

White Paper
Gerald Rogers
Benchmarking Architect
Edwin Verplanke
Platform Architect
Intel Corporation

Intel[®] Architecture and Homogeneous Multi-Core Computing in Embedded and Communications Platforms

January 2009



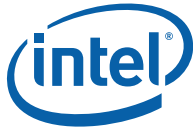
Executive Summary

Symmetric multi processing is without a doubt the most convenient and preferred method to take advantage of multi processor computer platforms. However, the Embedded and Communications usage models are considerably different from mainstream usage models, for that reason this paper proposes an alternative: homogeneous multi-core computing. Embedded designers could consolidate single threaded code and maintain performance. Often communications appliances integrate real time proprietary operating systems that are single threaded, what if designers could consolidate these kinds of operating systems and maintain determinism without the need to migrate to a symmetric multi-processor environment? This paper will cover what it takes to configure an Intel® architecture platform to perform homogeneous multi-core computing.



Contents

Introduction.....	4
Platform Requirements	4
Trust	4
Interprocess Communication	5
Relocation.....	5
Methodology	5
Boot Strap	5
Device Management	9
Interprocessor Communications	10
Conclusion	13
References.....	14



Introduction

The core count that will be integrated on a single die will continue to increase. Mostly software developers will take advantage of multi-core or multi-processing platforms through Symmetric Multi Processing (SMP) operating systems. Developers have the availability over a large set of tools and libraries that help parallelizing the application. These tools include Intel® Thread Checker, Intel® Thread Profiler, VTune™, etc. There are popular threading libraries such as Pthreads* and WinThreads* that help with scaling the application across multiple cores. However embedded developers do not always have these choices. In some cases the host operating system is proprietary, single threaded and/or real time. Rewriting these operating systems to take advantage of multi-core is non-trivial and could be a costly exercise. In addition the applications would have to be modified and all together this introduces risks and potentially decreases the reliability of mission critical platforms.

You could ask the question “Why not utilize virtual machine monitor and run several instances of the single threaded proprietary operating system?” Real time and performance requirements could prohibit the usage of this multi-core migration method. Another reason could be the potential reliability risk associated with the introduction of the virtual machine monitor. In most cases the virtual machine monitor vendor does not cater for these requirements. There are a limited number of VMM suppliers that supply a solution, though that may add a significant cost to the platform.

Platform Requirements

This paragraph describes the requirements that are involved with configuring a multi-core Intel® architecture platform for homogeneous multi core computing. The term *homogeneous multi-core computing* was carefully chosen, the Intel® architecture CPU's that we are referring too have multiple cores but are identical. The multi-core CPU may run one operating system image per core.

Trust

Trust is the key for a homogeneous multi-core processing system. Each independent system must ensure that it not interfere with other systems that coexist with it. Today hardware is designed with the criteria of a single system manager (operating system in most cases, but occasionally it is a simple executive that consists of one main loop). Therefore, it does not natively support multiple managers. Even with the usage of Intel®



Virtualization Technology, it is necessary that there exist a manager (in this case a virtual machine manager or system manager). The lack of native support for multiple managers does not preclude one from achieving such usage. To accomplish this on single manager hardware, it becomes critical that a few simple rules be followed:

- A system manager must remain within its own memory range.
- A system manager must be assigned to its own unique core.
- A system manager must be assigned its own set of system devices and not use any others.
- A system manager must know which hot plug devices belong to it.
- A symmetric multiprocessing system manager must restrict its cores to those it is assigned.

Interprocess Communication

In addition to trust, there must be a mechanism for communication between system managers. There may be instances where resource usage may be shared by two or more managers, thus requiring a sharing mechanism. Many mechanisms exist for this to happen. Semaphores can be used for sharing resources between two system managers. Memory rings can be used to share data between two system managers. TCP/IP can be used to share data between different system managers. Shared mailboxes could be another mechanism to ensure communication between system managers. These and other mechanisms can provide a way to ensure the sharing of resources and memory between the various system managers.

Relocation

A system manager must be capable of execution from any region of memory. In essence a system manager needs to be aware either at compile time or run time of where it has been located into memory. It is necessary in a heterogeneous system to relocate a system manager to any location in memory, and place a top on what memory it may access.

Methodology

Boot Strap

Bootstrapping is the first step in getting the separate CPU's to run independently. There are two primary methods of bootstrapping. In the first implementation one of the system managers (host operating system or simple executive) could bootstrap the other CPU's in the platform. This



bootstrap method is described in this example: [ETA: Experience with an Intel® Xeon™ Processor as a Packet Processing Engine](#).

A second possible implementation is to use a bootstrap agent. This agent would load on the boot strap processor, and then initiate a boot message to the application processors to boot each system manager.

The next two sections will describe each of the methods in further detail.

1. Host Operating System Boot Strapping

In the host operating system bootstrapping method, you would use the host operating system to boot strap all the other CPU's. This can be accomplished utilizing Linux and kernel threads. Each system manager is compiled as a kernel module which spawns a kernel thread to execute the system manager's code. The host operating system uses processor affinity to assign the system manager (launched through the kernel thread) to a given core. The Linux* kernel scheduler is then restricted from executing code on this core. The kernel thread and associated system manager code, allocates a large heap of memory from the Linux kernel heap, and then manages that heap for its own. The system manager kernel thread is moved to the highest priority, and is always scheduled with minimal delay onto its assigned core.

Steps:

1. Create kernel module with proper kernel wrappers.
2. Put into the main routine of the kernel module a start of a kernel thread that starts execution of your real time operating system or the simple executive.
3. Modify a call from your module to set kernel thread to highest priority.
4. Linux* kernel needs to be slightly modified to restrict the execution of programs on this core. This is done by patching the SMP code in /kernel. Two methodologies are to put into the kernel a routine that can be called by your module, or by statically restricting the Linux kernel.
5. To control devices, you will need to register an interrupt routine with the Linux kernel to handle that device. If you want to override the default ISR routines of Linux a special routine is needed.
6. If desired, you will need to put in some form of communication between your kernel module and Linux. The mechanism could consist of using shared memory or a message box.
7. If your system manager is multi-tasking, then it would need to allocate a timer. This may be done through Linux, introducing extra processing overhead to manage the timer, or your system manager could register the ISR to service the Timer.



8. Communication between Linux and the system manager can use any of the methods mentioned in the Interprocess Communication. If the shared memory method for communication is used, Linux would be restricted to not use the region allocated to your system manager and the shared memory region, you will have to use the mmap Linux routines to access them. The mmap routines allow mapping physical memory into the Linux domain, even if Linux doesn't have direct access to that memory. Please refer to the Linux documentation for further explanation.

2. Boot Strap Agent

Bootstrapping with a boot agent is somewhat more complicated than previous approach. In this case there is no host operating system to spawn the other operating systems or simple executives. In addition there will be no interaction and no scheduling overhead associated with the host operating system.

Bootstrapping from a boot agent can be accomplished in two ways, either from the BIOS EFI shell or from a bootstrap agent such as Grub, Redboot* or other open source boot loader. From here, a special routine is required to setup a jump point in memory, load the source code into memory, and then initiate the application processor to move into protected mode and start executing at the jump point.

Intel® architecture supplies various mechanisms for managing and improving the performance of multi-core processors. One of these key components is the local APIC (Advanced Programmable Interrupt Controller). Each CPU in a multi-core processor contains a local APIC and has two primary functions:

- It receives interrupts from the processor's interrupt pins, from internal sources and from an external I/O APIC (or other external interrupt controller). It sends these to the processor core for handling.
- In multiple processor (MP) systems or multi-core CPU's, it sends and receives interprocessor interrupt (IPI) messages to and from other logical processors on the system bus. IPI messages can be used to distribute interrupts among the processors in the system or to execute system wide functions (such as, booting up processors or distributing work among a group of processors).

The local APIC is also used to initialize operating system or simple executive. The bootloader can issue these messages targeted at the local APIC, below is the startup sequence demonstrated.

OS Relocation

An operating system or application must be able to relocate itself either dynamically or during compilation to a location other than what it is used to being started. The following sections provide samples on how to do this.



Processor Mode

Due to the nature of splitting the memory space into multiple sections an application processor would not be able to operate in Real mode unless it resides below the 1 MB region. In big real mode, a real mode application could operate in up to 4 GB of memory. Protected mode applications would work within any memory region.

The bootstrap agent will need to move to protected mode or big real (if memory is less than 4GB) mode in order for it to load an application into higher memory. Since it will be operating strictly on real memory, the Global Descriptor Table and (GDT) and Local Descriptor Table (LDT) need to be setup such that they do not use virtual swapping to any drive, but use real addressing instead. Once the processor mode has been switched the bootstrap agent can now load each application and initiate them. For details on how to switch between the various processor modes please refer to the Intel® architecture software development manuals.

Now that the processor mode has been set we can load the application into memory. This is very straightforward in that you need to read from your source the application code, and write it directly into the memory location where it starts.

After loading the application code you need to write into a memory location less than 1 MB a start pointer. This start pointer would need to modify the mode of the application processor to switch to Big Real or Protected mode and then jump to the location specified by the start pointer, and begin executing the code.

- Choose a start pointer not used by Boot Strap agent example 5000:0000
- 5000:0000 : start pointer Jump Point
- 5000:0008 : Switch Processor Mode
- 5000:00xx : Jump to start pointer and begin execution
- Code Snippet to start Application Processor
- Fill in message

```
Int message = 0
int level = 4 // Interrupt Edge level defined in spec
int init = 6 // Init IPI message
int start = 5 // Start IPI message
int segment // Start pointer 4kB offset from base
memory
int offset // Start pointer offset from 4kB block
start
Message |= level <<32; // Set the level
Message |= init <<24; // Command is send init
*0xfe00310 = apic_id // Platform dependent
*0xfe00310 = message // Send message to init application
```



```
processor
Message = 0
Message | = level <<32;
Message | = start<<24
Message | = (segment << 4) | (offset)
*0xfee00310 = apic_id // Platform dependent
*0xfee00310 = message // Send message to start
application processor
```

At this point you are complete, and the AP processor should be executing.

Device Management

Embedded and communications platforms often have static device configuration, during its entire life cycle hardware changes are minimal. This is very advantageous from a homogenous platform configuration since programmers do not have to deal with plug and play. Before the platform is brought up the programmer will have to determine what target operating system will take ownership of a specific device.

PCI Device Assignment

Device assignment is static with the exception of hotplug devices. It is the responsibility of each system manager to know which devices on the PCI bus or USB bus are assigned for its use. In the PC environment the BIOS will walk the PCI bus, and assign appropriate memory to each device on the bus. The system manager will then be responsible for loading appropriate device drivers for each assigned device. In a hotplug configuration, each system manager would be responsible for recognizing a hotplug event, and determining if a device is to be managed. The designer would have to ensure that no two system managers would take possession of a single hotplug device. The same rules for PCI apply for USB or any other device. With the exception of PCI-Express* devices share Interrupt lines.

Timers

The Operating Systems that will be launched by the boot loader or host Operating System often have some hardware timer requirements. Most Linux distributions have support for various timer sources, these timers can be selected at kernel compile time. The small executives we referred to earlier generally do not require any timers and therefore will not have this requirement.

Intel® architecture platforms support various hardware timers sources, often the chipset has support for several high precision event timers and the Legacy Programmable Interrupt Timer (8254). In addition each CPU in the platform has a timer source, this timer is part of the local APIC.



I/O APIC

The external I/O APIC is part of Intel's system chip set. Its primary function is to receive external interrupt events from the system and its associated I/O devices and relay them to the local APIC as interrupt messages. In multi-processor or multi-core systems, the I/O APIC also provides a mechanism for distributing external interrupts to the local APICs of selected processors or groups of processors on the system bus. In a homogeneous system configuration the host operating system (method 1) or the bootloader (method 2) would have to setup the I/O APIC to make sure the interrupts of the I/O devices are assigned to the appropriate core where the target system manager or simple executive will be executing.

Interprocessor Communications

In a symmetric multi-processing operating system it is relatively simple to share data between applications, processes or threads since they all share the same memory subsystem. In a homogeneous configuration the target operating systems, system managers and/or small executives are not necessarily aware of each others existence and therefore a sharing mechanism may have to be implemented.

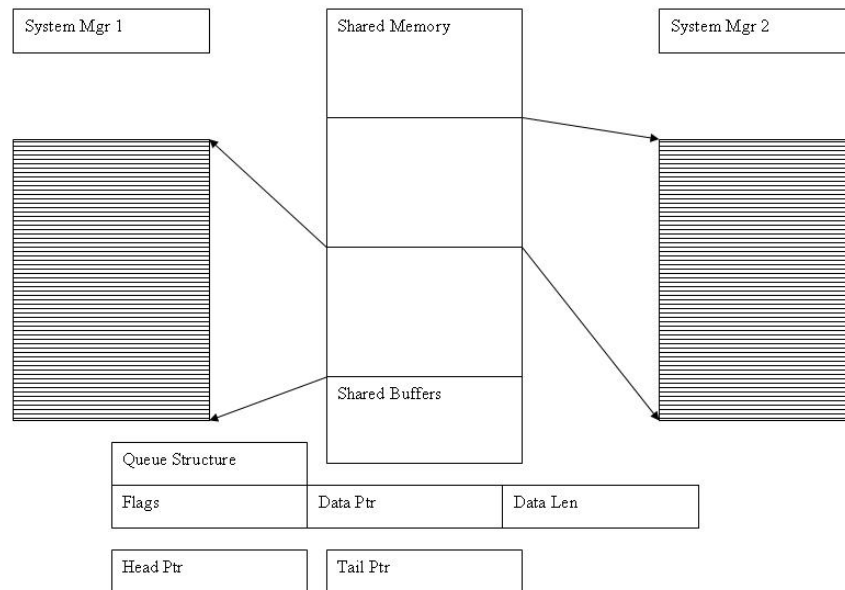
Shared memory

In a shared memory setup, two system managers will be given an overlapping region of memory in their global descriptor tables. This region would allow the two system managers to communicate with one another by reading and writing to this memory location. Various mechanisms can be used to ensure synchronization between the two system managers. You can use Atomic memory operations, Shared Semaphores or other form of locks to ensure that memory integrity is maintained. Within Intel® architecture the inherent Atomic memory operations will ensure some form of integrity across multiple processing cores.

Interprocess Queueing

Queues are a mechanism of creating communication between two separate system managers. In essence using shared memory the two systems will create bidirectional queues or rings to communicate with one another.

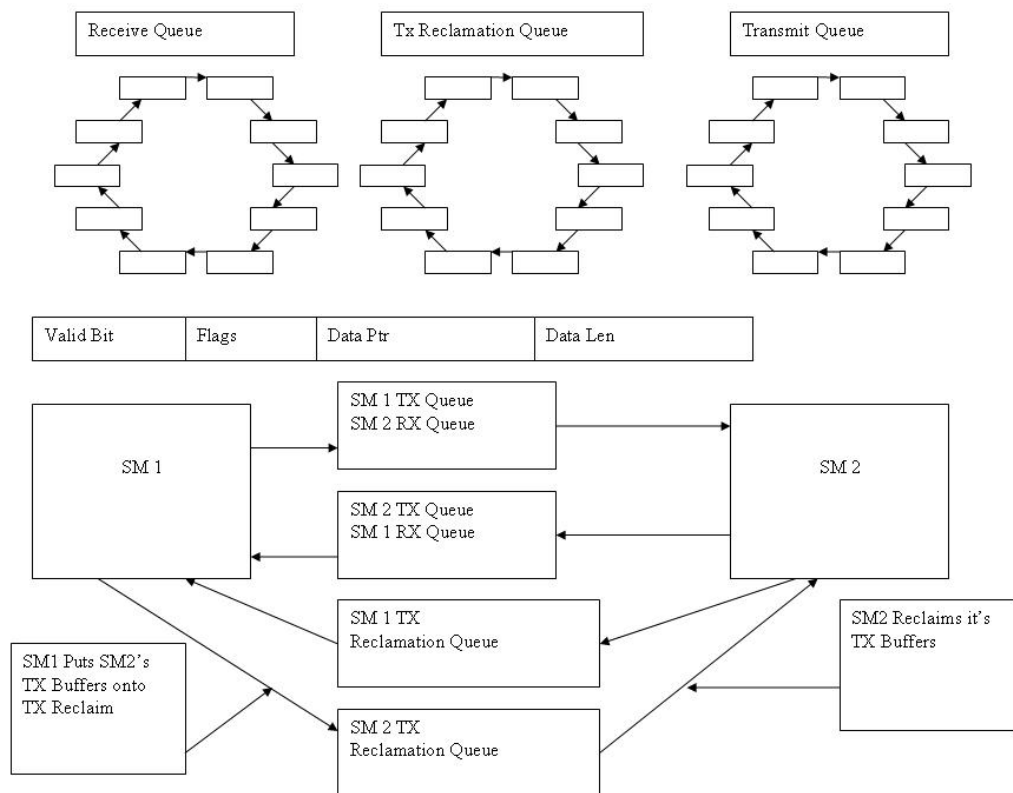
Figure 1. AMP Memory Map



[Figure 1](#) illustrates a common queue structure. If System Manager 1 wants to send a message to System Manager 2, it will update the head and tail pointers respectively. Now in order for this to work, there needs to be a lock pointer in the shared memory to ensure that the two System Managers don't update the pointers simultaneously. This approach will ensure that the two systems don't cause a corruption. In addition to the queues being in shared memory, the buffers of data will need to be in the same memory region. The queue structure introduces an overhead in manipulation of the pointers by introducing a locking mechanism. An alternative approach is illustrated in [Figure 2](#) using ring buffers that mix the locking mechanism into the data structure of the ring entries.



Figure 2. Ring Buffer Architecture



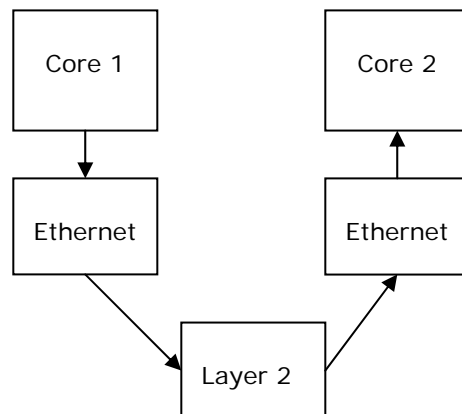
In the ring buffer setup, the buffer descriptors are pre-allocated on each System Manager. Each descriptor has a valid bit, flags, data ptr and data len. To communicate a system manager will place onto the receive queue of the other system manager, and update the valid bit. Now since only one agent will be updating the valid bit at any time, and the Atomic operations of the processors ensure consistency, there is no need for locking. What happens in this processing is that the receiving system manager will listen on the receive queue for a buffer to change its valid bit to valid from invalid. When this occurs, then it can assume that the buffer is accessible and usable for processing. This also assumes that the system manager that placed the buffer pointer onto the queue has ensured that all the pointers and values



have been synced within memory. Now once the packet on the receive queue has been processed, then the buffer pointer will be placed back onto the TX Reclamation buffer of the System Manager 1 transmitting the packet. This mechanism is very common in the embedded and communications platforms, and offers an efficient mechanism for synchronizing communication between two separate processing elements.

Ethernet

Ethernet can be used to communicate between two domains, and eliminates the shared memory overheads. The drawback to this mechanism will be increased latency since the exchanged buffers have to traverse the physical Ethernet hardware. It is possible to use the ring or queue structures to implement a pseudo physical wire, which would put the Ethernet level communication at a higher protocol level. So with this implementation, you need the setup in the following drawing.



When Core 1 transmits its packet, it will send it out the Ethernet interface. Core 2 will receive its packet on the Ethernet interface, and then operate upon the data. This mechanism eliminates the synchronization issues in the other mechanisms, but introduces a higher latency.

Conclusion

The Intel® architecture platform is optimized for symmetric multi-processing operating systems; however, with some platform reconfiguration the processor cores can operate in a homogeneous fashion where single threaded operating systems can run independently from each other.

Developers can take advantage of this multi-core migration approach as long as a set of rules are being respected. These rules make sure that the operating systems or simple executives assigned to specific cores in the platform do not have overlapping memory regions, each have exclusive device ownership and related interrupts are distributed appropriately.



References

[ETA: Experience with an Intel® Xeon™ Processor as a Packet Processing Engine](#)

IEEE Computer Society has published an Intel research paper about using an off-the-shelf processor for improved packet processing.

Document Number: [IR-TR-2004-37]

Publish Date: 1/1/2004

Primary Author: ([Lookup bios](#)): REGNIER, GREG J

Other Author(s) ([Lookup bios](#)): MINTURN, DAVE B; MCoALPINE, GARY L; SALETORE, VIKRAM A; FOONG, ANNIE

View other documents in category: [Journal Paper](#)

Intel® 64 and IA-32 Architectures Software Developer's Manuals

<http://www.intel.com/products/processor/manuals/>



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

This paper is for informational purposes only. THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino logo, Core Inside, Dialogic, FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel EP80579 Integrated Processor, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, IPLink, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skool, Sound Mark, The Journey Inside, VTune, Xeon, and Xeon Inside are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2008 Intel Corporation. All rights reserved.