

Performance Efficiencies in Convergent Processing

Tom Barrett, CEO, Quadros Systems, Inc.

Convergent processors combine DSP and microcontroller functionality into a unified architecture. Traditional real-time operating systems are not designed to efficiently handle both the synchronous and asynchronous processing capabilities these processors. The RTXQ Quadros dual-mode RTOS was designed from the ground up for these convergent processing applications. Quadros Systems developed a test application to measure the efficiencies of this unique operating system compared to a traditional approach. The results, presented here, show an impressive improvement in processor efficiency with the dual-mode RTOS.

Today's embedded applications are rapidly evolving. The hard line of separation between control processing and signal processing is now a relic of the past. Typical MCU-based control applications now often include streaming media processing and other DSP functions. Traditional DSP applications now combine communications and other control functionality.

A new class of processors has emerged that combines both DSP and RISC/microcontroller capabilities into a single, unified architecture. These convergent processors can operate solely as a DSP engine, or be totally dedicated to a control application, or any point in between. This allows designers of everything from portable devices to industrial control to automotive infotainment to take advantage of lower costs and a smaller footprint, since they can often replace two devices – RISC & DSP - with a single convergent processor. The Blackfin processor from Analog Devices is a prime example of such a convergent device.

Software Issues

The decision to use a convergent processor next leads to a decision about which operating system to use. A traditional multi-tasking RTOS would potentially add an enormous overhead burden to the DSP portion of the application. A simple scheduler, as might be used to support a pure DSP

application, would unlikely be a good fit for the control needs of the application. Let's briefly examine these choices to see where the problems lie.

The Traditional RTOS

The traditional multitasking RTOS model requires that each task have its own stack, allowing it to block while waiting for some event. Each task has a priority that allows it to preempt or to be preempted by a ready task of higher priority. Tasks also have the ability, due to the fact that they have an associated stack, to block (i.e., wait) for purposes of synchronization with system events.

In the traditional control system RTOS, the change of context from one task to another can involve a lot of operations. The current process' context has to be saved on its stack and the new current task's context has to be moved into the physical registers of the CPU. Sounds simple enough, but the fact is that with today's complex processors, a large context, often consisting of several tens of bytes, is the rule rather than the exception. The larger the context, the longer it takes to switch contexts.

This is tried and true operating system knowledge, having been around since the mid-60s. It's a great model for control processing, where tasks need to wait for a synchronizing event, or react to asynchronous events in a timely manner.

The DSP RTOS

DSP applications are typically real-time in nature and follow a block mode or data flow design. In them, a block of data is collected and then passed against an algorithm that loops until it consumes the data, often producing yet another block of data that gets sent to yet another algorithm for still more processing. Due to the real-time nature of the data, the DSP algorithm generally must start within a very tight time window once the input data has been collected. That timeliness means that there needs to be a predictable response time from the point at which the data is ready until the algorithm begins to consume it, and a predictable amount of time to consume the data and to output any data block for downstream algorithms.

To give structure to the processing of DSP algorithms, developers often design a custom executive that is thin on services but very fast and with a small footprint. The minimalist “DSP RTOS” is little more than an ad hoc loop that calls functions that perform the application’s algorithms. When there is some attempt to organize the application around a rule-based architecture, it is usually a cooperative scheduler or a cyclic executive model in which all processes, including the operating system elements, share a single stack. Code processes in either of these two architectures generally have three common characteristics— (1) the process cannot block, i.e., wait; (2) once started, the process must run to completion before another process can be scheduled; and (3) the process has little or no context to save and restore.

The Problem

As long as the application designer stays wholly in the DSP/data flow domain or wholly in the control domain, the two RTOS models just described work quite nicely. The problem arises when the developer wants to take advantage of

the potential that convergent processor hardware offers, where the application is split between DSP/data flow and control code. Certainly, the traditional RTOS model can and has been used for a long time on some DSP processors. In these applications the developer must alter his DSP application to fit the policies, requirements and performance of the control RTOS.

In order to meet the real-time requirements of the DSP application, the user of a task-based, traditional RTOS typically runs the DSP tasks at very high priorities, generally higher than that for a control task. Because DSP operations

often involve processing data as the result of interrupts, there can be a good deal of time spent switching from control tasks to DSP tasks. That increased time is one form of loading on the system. Another form of loading is the amount of time the task spends making calls to the services of the operating system.

Similarly, the use of an RTOS designed for DSP applications is often problematic when applied to a control application. Because control tasks are usually event driven, they lie dormant until some event causes them to wake up, at which time they do their job and then go back to sleep until the next event wakes them up. In simple terms, they need to block until the event occurs and then resume processing at that point. Since typical DSP RTOS processing operations are not designed to block or wait, the developer must alter his application and restructure his control operations.

A Dual-Mode RTOS

Quadros Systems has taken the approach that the operating system needs to be able to adapt to the needs of the application, not vice versa. This is a more natural approach to a workable system. The RTXC Quadros RTOS is a family of

***The problem arises
when the developer
decides to take
advantage of
the full potential of
convergent processing***

four real-time operating systems with a common code base that is designed to provide an optimized environment for any application, whether pure DSP or pure control, or anywhere in between. RTX provides a traditional multi-stack RTOS for control applications; a single stack RTOS for DSP/Data flow applications; a version for multiprocessing systems; and finally a dual mode version addressing the specific requirements of the convergent processor.

The dual-mode RTOS (RTXC/dm) combines a traditional task-based kernel architecture for real-time control processing with a specialized

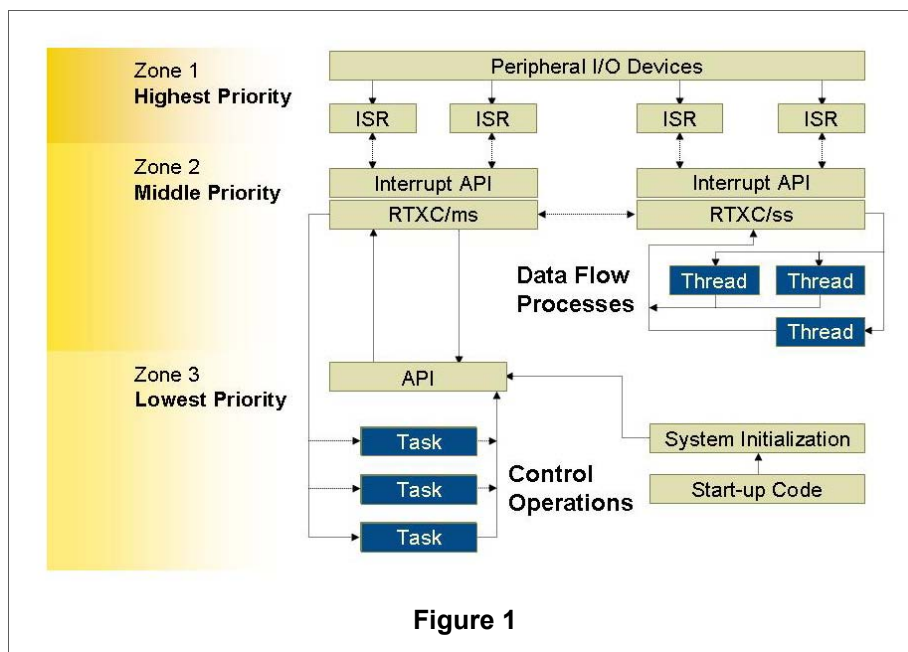


Figure 1

executive for DSP and dataflow operations. The architecture accommodates the different needs of the individual domains, DSP and control, by separating them. Yet even though they are separated, they are united by a common Application Programming Interface (API). This unified RTOS solution enables both types of application code to run fully optimized on a single processor. Such a dual mode approach deals directly with two important issues often associated with the selection of a convergent processor.

First, the two domains are separated much as they have been with domain-specific processors of the past. DSP engineers can still do DSP programming and control engineers can still do control coding. But with RTX/dm, now they can also use the same development tools and can communicate easily between the domains, using the object classes and

related services of the operating system instead of hand-crafted processor-to-processor links.

Second, the DSP processes, lightweight code entities, called Threads in RTX Quadros (not to be confused with Unix, Linux or Windows threads), run at a priority that is higher than all control tasks, ensuring they get access to the CPU in

order to meet their real-time requirements. Their lightweight nature derives from the fact that they have no context, making the switch from thread to thread very fast. Furthermore, threads run at a priority just below that of interrupt services, a position that tends

to reduce Thread startup latency and minimize jitter.

Figure 1 shows a schematic layout of RTX/dm. A prioritization scheme ensures that both processing models can co-exist successfully in a single-kernel architecture. Dataflow applications that run threads (Zone 2) always have a higher priority than control plane tasks (Zone 3). 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 operations can gain control of the processor in between operations in Zone 2.

Comparing the RTOS Models

One very obvious question is: How much difference does all this actually make in a practical application?

To demonstrate the inherent capabilities of a convergent processor when coupled with a matching RTOS, Quadros System created two test applications. These tests provide a side-by-side comparison of a traditional event-driven, multi-tasking RTOS (RTXC/ms) against the RTXC dual mode RTOS (RTXC/dm), when running a dataflow, computationally intensive application – audio decoding and playback.

In these tests we placed an increasingly heavier load on each of the two systems and compare the relative processor utilizations. We selected the Blackfin processor from Analog Devices’ BF533 EZ-KIT Lite evaluation package as an ideal target because of its convergent processing capabilities – totally a DSP processor, totally a control processor, or anything in between.

For the purpose of this test we generated processor load by manipulating the number of interrupts, kernel services and context switches that the system must manage. Because of the way the DMA channel works on the Blackfin processor, it is possible to increase system loading by increasing the number of DMA interrupts the system must process. More DMA interrupts occur by decreasing the size of the blocks managed by the DMA as it feeds the audio codec, which runs at a fixed frequency of 48 KHz. The tests use a fixed size block of memory from which the DMA buffers are assigned such that the number of blocks in that fixed size space (16 Kbytes) determines the size of the DMA blocks.

The Analog Devices 1836 audio codec on the BF533 Lite board supports two independent channels, A & B, and each of those channels has a Left and Right pair. Each audio sample requires four bytes for one side of a channel. Thus, it requires 16 bytes to feed the audio codec for one sample time. As shown in Figures 2 and 3, there is a process called Reformat Filter. It is the job of that process to take a single decoded audio sample and to format it into a sample consistent with the needs of

the needs of the audio codec.

Figure 2 shows a basic flow diagram of the test application when organized under a traditional control RTOS model. (RTXC/ms).

From the bottom, the user selects a song to play. The file name is passed to the music file decoder. The decoder fetches

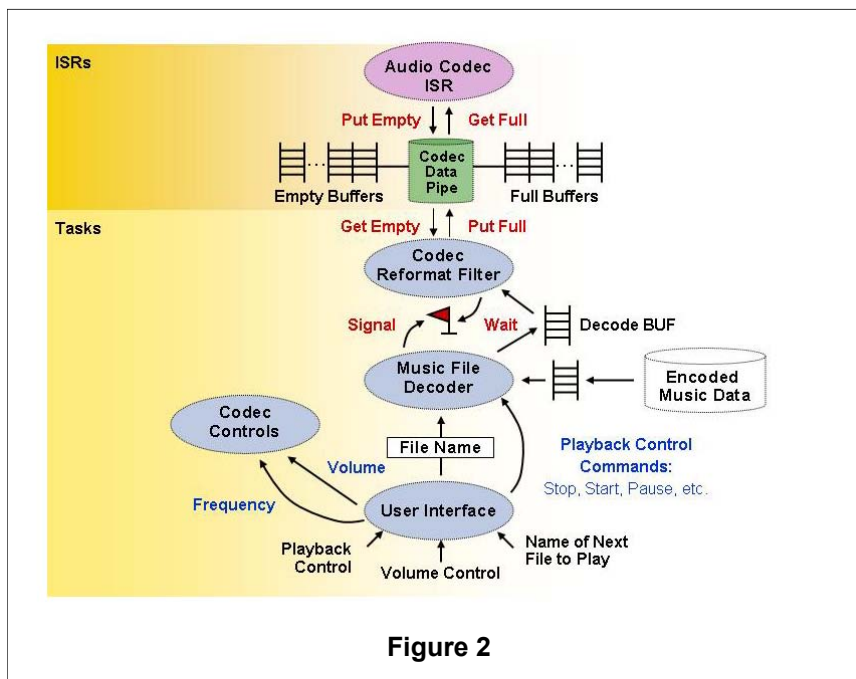


Figure 2

blocks of encoded music (Ogg Vorbis format) from the storage media and decodes them. Decoded blocks of sound samples are passed to a re-format filter task whose responsibility it is to re-form the decoded music data into blocks of data consistent with the needs of the DMA-driven audio codec. The reformat task makes use of RTXC Quadros Pipes to move data along these paths in an efficient manner. An RTXC Quadros semaphore ensures that the Decoder and the Reformat Filter tasks maintain the proper synchronization.

The test progressively increases system loading by selecting the number of DMA buffers available to the application. Larger buffer sizes reduce the number of interrupts and con-

text switches, allowing more continuous compute time. Larger numbers of buffers mean smaller sized blocks that result in more DMA interrupt processing and more task context switches, both sources of system loading.

Figure 3 shows the same application, employing a dual mode RTOS (RTXC/dm), for a test of the Convergent Processing approach.

The same computational elements exist as in the previous model, but this application design shifts the data flow processing of the Decoder and the Reformat Filter to two

Threads on the same priority level. Computationally, the Decode functions in a similar manner but activates the Reformat Filter by scheduling it and then releasing CPU control, allowing the Reformat Filter to run. That is a context switch but as Threads have no context, the amount of registers saved and restored is nil, thereby reducing the cycles necessary to process a buffer.

Loading is further reduced by eliminating the ISR-to-task context switch in favor of switching from the ISR to a Thread. Because there is no context in and out of a Thread, it is possible to reduce the total number of cycles even more.

It takes the same number of cycles to decode and reformat a block of data whether it is done by a Task or by a Thread. Assuming the same size buffers in both models, the number of context switches and interrupts is the same. In fact, the

improved efficiency results from reducing the overhead of the interrupt service processing, context switches and synchronization. It should also be noted that kernel services invoked at the Thread level operate about 3-5 times faster than the same services called from a Task.

In separate loads, we ran the two test applications, using the same number of DMA buffers (4, 8, 16, 32, 64, 96, 128, 192 and 256). Once the playback began, we monitored CPU utilization during the playing of the sound clips. For test

purposes, we limited the duration of the sound clips to about 30 seconds. The results of those test runs are summarized in the graph in Figure 4, where the Y-axis is the average CPU utilization value produced by the program. The X-axis shows the number of DMA buffers chosen for each test.

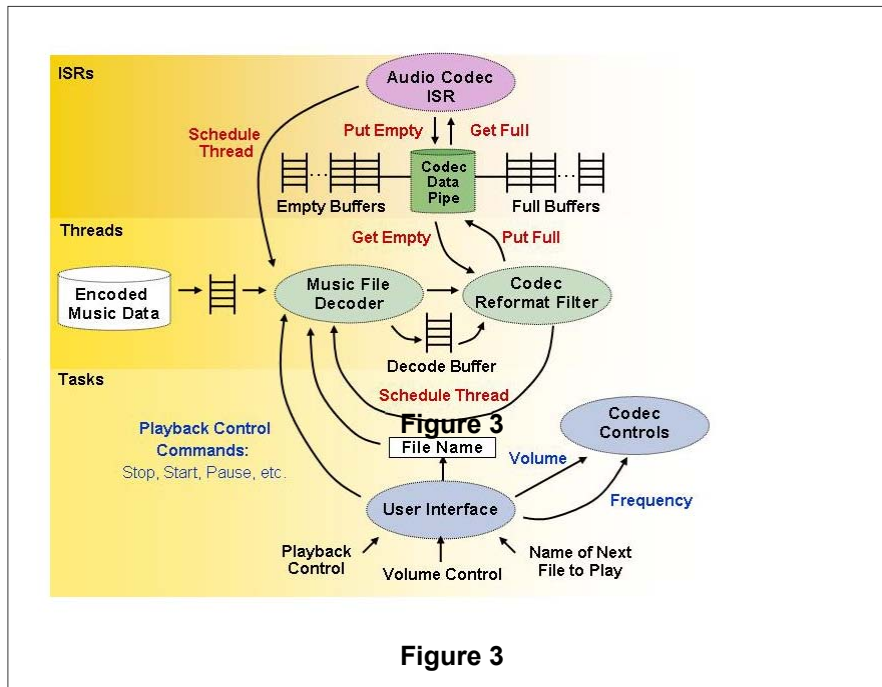


Figure 3

Note that the X-axis uses a non-linear scale.

The performance difference between the two tests begins to appear fairly early, as depicted in Figure 4. With the final choice of 256 DMA buffers, the system loading is heavily influenced by the number of context switches and the interrupt frequency that those 256 buffers put on the system. That selection results in a buffer size of 64 bytes (16,384/256), which is further divided by the requirement of 16 bytes per audio sample. Thus, the choice of 256 buffers represents four audio samples that go to the codec. With the codec running at 48 KHz, those four samples represent a time duration of 83.33 microseconds between DMA interrupts, or, an interrupt frequency of 12 KHz.

By contrast, the Thread-based test consumed fewer CPU cycles, approximately 40% fewer (25% vs 36%), as shown on Figure 5. Throughout the tests at all choices for the number of DMA buffers, there was no perceptible loss of sound quality at all, even at the last selection of 256 buffers. Clearly, the CPU utilization is noticeably better for the Thread-based approach in comparison to the Task-based approach as the number of DMA buffers selected increases system loading.

Conclusion

Convergent processors, such as Analog Devices’ Blackfin processor combine the capabilities and functionality required for both DSP and control processing. Engineers approaching convergent processing from either a DSP or a control background will find these processors offer multiple advantages with few compromises. There is a real-time operating system, RTXC Quadros, which can fulfill the developer’s expectations for an RTOS whether he chooses to use the processor as a DSP engine, a control processor or as a true convergent processor.

The dual-mode RTOS has been demonstrated to provide greater processor utilization efficiencies for applications using both DSP and control processing. Thus it is possible

to use lower clock frequencies and still meet the application’s requirements with a resulting reduction in power consumption. Consider, too, how having more free cycles allows a design to expand features and functions without the

need to use a more powerful processor. And let us not forget the consequent financial benefits gained from a common development environment for the DSP and the control software, as well as the reduced real estate requirement of a single

processor solution. Convergent processing offers all these and more. It is a reality that the developer can use today.

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

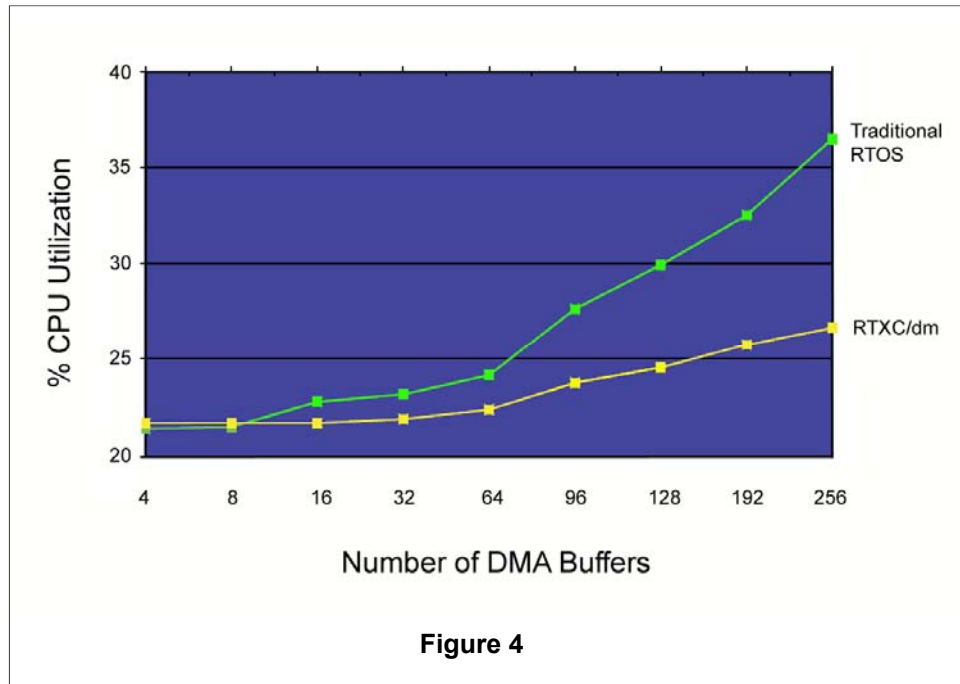


Figure 4