Running AMP, SMP or BMP Mode for Multicore Embedded Systems

QNX Software Systems

Introduction

Multicore processors have become mainstream, introducing advanced levels of performance to servers, desktops, netbooks and even the latest generation of tablets. The potential benefits for embedded systems are even greater. Networking elements, medical devices and defense and aerospace applications are all growing in complexity and demanding ever-increasing computational power. At the same time, many of these systems must continue to address the thermal dissipation and low power constraints inherent to embedded devices. Freescale QorIQ processors directly address these requirements by providing much better processing capacity per watt and per square inch than conventional single-core processors.

Multicore processors such as Freescale QorIQ platforms are, in effect, multiprocessing systems-on-chip (SoC). Many Freescale SoCs have separate L1 and L2 caches per core, but use a shared L3 cache, memory subsystem, interrupt subsystem and peripherals. To take advantage of these processors, embedded developers must graduate from a serial execution model, where software tasks take turns running on a single processor, to a parallel execution model, where multiple software tasks can run simultaneously. The more parallelism developers can achieve, the better their multicore systems will perform.

Table 1: Three Approaches to Multiprocessing

Model	How it Works	Key Advantages
Asymmetric multiprocessing (AMP)	A separate OS, or a separate copy of the same OS, manages each core. Typically, each software process is locked to a single core (e.g. process A runs only on core 1, process B runs only on core 2, etc.).	Provides an execution environment similar to that of uniprocessor systems, allowing simple migration of legacy code. Also allows developers to manage each core independently.
Symmetric multiprocessing (SMP)	A single OS manages all processor cores simultaneously. The OS can dynamically schedule any process on any core, enabling full utilization of all cores.	Provides greater scalability and parallelism than AMP, along with simpler shared resource management.
Bound multiprocessing (BMP)	A single OS manages all cores simultaneously. As in SMP, the OS can dynamically schedule processes on any core. However, the developer can also lock any process (and all of its associated threads) to a specific core.	Combines the developer control of AMP with the transparent resource management of SMP. The option to lock threads to any core simplifies migration of legacy code and allows designers to dedicate cores to specific operations.

To address these challenges, developers must find tools that can analyze the complex system-level behavior that occurs in a multicore chip. At any instant, threads can be migrating across cores, communicating with threads on other cores or sharing resources with threads on other cores—complex interactions that conventional debug tools were never designed to analyze.

Fortunately, vendors such as QNX Software Systems have introduced system tracing tools that provide a comprehensive view of multicore behavior, allowing the developer to visualize interactions between cores and eliminate a variety of performance bottlenecks. Using the information that these tools generate, the developer can reduce resource contention, optimize thread migration, identify opportunities for parallelism and achieve maximum utilization of every processor core.

Multiprocessing Modes and the Role of the OS

Developers must also choose the appropriate form of multiprocessing for their application requirements. This choice will determine how easily both new and existing code can achieve maximum concurrency. As Table 1 illustrates, developers have three basic forms to choose from: Asymmetric multiprocessing (AMP), symmetric multiprocessing (SMP) and bound multiprocessing (BMP).

Asymmetric Multiprocessing (AMP)

AMP provides an execution environment similar to that of conventional uniprocessor systems, which most developers already know and understand. Consequently, it offers a relatively straightforward path for porting legacy code. It also provides a direct mechanism for controlling how the CPU cores are used. And, in most cases, it lets developers work with standard debugging tools and techniques.

AMP can be either homogeneous, where each core runs the same type and version of OS, or heterogeneous, where each core runs either a different OS or a different version of the same OS. In a homogeneous environment, developers can make best use of the multiple cores by choosing an OS that offers a distributed programming model, such as the QNX® Neutrino® RTOS. Properly implemented, the model will allow applications running on one core to communicate transparently with applications and system services (e.g. device drivers, protocol stacks) on other cores, but without the high CPU utilization imposed by traditional forms of interprocessor communication.

A heterogeneous environment has somewhat different requirements. In this case, the developer must either implement a proprietary communications scheme or choose two OSs that share a common infrastructure (likely IP based) for interprocessor communications. To help avoid resource conflicts, the OSs should also provide standardized mechanisms for accessing shared hardware components. In virtually all cases, OS support for a lean and easy-to-use communications protocol will greatly enhance core-to-core operation. In particular, an OS built with the distributed programming paradigm in mind can take greater advantage of the parallelism provided by the multiple cores.

In the homogenous example shown in Figure 1, one core of the P2020 processor handles ingress traffic from a hardware interface while the other handles the egress traffic. Because the traffic exists as two independent streams, the two cores don't need to communicate or share data with each other. As a result, the OS doesn't have to provide core-to-core IPC. It must, however, provide the real-time performance needed to manage the traffic flows.

Figure 1: Using Homogenous AMP to Handle Both Ingress and Egress Traffic

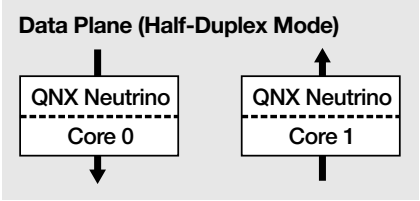
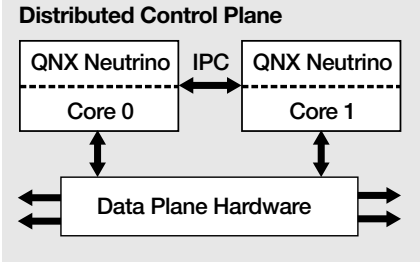


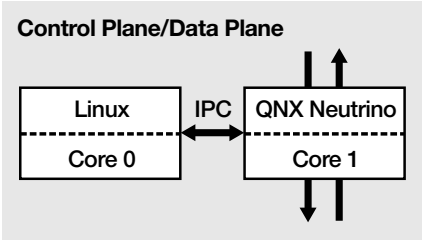
Figure 2 shows another homogenous example, but this time both e500 cores implement a distributed control plane, with each core handling different aspects of a data plane. To control the data plane correctly, applications running on the multiple cores must function in a coordinated fashion. To enable this coordination, the OS should provide strong IPC support, such as a shared memory infrastructure for routing table information.

Figure 2: Using Homogenous AMP to Implement a Distributed Control Plane



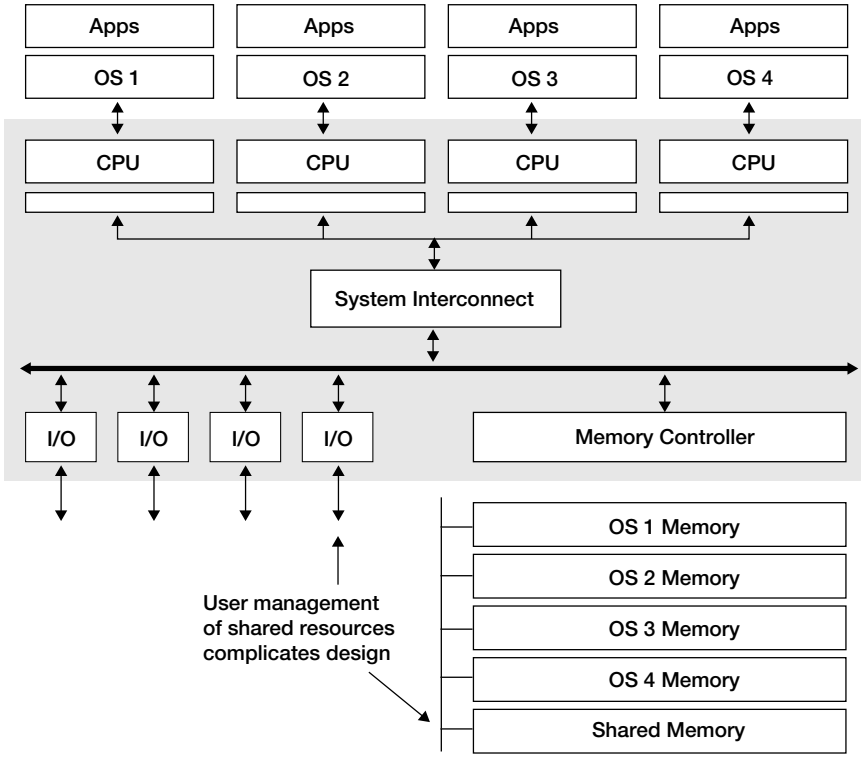
In the heterogeneous example shown in Figure 3, one core implements the control plane, while the other handles all the data plane traffic, which has real-time performance requirements. In this case, the OSs running on the two cores both need to provide a consistent IPC mechanism, such as the transparent inter-process communication (TIPC) protocol, that allows the cores to communicate efficiently, possibly through shared data structures.

Figure 3: AMP Control/Data Plane



In virtually all cases, OS support for a lean and easy-to-use communications protocol will greatly enhance core-to-core operation. In particular, an OS built with the distributed programming paradigm in mind can take greater advantage of the parallelism provided by the multiple cores.

Figure 4: AMP Multicore System



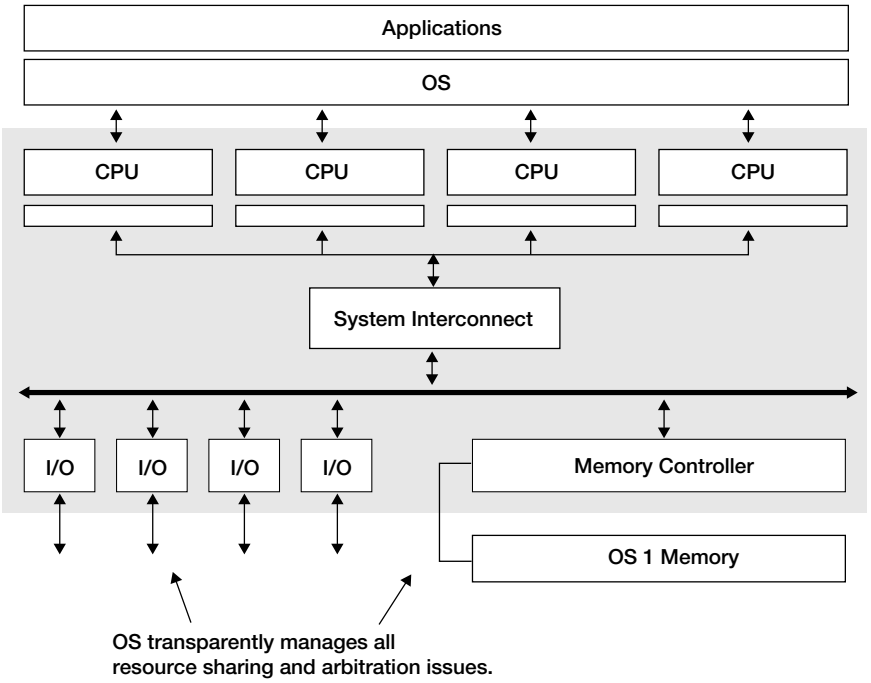
CPU Utilization in AMP Mode

In AMP mode, a process and all of its threads are locked to a single processor core. While this approach is useful for running legacy code, it can result in underutilization of processor cores. For instance, if one core becomes busy, applications running on that core cannot, in most cases, migrate to a core that has more CPU cycles available (refer Figure 4). Though such dynamic migration is possible, it typically involves complex checkpointing of the application's state and can result in a service interruption while the application is stopped on one core and restarted on another. This migration becomes even more difficult, if not impossible, if the cores use different OSs.

Symmetric Multiprocessing (SMP) Mode

Allocating resources in a multicore design can be difficult, especially when multiple software components are unaware of how other components are employing those resources. SMP addresses many of the issues by running only one copy of an OS across all the chip's cores. Because the OS has insight into all system elements at all times, it can allocate resources on multiple cores with little or no input from the application designer. By running only one copy of the OS, SMP can dynamically allocate resources to specific applications rather than to CPU cores, thereby enabling greater utilization of available processing power.

Figure 5: SMP Multicore System



Because a single OS controls every core, all intercore IPC is local. This reduces the memory footprint and improves performance dramatically as the system no longer needs a heavy networking protocol to implement communication between applications running on different cores. Communication and synchronization can take the simple form of POSIX primitives (e.g. semaphores) or a native lightweight local-transport capability such as QNX distributed processing.

A single instance of the OS across all cores simplifies optimization and debugging. Visualization tools such as the system profiler in the QNX Momentics® Tool Suite can track thread migration from core to core, scheduling events, application-to-application messaging, CPU utilization and other events, all with high-resolution timestamping.

A well-designed SMP OS such as the QNX Neutrino RTOS allows the threads of execution within an application to run concurrently on any core. This concurrency makes the majority of the compute power of the chip available to applications at nearly all times. If the OS provides appropriate preemption and thread prioritization capabilities, it can also help the application designer ensure that CPU cycles go to the application that needs them the most.

In the control plane scenario in Figure 5, SMP allows all of the threads in the various processes to run on any core. For instance, the command-line interface (CLI) process can run at the same time that the routing application performs a compute-intensive calculation.

Once designed, a process can run equally well on a single-core, dual-core, or N-core system, the only potential change being the number of threads that the application needs to create to maximize performance. In full SMP mode, an RTOS like QNX Neutrino will schedule the highest-priority ready thread to execute on the first available CPU core. As a result, application threads can utilize the full extent of available CPU power rather than being restricted to a single CPU.

Bound Multiprocessing (BMP) Mode

BMP, an approach pioneered by QNX Software Systems, offers the benefits of SMP's transparent resource management, but gives designers the ability to lock any application (and all of its threads) to a specific core to help migrate uniprocessor code to a multicore environment.

As with SMP, a single copy of the OS maintains an overall view of all system resources, allowing them to be dynamically allocated and shared among applications. But, during application initialization, a setting determined by the system designer forces all of an application's threads to execute only on a specified core.

Compared to full, floating SMP operation, BMP offers several advantages.

- Allows legacy applications written for uniprocessor environments to run correctly in a concurrent multicore environment, without modifications
- Eliminates the processor-cache "thrashing" that can sometimes reduce performance in an SMP system
- Enables simpler application debugging than traditional SMP by running all execution threads within an application on a single core
- Supports simultaneous BMP and SMP operation, allowing legacy applications to coexist with applications that take full advantage of parallelism of multicore hardware

Figure 6: Using BMP for Half-Duplex Mode

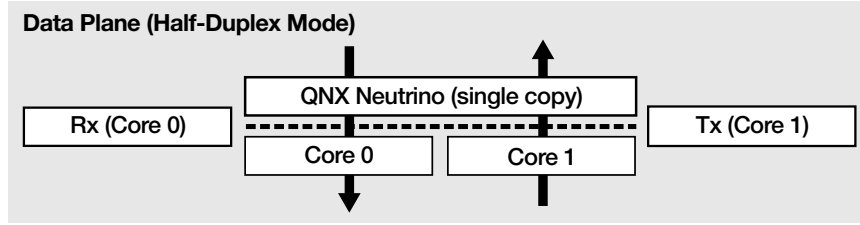
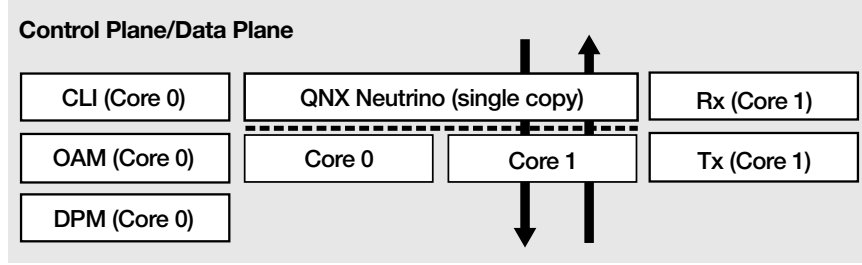


Figure 7: Using BMP for Both Control-Plane and Data-Plane Operations



In the example shown in Figure 6, a BMP system is running in half-duplex mode. A receive process with multiple threads runs on core 0 and a transmit process, also with multiple threads, runs on core 1. As in SMP, the OS is fully aware of what all the cores are doing, making operational and performance information for the system as a whole readily available. This approach spares developers the onerous task of having to gather information from each of the cores separately and then somehow combining that information for analysis.

In the example shown in Figure 7, control plane applications (command-line interface; operations, administration, and maintenance; data plane management) run on core 0, while data plane ingress and egress applications run on core 1. Developers can easily implement the IPC for this scenario, using either local OS mechanisms or synchronized protected shared memory structures.

A Solid Foundation

Making the leap to multicore processors may seem daunting at first, but the benefits can far outweigh any potential misgivings. The choice of hardware and software is not to be taken lightly and may dictate many design decisions and ultimately the success of the project. Selecting an OS that supports processing models which allow for both migration of legacy code and full symmetric operation while minimizing complexity provides a solid foundation on which to build new products. Tools play a key role as well. The ability to visualize thread-level interaction, CPU utilization and other key variables across multiple cores provides the developer with a white-box view into system operation.

QNX Software Systems and Freescale have jointly supported many development programs using multiprocessing in their products. Starting more than 10 years ago with dual MPC744x processors combined with a discrete SMP system controller to the latest QorIQ processors, QNX and Freescale have been at the forefront of multicore development.

How to Reach Us:

Home Page:
freescale.com

Power Architecture Portfolio Information:
freescale.com/power

e-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
1-800-521-6274
480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447
303-675-2140
Fax: 303-675 2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™

For more information, visit freescale.com/power

Document Number: PWRARBYNDBITSRAS REV 0

BUILT ON Power™