

# Future Applications Favor a New Generation of Scalable RTOSs

Newer systems incorporating multiple homogeneous and heterogeneous processing cores will require RTOSs that can be adapted to their processing needs. This will also lead to a new generation of tools.

by Tom Barrett, Quadros Systems

The design of current generation RTOSs used in embedded systems typically operates within a fairly well-defined box. Whether of commercial or in-house origin, the model is familiar: schedule application code processes for execution, manage events, time and memory, move data between processes and handle interrupts. While such operations will remain a requirement, augmenting them with capabilities that go outside the box of the normal RTOS design becomes increasingly important to future applications. One feature in particular—RTOS scalability—will quite likely determine whether the system is both operationally and economically successful.

RTOS scalability has been around for some time but it seems

to have no standard definition. Absent such a standard, one definition is simply the ability to adapt and fit the RTOS to the computational needs of the application. To keep it simple, one can make a conceptual division of RTOS scalability into two basic types, or generations. Initially, First Generation Scaling was concerned with making the RTOS fit a particular memory budget. The next generation of RTOS scalability will continue those same scaling imperatives but will also be concerned with adapting the RTOS to the needs of the application's processing model. Because the processing model will play such an important role in Second Generation Scaling, consider the types of processing an embedded system developer is likely to use.

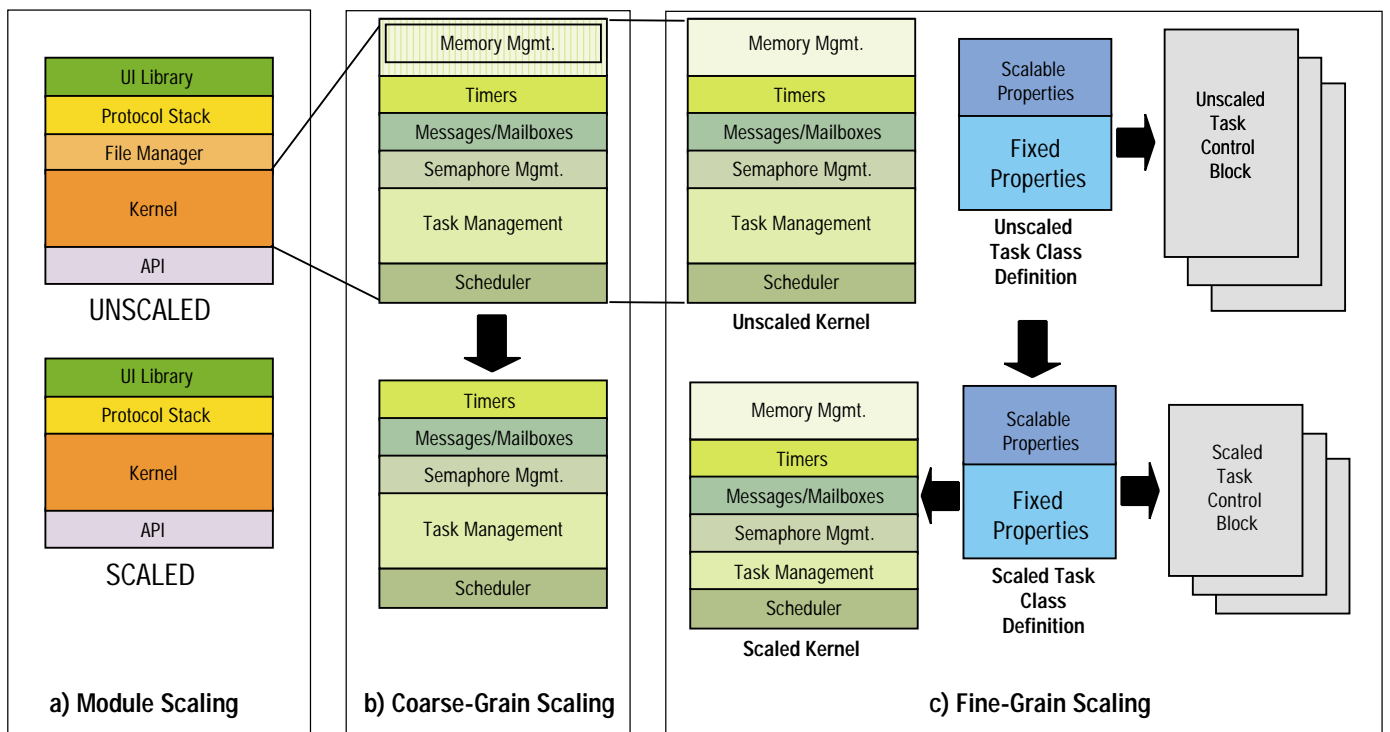


Figure 1 First generation scaling methodologies

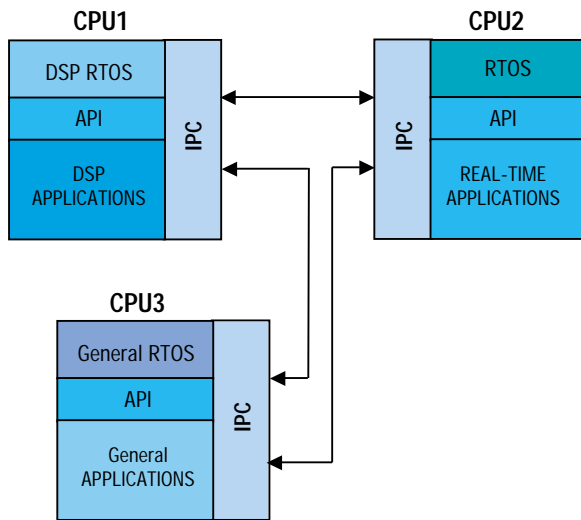


Figure 2 First generation application model

## Basic Processing Models

There are three basic types of processing commonly encountered in embedded systems: real-time (or control), digital signal processing (DSP) and general computation. Each of them has its own unique set of operating system requirements.

Real-time—or control—processes must be allowed to stop and wait for synchronizing events to occur. To support that requirement, control applications usually employ a multitasking RTOS in which the scheduler determines which task gets control of the CPU. Whenever there is a change of processes (a context switch), the RTOS must save and restore the processes' contexts (registers, program counter, etc.)—actions that can involve moving a large number of bytes and that consume a lot of processor cycles. However, such actions make it possible for the processes to stop and start according to the current dynamics of the system, which, though ideal for real-time processing, are not desirable for DSP processing.

DSP, on the other hand, has a data flow nature in which a process executes an algorithm on a block of data without stopping, producing another block of data that it passes on to another stage in the sequence. DSP processing often involves high-frequency I/O, making it important for the RTOS to respond to interrupts with minimum latency. Ideally, the RTOS needs to save and restore a minimum amount of context between execution cycles of DSP processes as well as have low overhead in both the process scheduler and kernel services.

The third model, general processing, usually doesn't require deterministic, preemptive scheduling such as that often required by real-time processing. Ordinarily, a round robin scheduling of processes is sufficient. And it is not uncommon to see time-sliced scheduling, a variant of round robin, employed in this type of processing. By its very nature, general processing isn't time-critical and has looser process scheduling requirements than either DSP or real-time processing.

## First Generation

First generation scaling methods were focused on fitting the necessary RTOS components into the smallest memory footprint, ROM and RAM. Such coarse-grained scaling was limited to including or excluding modules or subsystems as shown (Figures 1 a and b). In both examples, the scaling produces a smaller footprint than that for the unscaled RTOS.

Fine-grained scaling, such as that shown in Figure 1c, allows the user to scale the sizes of objects the kernel uses by configuring the class definition to exclude certain properties. Kernel objects instantiated from such a definition will be smaller, requiring less RAM than would otherwise be used. When coupled with fine-grained scaling, the result is an RTOS that fits the user's specification in both function and footprint.

First generation RTOS designs usually involve a single processor but it is not uncommon to see systems that employ multiple processors segregated according to the type of processing, as

depicted in Figure 2, with the different processors connected via interprocessor communication (IPC) links. However, as system evolution continues toward ever more complex applications, there is a clear need to make development easier, less time consuming and more immune to evolutionary changes.

### Second Generation

It is those evolutionary changes that have brought about the need for a second generation RTOS. While having many new features, the second generation RTOS includes the memory scalability methods

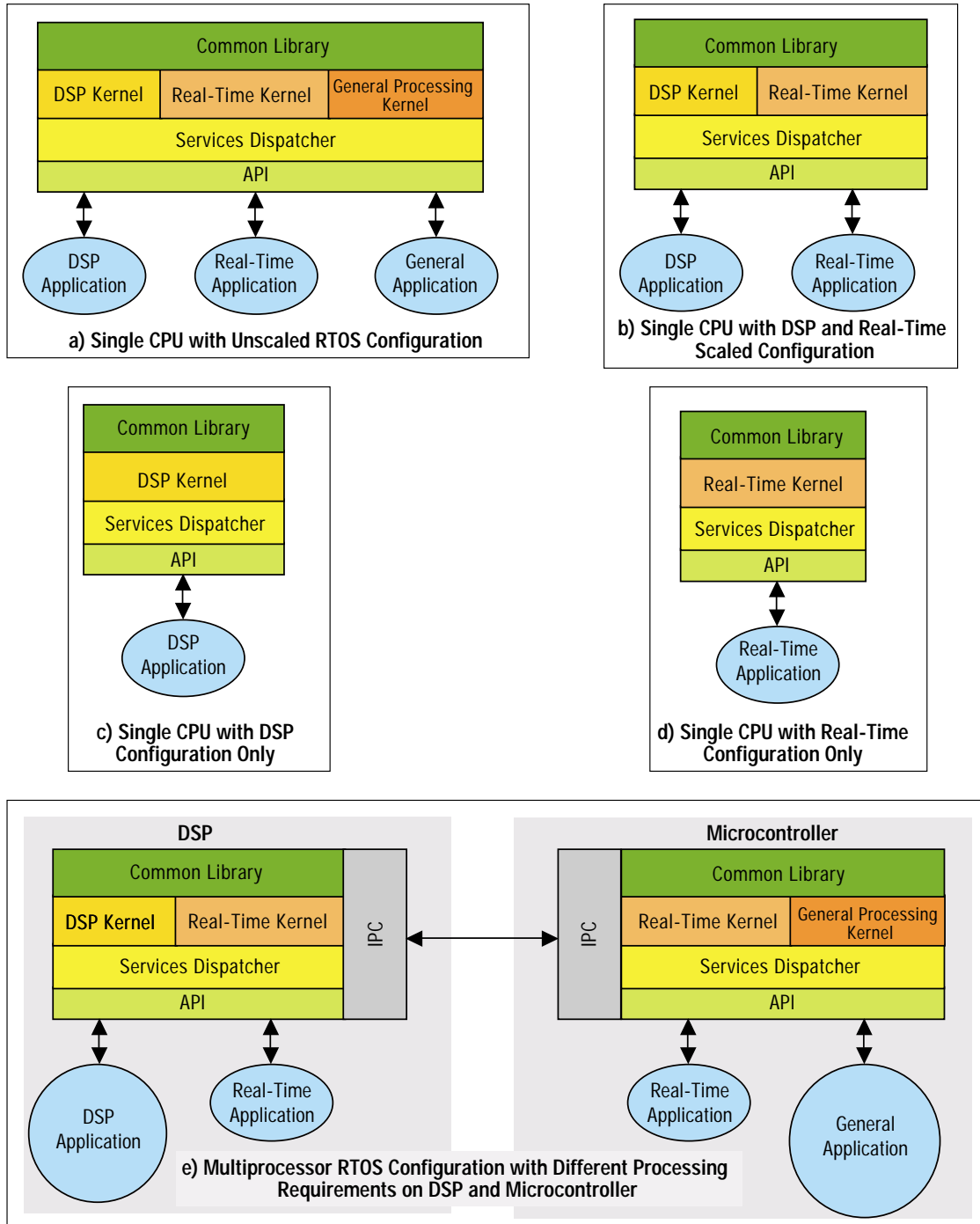


Figure 3 Second generation RTOS scaling

of its predecessors because the imperatives of memory footprint and power budgets are not likely to go away. What differentiates the second generation RTOS from the first is its ability to scale according to the computational needs of the application. By doing so, the second generation RTOS promotes preservation of the investment in application code to a level unattainable in the first generation through a high degree of code reuse, consistent APIs, and common tool usage.

Second generation applications are, like first generation, a mixture of real-time (or control), DSP or general processing. The convergence of these processing models within the same system has previously relied on different types of processors. But new processors are evolving to support the convergent nature of the applications, making it technically sound and economically feasible to implement them in a single core. For example, there are single core microcontrollers such as the Hitachi SH3, Infineon's TriCore or Motorola's StarCore that combine DSP capability with that of a very capable microprocessor suitable for real-time or general data processing.

In addition, there are new multicore processors appearing that will enable new and even more complex applications. Some have heterogeneous cores, like the Texas Instruments OMAP family, combining microcontroller and DSP cores on a single chip. Others have multiple homogeneous cores, such as Agere's StarPro or Motorola's MSC8102, using three and four StarCore processors, respectively.

For convergent applications to realize the full benefits and power offered by these new engines, the system developer should look beyond the typical first generation RTOS to those of the second generation, such as RTXQ Quadros RTOS. The second generation RTOS will support the convergent application requirements with multiple process schedulers, API sets and memory footprints that scale accordingly and without regard to the number of processors in the system.

Figures 3a through 3d depict scaling of a second generation RTOS on a single processor system. Figure 3a shows an integrated design with components to meet the requirements of DSP, real-time (control) and general processing. In Figure 3b, the RTOS footprint and functionality scales down when there is no general processing requirement. And when the system has only a DSP or a control processing requirement, Figures 3c and d respectively show even further scaling to meet the system's particular computational nature. Compare the possibilities these Figures represent to a first generation, single processor application.

A well planned second generation RTOS, however, is not going to be confined to system implementations using a single processor. Figure 3e shows a multiprocessor system in which one processor is heavily DSP-oriented and the other is primarily used for general processing. Both processors run the same RTOS in a peer-to-peer organization with RTOSs scaled according to the needs of the processing models used in the individual processors.

Ideally, applications on one processor are unaware of the existence of the other. For example, both CPUs contain a real-time processing requirement, allowing the developer to design control application code as though it were to run on a single processor. Because of such coding transparency, control application code could conceivably be shifted from one processor to the other to balance the computational loading of the system.

Figure 3 in its entirety reflects the design model of a new generation of scalable RTOSs. It provides the needed scalability, the

benefits of a single API and related kernel services along with the performance that each type of processing requires. The merits of such an RTOS design become further apparent when the processing requirements of the system expand. For example, a DSP system such as shown in Figure 3c could expand to include control processing. As a result, the RTOS configuration would scale to that of Figure 3b and the original code from Figure 3c could be reused without change or with only minor change.

### Development Considerations

Migrating from a first to a second generation RTOS for applications using a single processor is not going to significantly change the requirements for compilers, linkers and debuggers. However, where multiple processors are involved, there will have to be some rethinking of the more familiar development tools, especially debuggers. A configuration tool will be needed to specify and generate the desired RTOS configuration on each individual processor in the system in order to keep developers out of trouble, especially in a system of heterogeneous processors.

Like compilers, Integrated Development Environments (IDE) used in single processor environments will not be in for much change. But IDEs for multiple processor systems will present opportunities for new thinking as will debuggers operating within them. In systems employing multiple homogeneous processors, there need be only one debugger but the user will have to be able to control on which processor the current debugging action is occurring. Additionally, debuggers that have associated kernel awareness modules will have to present kernel information specific to a given processor, not necessarily the one on which the current debug action is taking place.

When the environment employs multiple heterogeneous processors, the difficulty is compounded. The IDE will need to support multiple compilers, assemblers and linkers as well as multiple debuggers. Such an environment is certainly more difficult to work in but not impossible. To date, the ability to operate in such a mixed processor environment is rare and only specialized environments are supported. Because of the close coupling between the tools and the silicon, it is likely that in the near future tools will come from vendors associated with semiconductor manufacturers.

In the final analysis, the evolution of embedded systems continues to be a chicken-and-egg situation with respect to its driving force: Is it the knowledge and ability to solve a problem or the availability of silicon, tools and operating systems used to implement a solution? It is that evolution that is leading the embedded systems industry to a second generation of applications and to the second generation RTOS products to run them.

Code reusability has been and will continue to be the single largest factor in managing the cost of embedded systems. Second generation RTOSs are designed to work with DSPs and microprocessors, allowing designers to preserve their application code investment as they move to multi-core or multiprocessor systems with convergent processing requirements. ■

Quadros Systems  
Houston, TX.  
(832) 351-2830.  
[www.quadros.com].