

Optimizing Multicore

Stephen Lau, Product Management, Emulation Technology, Texas Instruments

What challenges do developers face working with multicore devices?



As the capabilities of end products continue to multiply, many embedded system developers are forced to steer away from working with single core processors and move towards multicore processors. Only multicore processors can deliver on today's demands of increased processing power, robust power performance and lower costs. For developers, making the switch to multicore processors, challenges will arise but the benefit of achieving full multicore entitlement at the end is worth it.

Scaling from a single core to a multicore system requires new insight and knowledge of how to partition threads among processing elements in a multicore device. In addition, processing elements in a multicore device share peripherals, and the optimal use of these peripherals is key to maintaining performance.

In general, there are two groups of multicore processors differentiated by their architectures: homogenous multicore processors and heterogeneous multicore processors. A homogenous multicore processor has two or more identical programmable cores that share peripherals and memory. A heterogeneous processor features multiple unique processing elements each tailored to a specific function, and each core might have only selective access to peripherals and memory.

Developers working with either homogenous or heterogeneous multicore systems need to focus on two classes of problems: synchronization and timing issues and performance evaluation.

Multicore systems have a focus on system integration. The increase in the number of cores has been mirrored by the increased integration of peripherals, bus fabrics, and multi-level memories. Processing elements in a multicore device execute a

Optimizing Multicore

Published on Electronic Component News (<http://www.ecnmag.com>)

portion of the device's functionality. Coupled with improved coding, packaging, and integration practices, traditional debug techniques such as synchronous run, step, and halt remain quite effective in multicore systems. However, when threads interact, timing or synchronization related problems become challenging.

Timing or synchronization-related problems result from dependencies between processing elements. This is exacerbated by the shared nature of system peripherals, bus fabrics and multi-level memories. With dependencies between processing elements, handoffs between threads can be incorrect, inefficient, or have subtle issues. These subtle issues are the most difficult to detect, isolate, and repair. For example, a frame-oriented processing system may miss a frame because processing was not completed in time. The application may not crash, but the application's processing results would be affected. This can manifest itself as one lower quality frame out of thousands. The cause of the timing issues may result from a lack of processing power, contention, or waiting from a shared system peripheral, bus, or memory. This makes these issues hard to isolate. Chip-level tools with insight into the relationship between processing elements and system peripherals, bus, and memory, are critical to optimizing and debugging the timing or synchronization-related problem in a multicore system.

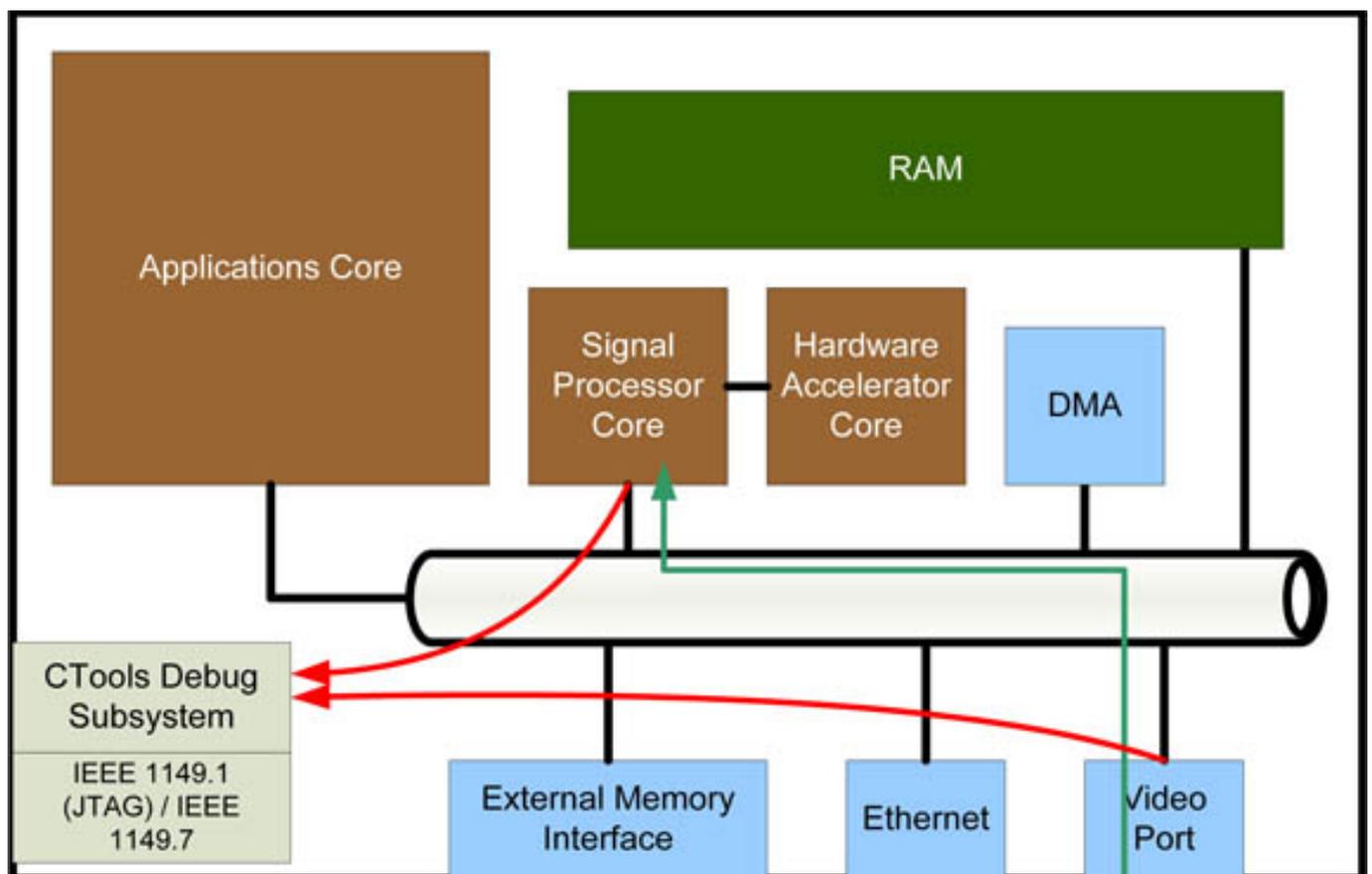


Figure 1. A heterogeneous multicore system monitors video traffic to the signal processing subsystem.

Multicore systems consist of multiple layers. Thus, multiple levels of visibility are needed, with each level providing an appropriate amount of visibility and control

Optimizing Multicore

Published on Electronic Component News (<http://www.ecnmag.com>)

capability to the developer. At the highest levels, a chip level view is needed. This should provide the ability to visualize interactions between processing elements and key peripherals. In addition to seeing the interactions between processing elements, the ability to correlate performance at key peripheral interfaces with tasks on the processing elements is crucial to performance optimization. At the chip level view, correlation of events with an accurate timestamp allows developers to identify and optimize performance in a meaningful way. For example, a user could identify the timing of data transfers between processing elements. In an alternative example, structural performance barriers can be identified, giving developers the ability to optimize for the entire multicore system.

At another level, the visibility may be much more detailed and include the ability to examine and change memory or settings, or a trace of every instruction executed by a processing element. A fine level of granularity provided by processor tracing helps users optimize or debug their software while running in a real-time system at full speed. Simulation environments often have difficulty modeling performance with numerous interrupts and inter-system interactions.

Optimizing multicore devices will be one of the biggest challenges facing developers in the future. The ability to expose a multicore device's tremendous processing potential will require insight into the synchronization and timing between processing elements and peripherals, as well as an understanding of the performance impediments between processing elements, system peripherals, bus fabrics, and multi-level memories. Additionally, new multicore processors will require robust chip-level tools which can provide multiple levels of visibility to aid developers to exploit their potential performance.

About the author

Stephen Lau is responsible for the definition of on-chip debug technology and associated emulator products deployed through TI's Third-Party Emulation Developer Community. He is also responsible for marketing IEEE 1149.7 technology, and developed TI's first commercial IP license for debug technology. Stephen holds a BS in Electrical Engineering from McMaster University in Hamilton, Ontario, Canada.

Source URL (retrieved on 03/09/2014 - 4:19am):

<http://www.ecnmag.com/articles/2010/06/optimizing-multicore>