes," in *Programming Languages*, Genuys, F (editor), Academic Press, 1965.

2. Levy, Markus, "Windows CE: at the center of a juggling act," *EDN*, July 17, 1997, pg 38.



You can reach Technical Editor Warren Webb at 1-619-513-3713, fax 1-619-486-3646, wwwwebb@cts.com.

VOTE

Please use the Information Retrieval Service card to rate this article (circle one):

High
Interest
598

Medium
Interest
599

Low
Interest
600

What's Next in EDN

EDN's August 17:

- ◆ Automotive power semis

EDN's September 1:

- ◆ System-on-chip design

EDN's September 11:

- ◆ Testing communications channels

EDN's September 24:

- ◆ 25th Annual Microprocessor Directory