An Operating System Designed for Convergent Processing

Stephen E. Martin, Director of Marketing, Quadros Systems, Inc.

Many of today's embedded system designs make use of both microcontroller technology and digital signal processing (DSP) technology. Control applications need to efficiently process bit-level control and status information along with fast interrupt response. Mobile communications devices rely on a DSP to handle high speed voice and image processing and utilize a microprocessor to manage user-interface and other general purpose processing.

Device manufacturers have responded by integrating control and data plane processing into single-chip offerings. Microcontroller (MCU) manufacturers have extended their architectures to include DSP functions, such as MAC (multiply-accumulate) instructions, multiple bus and memory structures and special address generation. DSP manufacturers have incorporated features such as integrated peripherals, interrupt-driven I/O, timers, and larger external-memory ranges, offering the code density of an MCU with the high performance and low power of a DSP. Some examples of these convergent architectures include DSP56800E from Motorola, and the ARM11 cores from ARM Ltd. which feature 16-bit multiply-accumulate (MAC) instructions and a pipeline microarchitecture which performs at traditional DSP speeds.

Mixed control and signal processing architectures are not limited to single-core implementations. TI's OMAP® family is a good example of a dual-processor implementation that integrates signal and control/general processing.

While processing architectures have been adapting, the real-time operating systems (RTOS) that support them have not changed much in the past 20 years. The typical RTOS operates within a very familiar model: a task-based execution model with each task having its own independent stack; pre-emptive, priority-based task scheduling; plus a library of services to manage events, time and memory, move data between processes, and handle interrupts.

Given the apparent static nature of the RTOS offerings, many embedded developers now treat the RTOS function as a commodity, differentiated only by selling price and the checklist of middleware integrated with the RTOS: networking stacks, BSP/drivers, files systems. Developers are resigned to adapting their applications to the structure and model of these commodity RTOSes.

©2004 Quadros Systems, Inc. All rights reserved

A New RTOS Architecture

It seems clear that a new RTOS paradigm is needed; one that supports the increasing convergence of processing models; reduces system complexity; adapts to the processing needs of the application; and lowers unit cost and speed development time:

1. **Convergent Processing:** efficiently handle both data flow processes and traditional control processing
2. **Multicore:** designed to implement multicore and multiprocessing (heterogeneous or homogeneous) with a unified model.
3. **Memory Scaling:** provide tight control over memory footprint while delivering rich kernel services
4. **Application Friendly:** easily adapt to the unique processing needs of the application
5. **Scalable Platform for Future:** preserve legacy code while migrating to new processor architectures

Let’s discuss each of these desired RTOS attributes in more detail.

Convergent Processing

An RTOS model for convergent processing should support the unique needs of signal processing, control/real-time processing and general-purpose processing using a common API.

Digital signal processing has a data flow nature in which a process executes an algorithm (generally as a loop) on a block of data without stopping, while 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. Because each block of data represents a new computation environment, there is little residual information that needs to be saved between execution cycles of the algorithm. Ideally, the RTOS will 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. Also, the RTOS should allow for prioritization of signal processing tasks with the ability to preempt lower level tasks.

Real-time/control processes are 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 as a function of its priority. Whenever there is a change of processes (a context switch), the RTOS must save and restore the processes’ contexts (registers, etc.). These actions often involve saving and restoring a large number of bytes and consume a lot of processor cycles. However, such actions make it possible

©2004 Quadros Systems, Inc. All rights reserved

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.

General computation 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 that either DSP or real-time/control processing.

Multicore

Using current RTOS technology to support a multicore implementation requires a separate instantiation of an operating system for each processor or core. To minimize complexity the developer must either adapt the same RTOS model to different processing models or build an abstraction layer to communicate between different operating systems (see Figure 1).

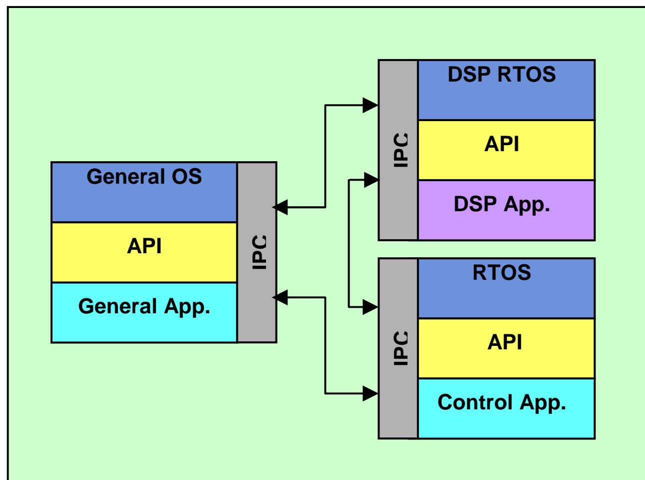

Figure 1. Multi-processor implementation with separate RTOS for each computational model.

The ideal RTOS for multicore applications would manage the system as a single processor, with the ability to execute application code distributed over multiple processors in homogeneous or heterogeneous multicore environments. This RTOS should also be able to handle interprocessor communications (IPC) seamlessly across a number of bus and communications structures (see Figure 2).

©2004 Quadros Systems, Inc. All rights reserved

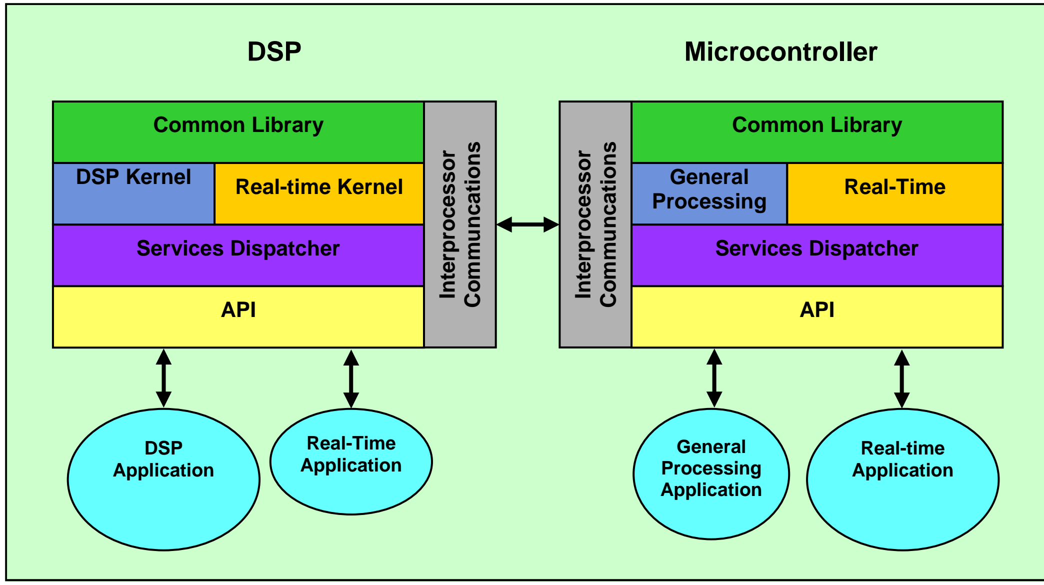

Figure 2. Next-generation RTOS implementation for multicore applications.

Memory Scaling

Coarse grain memory scaling, achieved by including or excluding modules or subsystems, is available from most RTOSs today. Because the large modules must be either included or excluded, with no ability to select or deselect underlying objects, this approach offers limited flexibility. To achieve the smallest memory footprint without sacrificing application performance developers need to have full control over kernel services and not compile in unused or unnecessary code.

The ideal RTOS would allow the user to scale the sizes of objects the kernel uses by configuring the class definition to include/exclude individual properties and services. Kernel objects instantiated from such a definition will be smaller, use less RAM, yet still provide a rich set of kernel services.

Scalable Platform for Future Projects

This new RTOS paradigm would allow the developer to migrate easily to new hardware while preserving valuable application code—the same convergent API structure, syntax and semantics whether for standalone DSP, microcontroller or a multicore/multiprocessor implementation.

While the advantages of code preservation and future design flexibility are quite obvious, real-world situations often conspire to limit the developer's options. When planning the next project development engineers are either beholden to hardware/software decisions made by their

©2004 Quadros Systems, Inc. All rights reserved

predecessors (because the cost to change is too high) or they must live with the significant cost and time delay of developing new applications development. Sometimes the choice of an operating system for a project is made based on the requirements of that project without factoring in the future costs of code migration.

RTXC Quadros Real-Time Operating System

The RTXC Quadros RTOS is an example of a next-generation real-time operating system designed to support both traditional and convergent processing environments. The RTOS features two kernel architectures, RTXC/ss and RTXC/ms, which can be used stand-alone or in combination.

RTXC/ss uses a cooperative multi-threaded architecture which is ideal for data plane applications. All threads run in a single stack, and no context is saved except on preemption. This model is very low latency and operates within a very small footprint. (Typical size is less than 5 Kbytes.)

RTXC/ms uses a traditional multi-tasking architecture for control/plane processing, where each task has its own stack. It is designed for applications needing preemptively scheduled, prioritized tasks. Typical size is less than 25 Kbytes and can be significantly smaller depending on user configuration.

Both architectures share pipes, counters, event sources, exceptions and alarms. RTXC/ms includes support for semaphores, mutexes, queues, mailboxes, messages and memory partitions. (See Figures 3 and 4.)

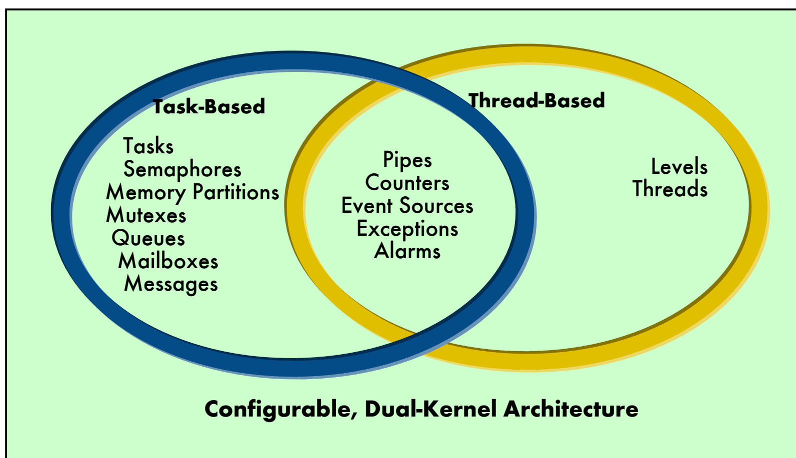

Figure 3. Two kernel architectures can be used in stand-alone configurations or in combination.

©2004 Quadros Systems, Inc. All rights reserved

RTXC/ms Multitasking Architecture for Control Plane Processing	RTXC/ss Multithreaded Architecture for Data Plan Processing
One stack per task	All threads use single stack
Multiple priorities	Multiple priority levels
Preemption between priorities	Preemption between levels
Context saved and restored as needed	No context saved or restored except on preemption
Can wait for an event	Cannot wait for an event
Lower priority than threads	Run to completion within a level

Figure 4. Comparison of the two kernel architectures in RTXC Quadros.

RTXC/dm (dual mode) combines the two kernel architectures into a single framework.This configuration is intended for convergent processing with support for both data plane and control plane processing.

RTXC/mp, the multicore/multiprocessor architecture can combine any number of RTXC/ms instantiations (with or without RTXC/ss) into a unified operating system with the ability to reuse existing application code. The same API structure is used for all implementations, allowing easy of migration and legacy code preservation.

Summary

Silicon manufacturers have continued to advance the state of the art with new convergent processing architectures that integrate microcontroller and signal processing functionality. These new silicon architectures demand a new generation of real-time operating system that will allow applications to take full advantage of this new level of processing capability.

This new RTOS would support control and data plane processing in single processor and multi-processor instantiations. It would offer fine-grain memory scaling to manage the memory footprint. It would be easily configured to fit the computational needs of the application, and its common API would allow for easy migration from less complex to more complex implementations (or vice versa) while preserving the application code. The RTXC Quadros real-time operating system is a good example of a next-generation operating system that will allow developers to take full advantage of the benefits of convergent processing.

©2004 Quadros Systems, Inc. All rights reserved