Freescale Semiconductor
White Paper

Document Number: P4080VRTASTWP
Rev 0, 08/2009

Hardware and Software Assists in Virtualization

Until recently, embedded systems were often built with computing blocks containing a single CPU. Each block included a processor, memory controller and I/O bridge. This model has been a long-lived one. As applications demanded additional performance, Moore's Law delivered ever faster and more sophisticated processors and bridge silicon. In return, power dissipation grew as clock-rates and silicon leakage increased with each generation of silicon technology.

Recently, an industry-wide consensus has emerged that physics no longer allows increasing clock rates within practical power envelopes. To deliver increasing performance, a similar consensus has arisen on the value of offering multiple processing cores at limited clock rates in place of single cores running at significantly higher clock rates. A multicore System-on-Chip (SoC) offers theoretical performance increases within practical power envelopes.

When attempting to capture this theoretical multiprocessor speedup, the first approach that comes to mind is leveraging implicit instruction level parallelism. The approach has been the subject of research and investment for years—with parallelizing compilers one of the most obvious outcomes.

Contents

Fine-grained parallelization has achieved limited success in some application spaces. However, it has been realized that embedded system architectures—particularly those in the telecom, networking and industrial space - are already naturally partitioned in at least three ways: data, control and management plane. This separation has driven strong interest amongst OEMs to map the entire system to a multicore SoC in a manner that simply integrates the previously stand-alone functions onto a single device.

This “consolidation” approach has many practical advantages including minimizing the impact to software architecture. Once an entire system solution has been moved to a single multicore device, optimizations to leverage course-grained parallelism can be applied gradually over time.

Consolidation of multiple system components into a single multicore SoC was seen first in the PC server space. With early attention from major PC silicon vendors, multicore devices coupled with compact blade mechanicals moved large server functionality into high density form factors.

Fully leveraging the potential performance of recent multicore SoCs requires new hardware capabilities and software components. The most important of these enhancements supports partitioning and virtualization of various I/O devices in the system and improves performance and minimizes redundant hardware.

1 Partitions

Multicore SoCs in the embedded space typically include a wide variety of hardware resources including processors, memory controllers, application accelerators and external I/O interfaces. Partitioning is the act of grouping these resources together in support of various application requirements as shown in Figure 1. Hardware is allocated to partitions as a shared or independent resource. Each partition contains one or more processors. A single software program or Operating System (OS) runs in each partition. Across partitions, different software environments are often present as previously diverse discrete implementations are aggregated.

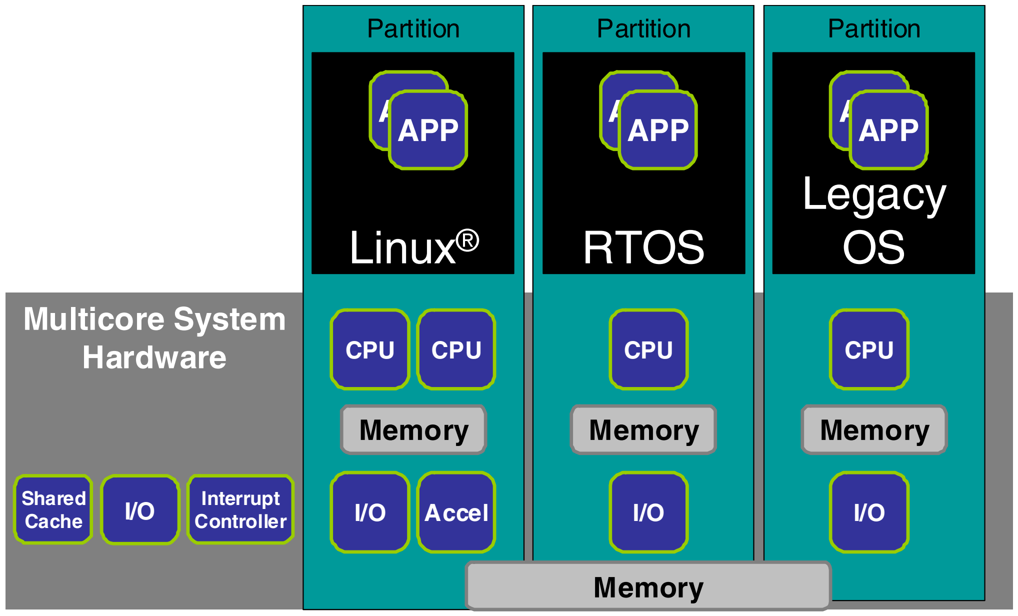

Figure 1. Multicore Partitioning

A partition is more than just a group of resources available to applications. It also implies a level of separation and protection that ensures software and hardware in one partition cannot interfere with the operation of others. Each partition operates independently. The platform should also have the ability to initialize and restart software in one partition while another continues to operate normally.

Consolidation of separate sub-systems into a single robust system depends on hardware-based protection capabilities to ensure failures are contained as well or better than prior discrete implementations.

With restart and protection capabilities, additional use models are possible, such as the ability to perform practical in-service upgrades of individual partitions from a 3rd-party or in-house OS to an open standard such as Linux®.

Protection allows isolation of non-trusted operating systems and applications (for example, sandboxing Linux from a proprietary OS). This allows, for example, software under test to be operated in a controlled fashion without impeding the operation of other partitions.

Hardware partitioning is a necessary but not sufficient capability when building a secure platform. Running trusted code in a separate secure partition where keys, rule definitions, access right control and other system-level security functions reside increases the degree of trust and protection the system can offer.

2 Multicore Use Models

When consolidating multiple systems into a multicore device, many possible mappings exist between software and available processors and I/O devices.

Multiprocessor systems with cache-coherent interconnects may operate with a single OS image running on all available processors in a symmetric multiprocessing system (SMP). In this model, all processors and I/O devices are located in a single logical partition and all caches in the system are coherent with each other. All applications run together on this single OS. Application threads are allocated by the OS to available processor cores. While common in PC server applications, this model is less frequent in embedded systems.

By contrast, embedded systems and applications are often diverse and asymmetric in nature. They tend to be separate applications and functions all operating cooperatively. This structure leads to an asymmetric multiprocessing (AMP) model where multiple logical partitions host different OS environments all operating cooperatively. In this model, processors and other hardware resources are assigned to an OS as needed.

3 Virtualization

Virtualization in a computing context is the process of emulating (or abstracting) computing resources. Using virtualization, one can define a logical partition to represent a collection of actual or emulated hardware resources. A single virtual computing machine runs on a single logical partition. Within each logical partition is found shared or private memory, one or more processors and a collection of accelerators and I/O devices. Each of these resources, including the processors, may be actual dedicated hardware or may be implemented by partially or completely virtualized hardware.

A hypervisor is a low-level software program that presents a virtual machine within a logical partition to a Guest OS. Usually, a hypervisor runs across the entire multicore SoC device. It is responsible for creating and managing one or more available logical partitions.

Logical partitions allow more partitions to exist than actual hardware would otherwise allow and enable a variety of useful system mappings between system hardware and logical partitions. At the extreme, a logical partition may be completely virtualized with no actual dedicated hardware. In this case, the hypervisor manages sharing of hardware resources across logical partitions and virtualizes hardware that cannot be otherwise dedicated to a partition.

In the server space, ensuring that expensive and power-hungry processors are fully utilized has motivated the use of this kind of virtualization to allow multiple independent OS instances to run on the same core with associated hardware resources. In this model, each OS runs in its own logical partition and there are often many more logical partitions than actual hardware resources. Logical partitions are interleaved on the processor by the hypervisor. Achieving higher aggregate compute throughput depends on the periodic availability of idle cycles during normal operation of any given OS instance and the hypervisor efficiently virtualizing machine resources.

In the embedded space, a single processor without virtualization supports a single partition and OS only. With virtualization, multiple logical partitions allow multiple OS instances to run at the same time. Besides increasing aggregate performance as described above, virtualization can also add attractive security, protection and partition management features to this simple system. For example, sandboxing a less trusted OS or an externally visible management interface web-server could substantially improve system robustness.

Not all applications benefit from multiple logical partitions running on the same processor. Real-time or other performance constraints can impose additional complications when multiple OS instances are allowed to run on a single processor. For example, interrupts in this case must pass through the hypervisor to be redirected to a logical partition and may remain pending until the partition is again given processor time.

4 Hypervisors

Hypervisors are the glue between the available hardware and the logical partitions on which operating systems run. They expose hardware when available and virtualize when necessary.

The balance between the hypervisor hardware and software-based virtualization defines the performance and efficiency achievable by the system. In general, an entirely software-based virtual machine within a logical partition is possible but not generally very fast. Examples range from the extreme of a PC running a gate-level simulation of a processor system which boots an operating system to a software emulation of a historical computer architecture. More conventional examples include products such as VMware® Server or VirtualBox that can operate without specific hardware support on the host.

Opposite in extreme to a software-only virtual machine would be a system built with an unlimited transistor and power budget. Here each processor in the device would have its own dedicated set of accelerators and I/O devices.

Actual implementations choose a point between the two extremes, sometimes applying clever solutions that allow the majority of a resource to be shared without replicating it multiple times.

Hypervisors generally manage processor Memory Management Units (MMU), I/O MMUs and system-wide memory and I/O map configuration and management. These mechanisms are the basis of defining logical partitions within a system and enforcing their boundaries.

A key performance point between hardware and Guest OS instances are exceptions together with interrupts from internal and external sources. The hypervisor must ensure interrupts are routed to the appropriate destination. Sometimes they are taken by the hypervisor in its role of virtualizing a resource and at other times they are redirected to a Guest OS in a logical partition. For highest performance, interrupts should be directly routed to a logical partition.

Hypervisors often virtualize certain low-performance peripherals that are not commonly replicated in hardware. These include timers and serial ports for debug.

There are two classes of hypervisors depending on the nature of the virtualization done: “Full” and “Para” virtualization.

A hypervisor that provides full virtualization presents to a Guest OS a logical partition identical to the actual hardware. In this case, the Guest OS is completely unaware it is running in a virtualized partition managed by a hypervisor. Most services are provided by hardware but some may still be provided by software. The mix as usual dictates performance. The key value proposition is that OS and applications require no changes to run in this environment.

A hypervisor that provides para-virtualization presents to a Guest OS a logical partition that is not identical to the actual hardware. In this case, the OS is aware it is running on a hypervisor and explicitly calls it for services. Obviously, this approach requires modifications to the operating system to run with the hypervisor. By calling the hypervisor directly, services such as interrupts and translation lookaside buffer (TLB) management can potentially be provided at much higher performance since emulation is avoided.

Hypervisors must follow Guest OS memory management requests over each logical partition in the system. More specifically manipulation of all TLBs must be followed to ensure that the memory pages are allowed to be mapped for the partition in which the processors reside. There are many possible implementations of this capability in a hypervisor driven by performance expected by applications and available hardware assistance.

Note that a system implementation can be imagined where partitioning and available hardware is such that there is no need for hypervisor virtualization of hardware resources. In this situation, hypervisors remain important in that they still enforce strong logical partitioning.

5 Real Embedded System Implementation

Discussing hardware and software virtualization assists requires the context of a real implementation. Figure 2 shows a block diagram of the Freescale QorIQ P4080 multicore SoC device. This device delivers a multitude of partitionable hardware resources such as eight 1.5 GHz processors, dual DDR memory controller and a wide variety of hardware acceleration and I/O capabilities. In addition, Freescale is making available to customers an Embedded Hypervisor that represents one implementation of virtualization technology discussed here.

The remainder of this paper discusses a variety of hardware and software assist features available on the P4080 using the Freescale Embedded Hypervisor.

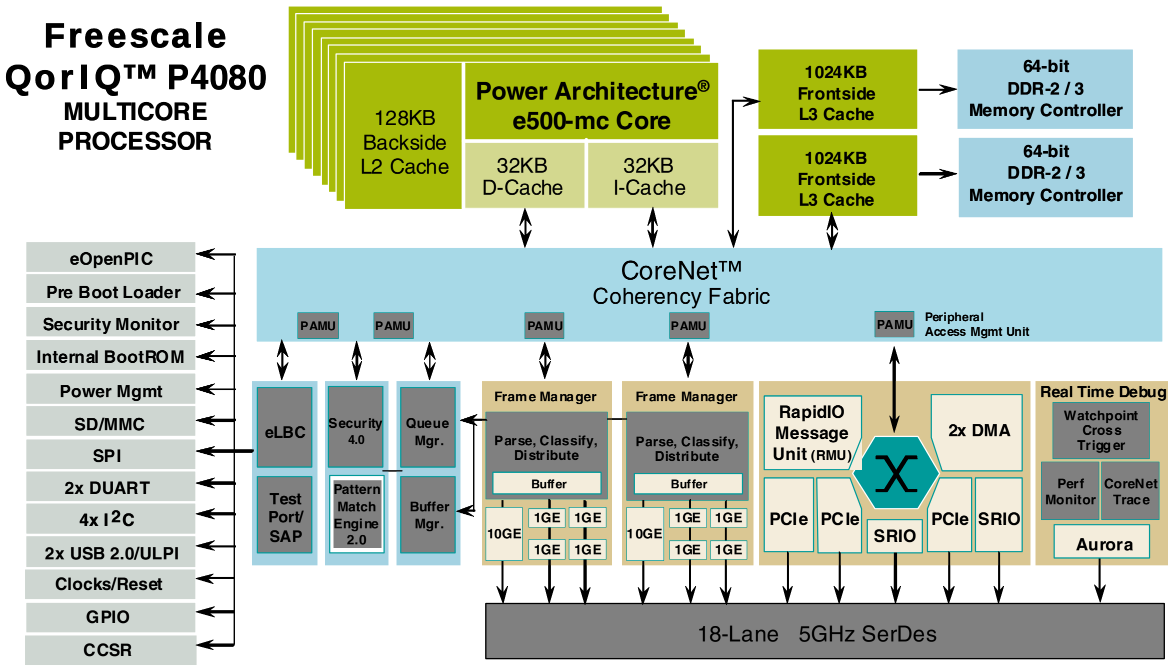

Figure 2. P4080 Multicore SoC

5.1 Hardware Assists

Moving from single to multicore devices imposes new architectural constraints on designers. For example, a high incentive exists to optimize CPU cores for area since multiple of them are integrated at once. This incentive competes with the need to add hardware support for partitioning, virtualization and hypervisor functions. One advantage of virtualization techniques is that there are mechanisms to allow features over time to move between actual hardware implementation and emulation by a hypervisor. The P4080 SoC platform, e500-mc processor core and virtualization feature set were carefully optimized to run target embedded application applications well.

Another example of the practical issues imposed by multicore is the need to enable efficient scaling when integrating additional cores. Earlier shared-bus architectures must transition to non-block switch fabrics to appropriately scale. The P4080 introduces the CoreNet™ switch fabric which allows concurrency across the fabric. In addition, it supports flexible partitioning to meet the needs of a variety of customer system architectures. Partitioning in this context means not just supporting the various chip-wide address protection mechanisms but also the ability to define subsets of cores that operate with cache-coherency.

5.1.1 New Modes and States

To enable virtualization in general and hypervisors specifically, one important processor enhancement is with regard to processor modes and states. Many processor architectures define a user and supervisor state. The OS kernel often runs in supervisor mode giving it access to privileged instructions and states, such as the MMU, while applications run in user mode. The hardware efficiently implements transitions between applications and kernel in support of a variety of services such as I/O device drivers, virtual memory management and other system calls provided by the kernel.

To support hypervisors, a third mode is frequently introduced “above” supervisor to run the hypervisor. In a manner analogous to the user-kernel transitions, hardware supports transitions between user or supervisor and “hypervisor” state. Hypervisor mode gives universal access to the hardware state of the device—including instructions and states unavailable to code running in the underlying supervisor or user mode.

In the case of the P4080, the new e500-mc core is enhanced to support a four state mechanism by introducing a new “guest” mode bit in addition to the existing “privileged” state bit. Together they define the mode and state in which code is running. When guest mode is enabled, the system can exist in either supervisor or user state thus allowing both OS and applications to run entirely in this guest mode. When not in guest mode, the processor is in “hypervisor” or “bare-metal” mode and can again be in either supervisor or user state. With two states in the hypervisor mode, the processor allows operating systems running directly on the hardware to operate in hypervisor state (hence the term “bare-metal” mode). When a hypervisor is present, it operates in the privileged state in hypervisor mode and thus runs directly on the hardware.

There are a number of additions made in support of embedded hypervisors. Many of them manage routing of interrupts. Proper management depends on the mode and state of the processor at the time of the interrupt.

A new set of frequently referenced registers have been defined while executing in Guest mode. Some examples include:

- Guest Save/Restore registers (GSRR0/1)
- Guest Interrupt Vector Prefix Register (GIVPR)
- Guest Interrupt Vector Registers (GIVORn)
- Guest Data Exception Address Register (GDEAR)
- Guest Exception Syndrome Register (GESR)
- Guest Special Purpose Registers (GSPRG0..3)
- Guest External Proxy (GEPR)
- Guest Processor ID Register (GPIR)

Most of these registers provide states important to servicing exceptions and interrupts and duplicate those available in hypervisor mode (or prior Power Architecture® implementations without guest mode). These registers have different offsets from the originals. To ensure legacy operating systems can run unchanged while in guest mode, references to the original non-guest versions are mapped to the guest versions above.

5.1.2 Exceptions and Interrupts

Key to high performance in virtualized systems is the manner in which interrupts are handled in the system. There are several choices in how to manage these events:

- Direct all exceptions and interrupts to the hypervisor which directs them to the appropriate Guest OS

- Direct most interrupts to the hypervisor but route performance critical interrupts to the appropriate Guest OS
- Keep interrupts that occur during execution within the same logical partition and mode in which they occur

In general, most interrupts and exceptions are directed to the hypervisor. The task of the hypervisor is to promptly re-direct or “reflect” the interrupt or exception to the appropriate logical partition and related processor(s). The overhead and latency imposed in this process must be carefully weighed with system level concerns.

The e500-mc processors and P4080 allow significant flexibility in how interrupts are reflected and where they are serviced. One example is the ability of the programmable interrupt controller (PIC) to route an external interrupt directly to a physical processor within a selected logical partition without having to pass through the hypervisor. This is intended for performance critical interrupts usually associated with hardware dedicated to that partition. An example might be the direct routing of PCI Express® interface interrupts to a partition tasked to manage that interface. Directly routing an external interrupt to the resource servicing it can substantially reduce latency since it avoids the redirection.

When an external interrupt is directed to a Guest OS and the interrupt occurs while executing in hypervisor mode, the interrupt remains pending until the processor transitions back to Guest mode. Acknowledgment of these interrupts occurs automatically by the processor hardware.

The e500-mc processor introduces some additional exceptions and interrupts to support the Embedded Hypervisor. To allow a para-virtualized OS to call the hypervisor, a special system call exists to cause the transition between guest and hypervisor mode. It uses the existing System Call instruction but with a different operand.

When a Guest OS tries to access a hypervisor privileged resource such as TLB or other protected resources within the processor, the exception always transitions execution to hypervisor mode and allows the hypervisor to check protections and virtualize it when necessary. In the case of cache access, normal cache load/store references proceed normally. For direct cache operations, a mode exists to allow Guests to perform some operations such as locking the cache if desired.

A new interrupt has been introduced for processor-to-processor doorbell interrupts. Besides being a general “shoulder-tap” mechanism for the system, these interrupts may be used by the hypervisor to reflect asynchronous interrupts to a Guest OS. The complication of reflecting an asynchronous interrupt to a Guest is that it cannot be reflected until the Guest is ready to accept it (i.e. it has been enabled). When an asynchronous interrupt occurs, the hypervisor uses the **msgsnd** instruction to send a Doorbell interrupt to the appropriate Guest. These exceptions remain pending until the Guest has set the appropriate enable bit. Once this exception occurs, it is immediately directed to the hypervisor which can now reflect the original interrupt knowing the interrupt has not been enabled.

5.1.3 Memory Management Assists

One important role of hypervisors in virtualization is managing and protecting logical partitions. It must ensure that a logical partition only has access to its allocated resources and must block unauthorized accesses.

To perform this service, the hypervisor directly manages the MMU of all processors and in many implementations also manages a similar hardware resource called the I/O MMU for all I/O devices which master transactions within the SoC.

In the case of the MMU, there are several possible approaches to management depending on whether the hypervisor is hosting a fully or para-virtualized OS.

5.1.3.1 Fully Virtualized Guest OS

In this case, the hypervisor has several choices. It can choose to reflect all memory management interrupts to the Guest OS and then capture and emulate all TLB manipulation instructions from the Guest.

In another possible approach, the hypervisor captures all TLB manipulations by the Guest in order to emulate them. Meanwhile, it builds a shadow page table. When a TLB Error or Miss occurs, it references its shadow page table in search of a relevant entry. If one is found, it manipulates the TLB directly and returns from the TLB miss exception without causing an exception to the Guest. If no translation is available, it reflects the miss to the Guest, emulates the TLB operations and updates its shadow page table. Whenever the information is in the shadow page table, this scheme eliminates the need for the hypervisor to emulate the miss-related TLB instructions on behalf of the Guest.

In both approaches, the Guest OS is unaware that the hypervisor is emulating the instructions. The key to this working is the need for TLB manipulation instructions to cause traps to hypervisor mode where emulation can occur. In the case of the e500-mc, this is indeed the truth.

5.1.3.2 Para-virtualized Guest OS

When modifications to the Guest OS can occur, it is possible for the OS to directly call the hypervisor for memory management services. In this approach, page tables could be given to the hypervisor, memory management exceptions are taken by the hypervisor and necessary TLB updates are performed by the hypervisor. This approach may offer the best performance since emulation of TLB instructions is not required.

5.1.3.3 Virtual and Real Addresses

To allow the TLB to simultaneously carry entries for more than one logical partition, the virtual address within the TLB must be extended with a field designating the logical partition. Furthermore, to allow the hypervisor itself to have entries in the TLB, a bit of virtual address differentiating entries for guest and hypervisor mode is also required.

On the e500-mc, the TLB virtual address is extended by an LPID value that is a unique identifier for a logical partition. The address is further extended with the GS bit that designates whether the entry is valid for when in Guest or hypervisor mode.

Because all e500-mc TLB instructions that expose physical mappings to the Guest OS are hypervisor privileged and hence emulated, it becomes possible for the hypervisor to shield the actual physical address of the TLB from the Guest. This allows the hypervisor to remap Guest OS “logical” physical addresses to a new “real” physical address. This may be useful, for example, when the hypervisor must allow multiple logical partitions to all believe their physical address starts at 0.

5.1.3.4 I/O MMU

One important hardware assist in support of secure logical partitions is the existence of I/O MMUs that control access by peripherals capable of initiating transactions to memory-mapped resources. Examples of such peripherals might be any device with a DMA engine or I/O interfaces that accept externally mastered transactions.

Simply put, an I/O MMU examines all address-based transactions originating from mastering devices and authorizes the transactions to proceed. It does so by keeping internal state that relates the originator of the transaction with authorized address regions and associated actions.

The P4080 implements its PAMU (I/O MMU) as a distributed entity throughout the SoC wherever transaction initiators exist. In addition to the role of controlling access by I/O devices, the PAMU also selectively allows transactions within certain address ranges to be written directly into the cache.

The hypervisor initializes and manages the I/O MMU to define and protect logical partitions in the system.

5.1.4 Hardware Resource Allocation and Sharing

How hardware resources on a multicore SoC are allocated and possibly shared across logical partitions is one of the most important performance attributes of the system. Dedication of a resource to a single partition may simplify software. However, in most systems this limits parallelism and sharing. For example, by dedicating an Ethernet controller to a single partition, no other partitions can share the processing of inbound packets or leverage unused outbound bandwidth unless software explicitly manages ownership—often using software semaphores.

A wide variety of internal accelerators and I/O devices can exist in a multicore SoC. Many devices are optimized for particular applications by implementing application specific functions. The P4080 implements encryption and pattern matching engines that support a variety of networking functions. The ability of multiple partitions to share these resources without explicitly passing ownership is an important feature of the P4080. Sharing is facilitated by a hardware resource that provides multiple dedicated hardware portals used to manage buffers and submit and retrieve work via queues. By dedicating a portal to a logical partition, the partition interacts with the encryption and pattern matching engine as if it was the only device using it.

External interfaces can also be shared through these hardware-based mechanisms. In the P4080, the Ethernet network interfaces participate in this same hardware queue and buffer management facility. It can classify inbound packets to a queue associated with a specific logical partition. Hardware then places the packet in a queue owned by that partition and accessed through its dedicated portal. The process is similar for egress where each logical partition can submit packets for transmission outbound using its private portal.

Besides hardware portals, another way to share hardware is to use the hypervisor to virtualize the resource. In the P4080, the Guest OS driver could use register emulation or a hypercall to the hypervisor's global driver which would then manage sharing amongst all logical partitions. Some examples of virtualized I/O provided by the P4080 Embedded Hypervisor include the PIC, I^{2}C interface, General Purpose I/O pins and Byte Channels as described below.

Some external interfaces can be challenging to share. This is especially true for address-mapped interfaces such as PCI Express. In this case, the lack of a standard channelized logical layer makes it difficult to decide how to direct inbound transactions to a particular logical partition. As a result, these interfaces are typically dedicated to one logical partition and all other partitions that wish to use the interface must do so through the dedicated partition.

NOTE

The PCISIG has completed work on the PC Server-focused I/O Virtualization (IOV) for PCI Express. This optional extension defines a mechanism for multiple logical partitions to share a PCI Express controller by creating associations between logical partitions and external PCI Express endpoints so traffic can be properly routed. While of interest to the PC Server market, these extensions are unlikely to be extensively supported in the near term by embedded silicon vendors.

The P4080 provides three PCI Express controllers, each of which the hypervisor allows to be dedicated to different logical partitions if desired.

5.1.5 Secure Boot and Platform Assurance

Modern embedded systems face new threats to their robustness from physical and remote hacking. Unauthorized exposure of internal device secrets such as sensitive software code or encryption keys can be catastrophic for customers and OEMs alike.

Multicore SoCs with robust partitioning support in hardware can leverage this capability as managed by a hypervisor to ensure the system is securely booted with trusted code and that the platform remains robust and secure during normal operation.

The P4080 implements such a facility. A trusted hypervisor together with MMU and I/O MMU facilities provide a strong foundation upon which these capabilities can be built. Additional hardware support during boot time assures that boot and runtime code is trusted before execution and prevents unauthorized debug access to secure state. For the P4080, virtualization provides key hooks leading to secure boot and platform assurance features. Once protected logical partitions are established, system designers can sandbox software applications and environments that are not trusted by leveraging all the mechanisms built to virtualize the hardware.

5.2 Software Assists

A wide array of capabilities can be implemented as software assists for virtualization. The assists provided by Freescale’s Embedded Hypervisor are used as an example of some of the capabilities tailored specifically for embedded systems. Figure 3 shows graphically the services and capabilities of this hypervisor. Obviously, this example does not address all possible assists but should suffice as a primer on what is possible.

The Embedded Hypervisor for the P4080 is designed to be a light-weight hypervisor and is based on the Power Architecture as documented in the PowerISA 2.06. To keep the scale manageable, the initial version focuses on static partitioning fixed at boot time or until a reconfigure and reboot is performed. In addition, the hypervisor does not initially support multiple logical partitions running on a single processor.

The hypervisor uses a combination of full and para-virtualization to enable high performance and minimal changes to the Guest OS. The hypervisor emulates privileged instructions such as TLB operations, provides a variety of hypercall-based services and uses a device tree structure at initialization for resource management.

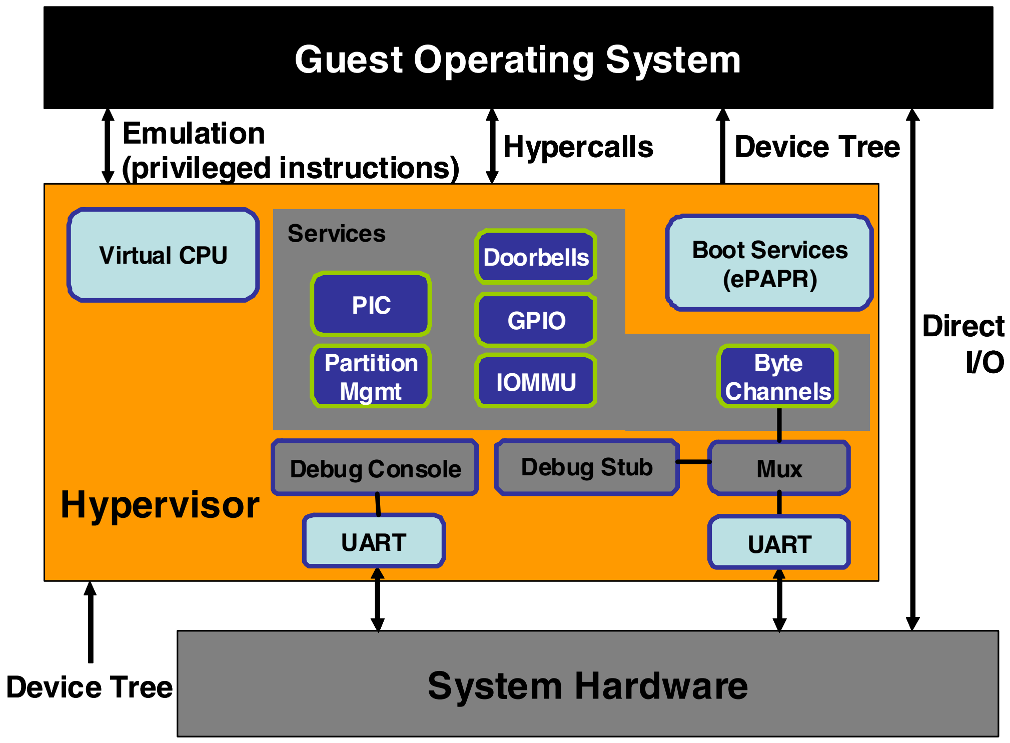

Figure 3. Freescale Embedded Hypervisor

It presents to the Guest OS a virtual processor that is very similar to the e500-mc processor but without the hypervisor extensions. Some differences include fewer MMU entries (to reserve some for the Hypervisor) and subtle differences in the watchdog timer and timebase (guest writes to the timebase registers are ignored). At the virtual machine level, there are PIC driver changes as emulated by the hypervisor.

A variety of hypercall services are also provided:

- Interrupt controller (MPIC)
- Byte-channels
- Inter-partition signaling
- Partition management including
  - Start/stop and image load initially and create/destroy later
  - Partition management interrupts
- Power management for processor cores to change clock frequency and power state
- Partitioning of GPIO pins
- I/O MMU management
  - Create/destroy mappings

Byte channels are a character I/O channel that flexibly connect partitions and the physical UART on the P4080. It includes support for a hypervisor console, byte-channel multiplexors and debug stubs. Through the use of an external byte-channel mux server on a PC, individual character I/O streams can be created to each logical partition in the system.

Using the same mechanism by which external interrupts are reflected to an appropriate Guest OS, a general inter-partition signaling facility is supported by the Embedded Hypervisor. One OS can signal others in a unicast or multi-cast fashion. The facility is intended to be a lightweight “shoulder tap” facility which passes a very limited amount of state.

Partition management services allow a partition to copy data to and from another partition—useful when loading OS images. It also supports the ability to start other partitions and stop or reboot them. Notification of events are also supported including events such as a watchdog timer expiration and Guest OS commanded reboots and errors.

Many of these boot facilities are standardized by the Power.org Embedded Power Architecture Platform Requirements specification (ePAPR). In addition, ePAPR defines a device tree that is a data structure that represents a logical partition’s hardware and virtual hardware resources.

6 Conclusion

Multicore SoCs such as the P4080 are driving new levels of virtualization capabilities into embedded systems. Aggregating formally separate standalone computing resources into a single multicore device presents great opportunities and some challenges. Hardware and software virtualization support is crucial to fully utilizing the capabilities of these complex devices.

Because the embedded world has always differed substantially in requirements from the PC Server space, it is clear that hardware and software trade-offs need to be very different thus resulting in features and performance better aligned to this market space.

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support

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)
www.freescale.com/support

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

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

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

Document Number: P4080VRTASTWP
Rev 0
08/2009