Why Use an RTOS?

by David Moore

Director of Engineering

An RTOS makes development easier for many projects, and it makes them more expandable, maintainable, portable, and secure. Time and cost savings result.

Multitasking

At the core of an RTOS is a multitasking kernel. Multitasking, alone, is enough reason to use an RTOS in many systems. It allows you to break a complex problem into simpler pieces and focus on the development of each task rather than on scheduling when things run. It also makes it easier to partition work among members of a team.

Automatic scheduling lifts a huge burden from you. You can write your tasks and leave it to the kernel to do the magic that makes them work together. There's no superloop that degrades as features are added and no need to continually have to tweak the timing to get important things to run and use spare time for background activities. Having a scheduler is especially important when you add complex communication protocols, such as TCP/IP, and when the application is more than trivial.

Often, what starts as a simple idea becomes more complex. Add to this new requests from marketing and customers. Quickly, your simple loop degenerates into spaghetti code that is difficult to modify and expand. With multitasking, new features are easily added as new tasks. The scheduler handles the rest. The kernel gives the application a simple but solid structure to build on.

Although designing a system on an RTOS from the start is best, it is also possible to migrate an existing system to multitasking using an evolutionary approach. This is the topic of our article *Making the Transition to a Multitasking Kernel*.

Efficiency

A related benefit of multitasking is it allows fuller use of the CPU. Waiting is intrinsic to embedded systems. During waits, the processor can be doing useful work rather than spinning in a loop. The technique used sans multitasking is to insert calls to background functions into these waiting spots within the important foreground code. It is not only difficult to fully utilize the time gaps this way, but such software is also difficult to maintain. As more code is added to either case, and as more operations are added to the main loop, timing is altered, so further and repeated adjustment is needed. This can be a perpetual nuisance and produces results that are less than optimal. It is simpler and better to let an automatic scheduler handle it. All spare time can be utilized, and idle looping is done only when there is nothing that needs to run. Also, idle time is collected into larger chunks, and during these, the processor can be put into a low power state, saving energy. The better efficiency may also allow you to use a cheaper processor.

Services

Kernel services are provided for resource management, memory management, event handling, messaging, interrupt handling and coordination with background code, and more. You avoid implementing similar services, so you can focus on application development. Also, well defined interfaces, designed with a great deal of care and fully documented, will not only offer good ways to do things but also lead to consistency in your application design. The kernel has been developed by people who have aptitude and interest in OS design, and the code has been proven in many systems before yours.

Structure

The foregoing features impose a structure on the application. This results in a desirable consistency of design.

It is a common misconception that all RTOSes are alike. In reality, each has been designed with its own principles that make it unique, and these will have a significant impact on the structure of your application. This is primarily true of the RTOS kernel, which is the key differentiator of each RTOS and the foundation upon which everything is built. For example, the smx kernel strives to minimize interrupt latency, so a key part of the design is the LSR. The LSR has a big influence on how you implement interrupt handling in your application. For more about it, please see our article *Link Service Routines*. Also, some kernels offer simplistic services, while others' are sophisticated, and these, too, impact the design of your application. For smx, some examples are messaging, mutexes, and event groups. You can learn more about these from the smx literature and manuals.

Diagnostics

Since an RTOS imposes a uniform system structure, it is possible to introduce sophisticated diagnostic tools, which further reduce development time, such as a kernel-aware debugger plugin, that helps you see overall system operation and internal details in a structured way. Good ones, such as smxAware, have graphical displays of execution timelines, profiling, resource usage, and memory layout, and textual displays of logged events and details of objects that are more informative than debugger watches.

Middleware

An RTOS is more than a kernel and includes middleware such as file systems, USB, TCP/IP, WiFi, GUI, security options, and bootloader. These modules are already integrated with the kernel and each other and include the drivers for the target hardware. In the case of SMX, a release of all modules is shipped that builds and runs immediately on the selected evaluation or development board. In addition to the modules developed by the RTOS vendor, other third-party modules are often already integrated and offered as well. Even new third-party libraries are often more easily integrated into an RTOS-based system, since they typically have a small, well-defined OS porting layer. Implementing this is typically easier than integrating into a bare metal project.

Portability

A good RTOS abstracts hardware details through a consistent API that defines at least the services it relies on. Therefore, your application will run on any platform the RTOS runs on. This is true of SMX, and its structure makes it easy to build an application for multiple targets from the same codebase.

Security

An RTOS typically offers various security features. One is encryption support for various communication protocols. Another is support for the memory protection unit (MPU) of the processor. The widespread testing of RTOS code by many people on many systems mentioned before assures that the software has relatively few errors, and this offers fewer surfaces for malicious attacks, as the vendors of code analysis tools will tell you.

Manpower

Adding an RTOS to your project is like adding several highly competent engineers, at a small fraction of the cost. A good RTOS has already solved many hidden problems that are time-consuming to solve. Using proven solutions reduces risk and gives your team the best chance of success.

Use of an RTOS also gives you access to the expertise of the engineers who developed each module, and they can advise and support you on your project. We, at Micro Digital, view ourselves as more of a development partner than a software vendor.

Conclusion

We have briefly surveyed the main benefits of using an RTOS. The next step is choosing one. Please continue by reading our article *How to Pick an RTOS*. Also see the smx Datasheet, Special Features, and other documents at www.smxrtos.com/special for information about what makes the smx kernel special vs. others.

David Moore
Micro Digital, Inc.
www.smxrtos.com