

Multicore communication: today and the future

By Tony Trawick

With the ever-increasing demands for embedded devices, multicore solutions are becoming more prevalent. The use of multiple cores increases the complexity of software design in many aspects, one of those being the communication between the different cores. How the communication will be used, what it should support, and how it should be implemented are all questions multicore systems software developers will ask. The majority will require a straightforward, simple approach to accomplishing the task.

Increased move to multicore platforms

If you have looked around the embedded hardware landscape lately, you may have noticed hardware vendors are offering a growing number of multicore development platforms. With increasing demands on embedded devices, it is no wonder that more processing power is needed. The move to multicore platforms is the natural evolution for embedded devices considering the convergence of functionality being placed on what have traditionally been known as single purpose devices. Take the phone for example. It has evolved from a device whose main purpose was to place a simple point-to-point call to a device that functions as a mobile phone, gaming unit, camera, media server, Web browser, and more, all in one, as illustrated by Figure 1.

While the convergence of functionality being placed on embedded devices in and of itself may not necessitate the move to multicore, additional considerations such as footprint, energy consumption, heat dissipation, and other aspects of driving a single processor at higher and higher frequencies demand the transition. The move toward multiple cores requires that the software be allocated in some fashion among the different and sometimes specialized processor cores.

The addition of multiple cores also creates the need for communication and synchronization. Software developers want an easy-to-use mechanism that allows them to take advantage of multicore systems efficiently. They also want that method to be extensible in the future. With this in mind, the need for a communication system, what it should support, and how it should be implemented will be explored.

Considerations and assumptions

Communication among multiple cores involves many aspects. However, it should be noted that this discussion does not address a distributed system that communicates over a network. This mechanism is intended for embedded devices and should be as simple as possible, while providing the needed abilities to accomplish its goal: to allow different cores to send data to each other through an easy-to-use API. In the embedded world, developers will typically know how many cores they have available, the data and applications that will be used, and how each specialized core will be utilized.

Today, in most cases, the cores are tightly coupled, sharing a block of memory. With a few notable exceptions, cores are limited to a small number per chip/processor. However, the number of cores contained within a processor is continually growing, making it necessary for the communication scheme to be easily scalable. Also, some situations involve multiple processors on



Figure 1

a single board that could be connected using a type of high-speed interconnect, creating a need for the scheme to support more than shared memory.

Using AMP or SMP solutions

Asymmetric MultiProcessing (AMP) and *Symmetric MultiProcessing* (SMP) are terms used to describe a Real-Time Operating System (RTOS) and hardware architecture of the device. Many detailed papers and books have been written on the subjects of AMP and SMP, so for the purposes of this discussion, analysis will focus on the high-level advantages and disadvantages of each system in a multicore environment and why the need for a simple communication scheme exists no matter which architecture is used.

SMP RTOSs are characterized as having a single image of the RTOS shared by many cores. They also offer the developer many advantages. SMP RTOSs handle the load balancing between the cores, which allows SMP systems to easily scale, but only up to a certain point. Another benefit is that all of the data is available for all of the cores and tasks, so any type of external communication mechanism between cores is not needed. All communication is handled within the SMP RTOS. The major disadvantage to SMP systems is the inability for the SMP RTOSs to operate in heterogeneous environments. This causes problems for the majority of embedded systems in which specialized cores are used in conjunction with general-purpose cores.

AMP RTOSs differ from SMPs in that instead of sharing cores with one image, AMPs have one image per core. The main advantage to an AMP system is that it can be used in a heterogeneous environment. However, AMP systems with large numbers of cores can become unwieldy, creating problems with scalability. AMP systems also will most likely require some type of mechanism to communicate among the cores to realize the full potential of the multicore system.

While it seems advantageous that an SMP system can handle communication by the RTOS, the future may require both architectures to have some type of communication mechanism outside the RTOS. In the not too distant future, a group of general-purpose cores will be able to replace a single general-purpose core, and a group of DSP cores will be able to replace a single specialized DSP core. Each group will have an SMP RTOS, but the two groups will still need to communicate with each other just as AMP systems do today. For current situations, it appears an AMP system is optimal, but whether it or an SMP system is used, a communication method between heterogeneous groups will be needed, as depicted in Figure 2.

Communication system details

After determining that a communication mechanism is necessary, developers need to concern themselves with the implementation details. Factors such as the physical interface supported, availability of guaranteed communication, protection, and configurability of the mechanism are all important factors that affect the overhead, speed, complexity, and scalability of the communication mechanism.

Today, communicating over shared memory handles the majority of cases needed, but as mentioned earlier, developers want a system that is extensible. Communication over other types of interconnects will be necessary in the future, and it is imperative to have the ability to add future interconnects interchangeably, providing transparency to the applications.

Guaranteeing that data be sent and received correctly between cores is very important and can also be a source of tremendous overhead to the system. In tightly coupled systems, the transport mechanisms such as shared memory are stable and can be trusted to send and receive data correctly. When receive notification is needed, a simple low-overhead protocol can be developed, taking advantage of the system being tightly coupled.

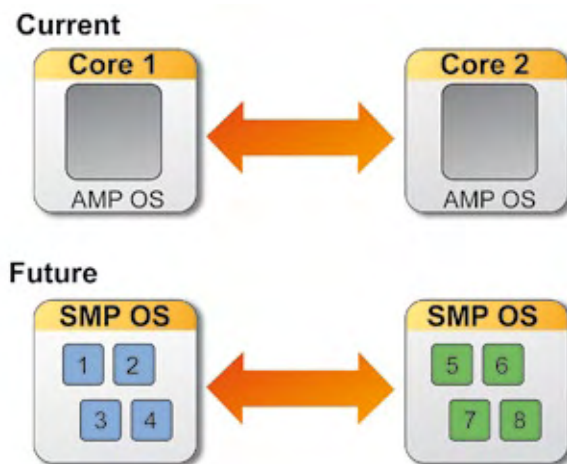


Figure 2

Typically in shared memory implementations, spin locks will be used for the protection mechanism. A spin lock usually takes advantage of a hardware atomic test and set operation. It operates by basically locking out other operations from using the protected entity. Waiting operations just spin in a loop until the protection is removed. The granularity of these spin locks greatly affects the speed and usability of the system. A very coarse granularity would have a single spin lock controlling the entire shared memory region. A very fine granularity would have a spin lock covering every single structure and object in the shared memory region. Each has its advantages and disadvantages, but the most efficient granularity will be between these extremes.

It is a tricky balancing act in the embedded world to provide configurability without creating burdensome overhead. For embedded devices, the developer will typically have a good idea of the amount of data to be communicated at one time and how often the data will be sent. This allows, for the most part, configuration to be done at compile time, removing the overhead of dynamic allocation but still allowing the developer to customize to any particular needs.

Scalability of the communication mechanism is also an important factor to consider, especially with the growing number of cores per chip expected in the future. Having to change communication mechanisms because more cores were added to the system is not very appealing and could be very expensive.

Multicore platforms are here to stay

Multicore solutions for embedded devices are here to stay and will only become more prevalent in the future. Multicore solutions today mostly contain two to four cores on a chip/processor, but this number has the potential to grow very large in the near future. Some cores today have upwards of 300 DSPs per chip. This number will continue to grow and be accompanied by combinations of different specialized cores.

Whether one opts for an AMP or SMP RTOS architecture, an external communication mechanism between cores or groups of cores will be necessary. The details of this mechanism should also be very important to the developer. The overhead, speed, and complexity should be considered, as well as the ability to support future additions with scalability. To ignore these factors could prove very costly. **ECD**

Tony Trawick is a software engineer for Mentor Graphics, working in the Nucleus PLUS kernel development group. He has worked in the embedded industry for more than four years,



and his embedded experience includes kernel porting, device driver writing, multicore/processor support, and kernel development. Tony is a graduate of the University of South Alabama with a BS in Computer Science and Physics as well

as an MBA from Spring Hill College in Mobile, Alabama.

To learn more, contact Tony at:

Mentor Graphics

739 N. University Blvd.

Mobile, AL 36608

251-208-3551

tony_trawick@mentor.com

www.mentorgraphics.com