CODE TIME TECHNOLOGIES

Zero Overhead Interrupt Segmentation

Whitepaper

Introduction

A lot of discussion and debate exists in embedded software development about the relative merits of two interrupt handling architectures. The debate surrounds the "unified" and "segmented" models, which take very different approaches to how the critical sections of a real-time operating system are protected during interrupt servicing.

This paper examines these two interrupt handling models, and introduces a novel hybrid, unique to the Abassi real-time kernel, that combines the best features of both. This hybrid does not require any changes to the user application, and operates automatically, with zero overhead.

Protecting the Kernel

Every real-time kernel contains critical sections which must be protected from simultaneous access. This simultaneous access could occur if the running task is executing a kernel service, and an interrupt occurs where the interrupt handler also executes a kernel service. Similarly, if an interrupt handler is executing on a platform that supports nested interrupts, and a higher priority interrupt handler is triggered, both accessing kernel services, a conflict can result. In both cases, the kernel accesses must be performed atomically.

The simplistic approach is to disable interrupts during critical regions in the kernel, and is a hallmark of the "unified" interrupt architecture.

An alternate approach is a "segmented" interrupt architecture, which splits an interrupt handler that needs access to kernel services into two distinct parts. The first part of the interrupt handler cannot access kernel services, and instead queues those requests. When the first part of the interrupt handler completes, the kernel finishes any kernel service that may have been interrupted, after which it services the deferred requests.

Unified Interrupt Architecture

In a unified interrupt architecture, kernel services can be directly accessed from within an interrupt handler. However, to protect the kernel services from simultaneous access, interrupts must be disabled during critical regions in the kernel. This reduces the system's responsiveness to interrupts, since any interrupt that arrives during the time where interrupts are disabled is delayed until the system re-enables them.

While these durations are typically short, the problem is more one of determinism than of duration. If some critical sections require only 20 cycles of protection, while others require many hundred cycles, the system response time becomes hard to quantify.

Futhermore, real-time kernels with a unified interrupt architecture usually place restrictions on which kernel services can be accessed within an interrupt handler, and these accesses are typically through an ISR-specific API. The user is now burdened with two sets of programming interfaces to learn and utilize.

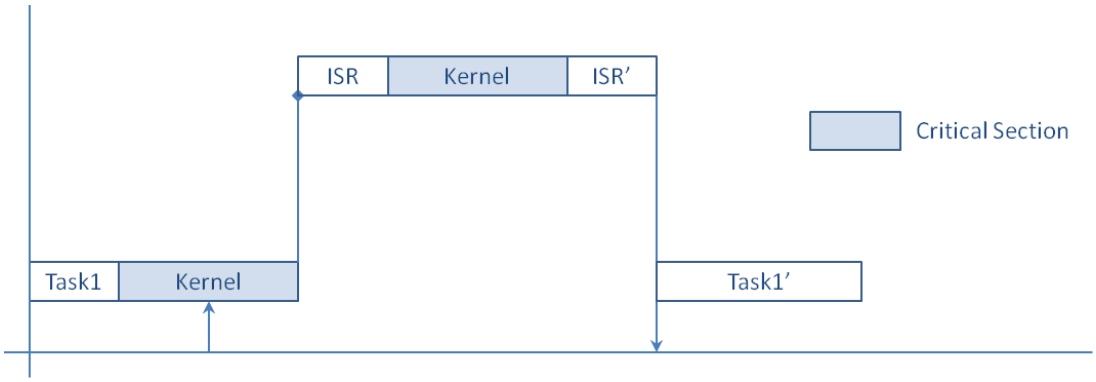

Figure 1. Unified Interrupt Architecture

Segmented Interrupt Architecture

By splitting an interrupt handler that needs access to kernel services into two distinct parts, a "segmented" interrupt architecture eliminates most of the critical sections in the kernel. The part of the processing that requires access to kernel services is offloaded to a secondary interrupt handler (LSR; Link Service Routine), which executes only after the kernel finishes handling any service that may have been interrupted.

The result is faster, more deterministic interrupt response times, since the kernel does not need to disable interrupts. And systems with a segmented interrupt architecture do not usually place restrictions on which kernel services can be accessed from an interrupt handler, and typically feature a uniform API for both ISR and non-ISR accesses.

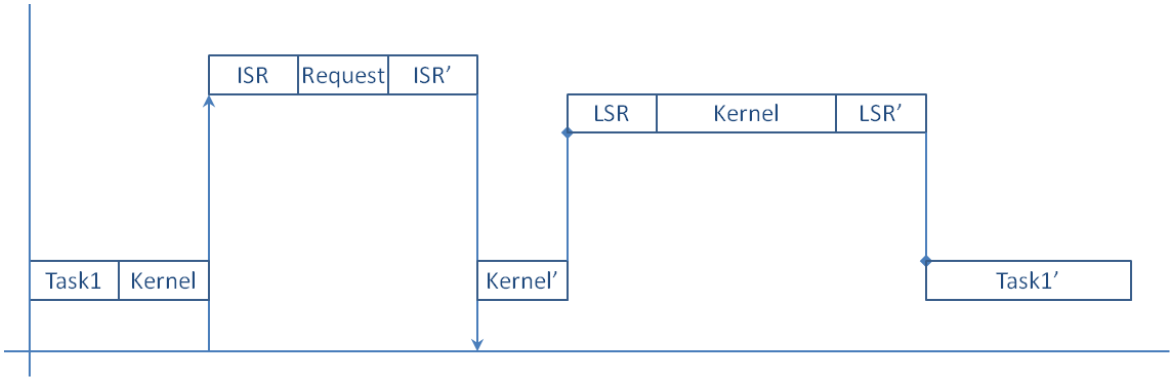

Figure 2. Segmented Interrupt Architecture

Abassi Advantages

One of the most common arguments against a segmented interrupt architecture is the overhead of instantiating a secondary interrupt handler. This typically requires the creation of an additional interrupt context, scheduling overhead, etc.

Zero Overhead Interrupt Segmentation

Abassi introduces a novel hybrid of the unified and segmented interrupt architectures that eliminates the overhead of a secondary interrupt handler, while maintaining full, safe access to all kernel services. This hybrid architecture is unique to the Abassi real-time kernel, does not require any changes to the user application, and operates automatically, with zero overhead.

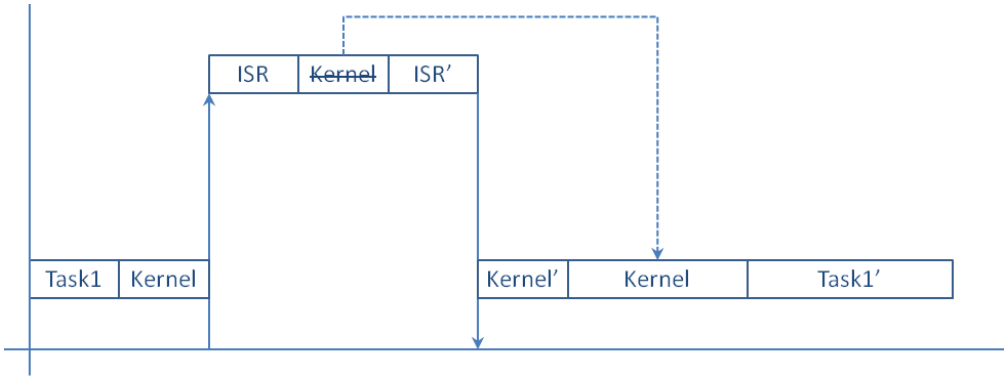

Figure 3. Hybrid Interrupt Architecture

In the Abassi real-time kernel, any access to a kernel service, whether in an interrupt handler or a task, is treated in a deferred manner. In the case of a task, the deferral time is effectively zero, since the request is serviced immediately after it is sent. When the service request arrives via an interrupt handler, it is automatically deferred until after the interrupt handler has completed, and the kernel finishes handling any service that may have been interrupted.

This hybrid approach has all the benefits of a segmented interrupt architecture, providing full access to kernel services within an interrupt handler, and faster, more deterministic interrupt response, but with zero overhead, and without changes to the user application. It also retains the sole strength of a unified interrupt architecture, by eliminating the need for secondary interrupt handlers.

Conclusion

By employing a greenfield design approach, Code Time Technologies has been able to create a next generation real-time kernel that vastly improves upon any available today. The Abassi real-time kernel outperforms all existing platforms by combining code size and CPU efficiency with an unrivalled set of features and usability enhancements.