

Software Requirements for New-Generation Convergent Processing Architectures

Tom Barrett, CEO, Quadros Systems, Inc.

A new generation of convergent processors such as the Blackfin embedded processor from Analog Devices offer the developer a unified, lower cost approach to handling high speed dataflow processing and event-driven control processing. Traditional RTOSes are designed to support only one processing model efficiently in the same processor architecture. The RTXC Quadros dual mode processor combines a light-weight executive for DSP/dataflow with a prioritized, preemptive kernel for control operations.

The drive for lower cost and increased performance in many applications, coupled with the need for reduced development time to meet tight market windows, has created a demand for a new type of processor architecture: the convergent processor—a device that combines the inherent functionality and capability of a traditional microcontroller (MCU) with the data management and processing capabilities of a digital signal processor (DSP). These performance pressures are particularly acute for developers designing high-volume, low-cost handheld devices that may require a wireless connection or an integrated modem, as well as complex high-performance multichannel communications/networking systems such as cellular base-stations and high-speed routers. Such applications increasingly must look to next-generation convergent processor architectures to meet their power consumption, silicon footprint, physical size, system performance and cost constraints.

A good example of a convergent architecture is the Blackfin embedded processor from Analog Devices. This processor combines a 32-bit RISC instruction set with dual 16-bit MAC engines. Its variable length instruction set extends up to 64-bit opcodes used in DSP inner loops, but is optimized so that 16-bit opcodes represent the most frequently used instructions. This processor combines MCU and DSP functionality into a unified architecture that allows flexible partitioning between control and signal processing needs. If the application demands, the convergent processor can act as 100% MCU with code density on par with industry stan-

dards, 100% DSP with clock rates at the leading edge of DSP technology, or some combination in between.

The RTOS Problem

While processing architectures have evolved, the real-time operating systems (RTOS) that support them have changed little in the past 20 years. This creates a problem for developers looking to utilize a convergent processor. An RTOS for a convergent processor needs to allow the designer maximum flexibility to balance the control and media streaming operations to meet his specific design application. This requires a marriage of two very different RTOS architectures.

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; and a library of services to manage events, time and memory, move data between processes, and handle interrupts. This model was developed to handle the event-driven nature of control processing

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 for the processes to stop and start according to the current dynamics of the system, which, though ideal for real-time processing, are inefficient for DSP processing.

Digital signal processing has an entirely different set of requirements. DSPs utilize a data flow architecture 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 operating system 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.

To effectively support this model, the developer needs a different type of RTOS that will save and restore a minimum amount of context between execution cycles 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. Generally developers have used home-grown kernels or specialized cyclical executives for developing DSP-based applications.

RTXC/dm—the convergent RTOS

The RTXC Quadros family of RTOSes was developed to give developers the flexibility and scalability they need to efficiently program with any or all of the major processing models: high speed dataflow processing, control processing, convergent processing, and multiprocessing. The RTXC dual mode (convergent) RTOS is one of four highly scalable

real-time operating systems in the RTXC Quadros family. It combines a traditional task-based kernel architecture for real-time control processing (RTXC/ms) with a specialized executive for DSP and dataflow operations (RTXC/ss) using a common API. This unified RTOS solution enables both types of application code to run fully optimized on a single processor.

All of the kernel functions and services of both RTXC/ss and RTXC/ms are fully available to the developer in RTXC/dm and may be scaled to fit the application requirements using the RTXCgen configuration tool.

To better understand the attributes of this dual-mode RTOS, we need to examine its components—RTXC/ms and RTXC/ss—in more detail.

RTXC/ss — a specialized executive for DSP and dataflow processing

The RTXC Quadros single stack executive (RTXC/ss) is built around a cooperative scheduler that is ideally suited for fast, timely responses to demands for dataflow operations such as streaming or signal processing. Three types of code entities operate in this environment: threads (lightweight, specialized tasks), interrupt service routines, and kernel services. Thread data, interrupt contexts and other variables are all stored on a common stack. The RTXC/ss scheduler grants processor control to threads in response to requests made from other threads or interrupt service routines via kernel services.

An RTXC/ss thread is coded as a C function, but has no context on entry and saves none when it returns processor control to the scheduler. This lack of context makes the transition from the scheduler to the thread very fast, a decided advantage when responding to a demanding operational deadline. During its execution cycle, a thread cannot

RTXC/dm combines a traditional task-based kernel architecture for real-time control processing with a specialized executive for DSP and dataflow operations

explicitly wait for a system event. The lack of context and the inability to block are the two primary attributes that distinguish a *thread* from a traditional RTOS *task*.

Because it has no context at entry, a thread must perform any required data initialization upon entry. When its operations are complete, the thread returns to the RTX/ss scheduler without a return value and without leaving any of its operational data on the stack.

A thread exists within a user-defined priority level. Multiple priority levels may be defined, allowing a thread at a higher priority level to preempt an executing thread running at a lower priority level. A thread with the same priority level as the executing thread cannot preempt it, but must wait until the current thread completes its execution cycle before the scheduler grants it processor control.

The RTX/ss environment is robust and efficient, supporting six object classes that yield over seventy kernel services. It is both code- and data-scalable through the configuration tool, RTXGen, which enables users to specify a configuration with a size between 2-10 Kbytes.

RTXC/ms—a preemptive, event-driven kernel for control processing

The RTX Quadros multi-stack kernel (RTXC/ms) is a traditional yet flexible event-driven, multitasking kernel architecture intended for use in applications such as communications, automotive, process control, and instrumentation systems. It is ideal for systems that require fast interrupt

response time and rapid, deterministic switching between tasks. Each task has its own stack, allowing it to wait for synchronization with system events. In addition to the task stacks, RTX/ms also has a system stack for use in processing kernel services and during interrupt service routines.

Each task has a defined priority, and the default task scheduling policy is preemptive according to priority. The RTX/ms scheduler assigns control of the processor to the highest priority task that is ready to run. If a task is in control of the

processor when a higher priority task becomes ready to run, the scheduler preempts the lower priority task and grants processor control to the one of higher priority.

In addition to preemptive scheduling,

the kernel also supports round robin and tick-sliced scheduling for tasks with the same priority. Because RTX/ms allows independent variables other than time, tick slicing is a general solution to limiting the duration of a task’s run time.

There are three additional code entities in the RTX/ms environment besides tasks: kernel service APIs, kernel services and interrupt service routines. Tasks and interrupt service routines perform the operations required by the application and invoke kernel services through their associated API functions to affect behavior of the system.

The RTX/ms kernel supports eleven classes of kernel objects and approximately 250 kernel services to support those objects. Objects exist for task synchronization, passing data, managing events and counters, alarms, managing memory and exclusive access to entities. Using knowledge of the

RTXC/ss	RTXC/ms
All threads use single stack	One stack per task
Multiple priority levels	Multiple priorities
Preemption between levels	Preemption between priorities
No context saved or restored except on preemption	Context saved and restored as needed
Cannot block or wait for an event (Runs to completion within a level)	Can block and/or wait for an event
Preempts task operations	Lower priority than threads

system design, developers can use the RTXGen utility to select the object classes and their properties, which scales the size and features of the kernel to the configuration best suited to meet the requirements of their application. Fully implemented without scaling, the size of the RTX/ms kernel is approximately 20 Kbytes, depending on the processor and the efficiency of the compiler. It can be scaled down to approximately 4.5 Kbytes, again depending on the processor and compiler.

Managing Threads and Tasks Using Prioritization

So that both processing models can coexist successfully in a single-kernel architecture, the RTX/dm RTOS uses three distinct priority zones. Interrupt servicing occupies the highest priority (Zone 1), which takes precedence over all operations in the other zones. Zone 2, the middle priority, is reserved for all RTX/ss operations, all thread operations and kernel operations of RTX/ms. Operations in Zone 3 are generally classified as system initialization or task operations and include calls to the API library for kernel services. Zone 3 operations occur when there are no Zone 1 or Zone 2 operations needing attention. In RTX/dm, Zone 3 operations are reserved for tasks.

In this model, dataflow applications that run threads always have a higher priority than control plane tasks. This means that operations, such as voice or video processing if organized with threads, preempt any task and run to completion before returning to the preempted task. Task-based opera-

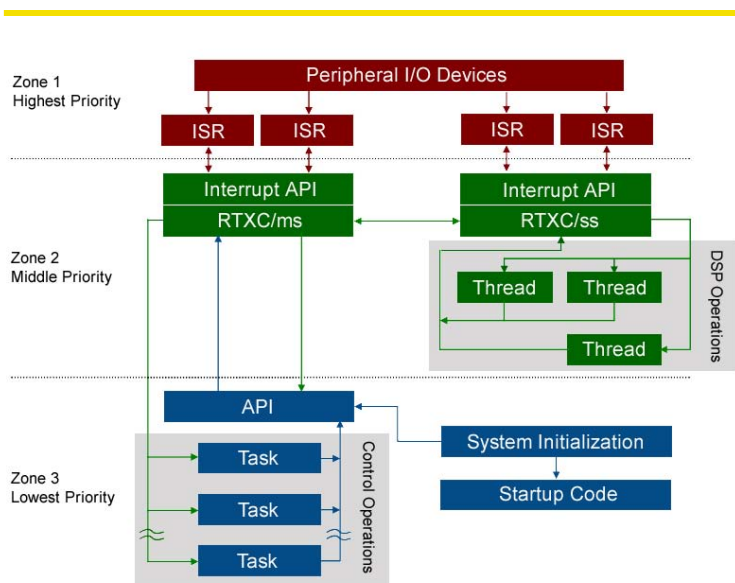
tions can gain control of the processor in between operations in Zone 2. Tasks can initiate threads. Because of the priority of Zone 2 over Zone 3, task-initiated threads cause immediate preemption of all other tasks. In addition, both tasks and threads can be placed at multiple priority levels within their respective zones giving the developer a high level of control over the application.

Summary

Silicon manufacturers continue to advance the state of the art with new convergent processing architectures that integrate microcontroller and signal processing functionality. These new architectures, such as the Blackfin embedded processor from Analog De-

vices, demand a new generation of real-time operating system that enables applications to take full advantage of the higher level processing capability. The RTX Quadros dual mode real-time operating system combines a low-latency cooperative scheduler and a prioritized, preemptive, event-driven scheduler into a single, integrated real-time operating system. Developers can now confidently choose a powerful convergent processor for their next project knowing that they are fully able to maximize the processing efficiency of both control and dataflow with a RTOS that adapts to the needs of the application.

For more information on the RTX Quadros RTOS family visit www.quadros.com.



In the RTX/dm RTOS, dataflow and control processes are organized by priority. Threads always operate at a higher priority than tasks.