EUROTECH

Optimizing Embedded Designs for the Intel® Atom™ Processor

Whitepaper

By: Jamey Dobbins

Abstract

The Intel® Atom™ Processor, with multithreading and virtualization, delivers x86 performance, minimizes power consumption and heat dissipation, eliminating the need for fans or heat-sink, and brings powerful new technology to battery operated, space constrained embedded applications. To take advantage of the all the benefits offered and migrate through numerous product options, designers must understand intricacies involved in hardware, software, and end delivery to select the right Intel® Atom™ Processor -based solution for their embedded devices.

Introduction

Compelling low power and higher performance embedded systems can now be realized by leveraging the advanced architecture and managed power characteristics of the IA-32 based Intel® Atom™ Processor. Performance computing features traditionally reserved for desktop and server class systems are now available for well optimized embedded systems and, if properly implemented, can be achieved in concert with low power operation. These features include: multithreading, Intel® Hyper-Threading Technology, virtualization, high level macro ops and advanced SSE3 math dedicated functions. In addition to these features, the Intel® Atom™ Processor can deliver sophisticated dynamic power management. This paper and presentation will outline how these performance capabilities coupled with advanced power managed system architecture can be productively employed for optimized embedded systems.

Multithreading and Intel® Hyper-Threading Technology

What is multithreading and how can it be used to optimize embedded applications? In the computer world, multithreading is the task of creating a new thread of execution within an existing process rather than starting a new process to begin a function. Essentially, the task of multithreading is intended to make wiser use of computer resources by allowing those already in use to be simultaneously utilized by a slight variant of the same process.

How does this differ from multitasking one might ask? In multitasking, a separate process is spawned or forked off of the original process and the forked process has a completely separate address space and process id. Threads on the other hand share the same address space and process id as the process that created them. Creation of threads requires less overhead, as measured in machine time, than the forking off of an entirely new process. Threads have an advantage over processes by sharing file handles. A file opened in the main process can share this handle with all of the threads it created. Communication between threads is also much easier than inter-process communication.

Parallelism

The key to optimize applications whether embedded or not is to promote parallelism which is the simultaneous processing of different data or tasks. Parallelism is achieved through two models: “data” and “functional” decomposition. As the names imply, these two models represent very different methods of applying multiple threads to achieve a higher level of performance within a single process. Data decomposition is where the same independent operation is applied to different sets of data. Functional decomposition means performing independent work on asynchronous threads.

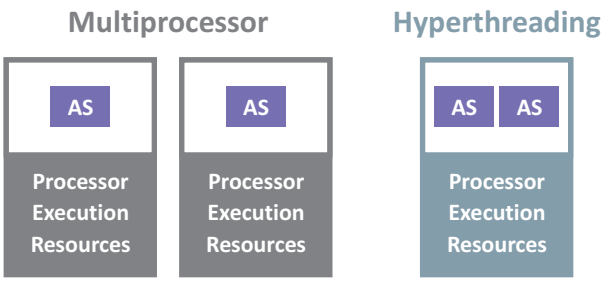
Examples of data decomposition would be mathematical algorithms like matrix multiplication that could split the row times column computation into ‘N’ threads each computing a different columns output. Applications such as a client-server communication, advanced 2D/3D graphic rendering with many characters or independent entities in the visible scene, complex HMI’s, or playing a video lend themselves to functional decomposition. In the client-server model a new thread is created for each client that communicates with the server. A graphics display application could use a different thread to control the actions of each entity or rendering layer on the screen. When executing playback of a video, threads are used to handle the audio, read data from the disk, and play the video.

Multi-core

Threads can run concurrently on multi-processor or multi-core hardware, yielding higher performance and increased system responsiveness. Multithreading is best suited for this type of system. Threads can, however, run on uni-processor cores and uni-processor machines. This is achieved by a "slight of hand trick" called time-slicing. Each thread is scheduled to run on the processor for a given amount of time called a time-slice. The operating systems scheduler is responsible for when and how long each thread of execution is allowed to run on the processor. There are different algorithms utilized for the scheduler that try to optimize the usage of each thread or process while also attempting to achieve a good level of system responsiveness.

Intel® Hyper-Threading Technology

You have probably heard the term "Hyper-threading" and wondered, what is this? Hyper-threading or "simultaneous multithreading" is a term coined by Intel®. Intel's technology essentially enables the operating system to behave as though it is controlling two processors, allowing two threads to be run in parallel, both on separate 'logical' processors within the same physical processor. The operating system effectively sees two processors through a mix of shared, replicated and partitioned chip resources, such as registers, math units and cache memory. Intel® Hyper-Threading Technology can significantly improve performance on a supported processor by keeping the processor execution pipelines and resources busy and minimizing idle cycles. See diagram below:



Where AS = architectural state (eax, ebx, control registers, etc.)

Figure 1- Intel® Hyper-Threading Technology

Low power embedded applications, typically in the past, ran on small RISC-based or low end x86 systems with little or no multithreading exploited to help improve performance. Newer technology like Intel's Atom™, which is architected for Intel® Hyper-Threading Technology, can perform operations very efficiently. All of this performance, coupled with low power operation of only 2-3 watts for an entire Compute-On-Module, make embedded multithreading a very viable solution for system optimization. Introducing parallelism into embedded applications can now reap higher performance when coupled with hardware that is geared toward this parallel computing. Keep an eye on multi-core technology processors also, which are anticipated to be moving rapidly towards the embedded systems market with low power architectures. Embedded systems and applications designed with parallelism can quickly adopt and exploit the performance of future generation multi-core processors.

Virtualization

Virtualization technology allows a CPU to behave as if it were several CPUs working parallel, enabling several operating systems to run at the same on the same machine.

Virtualization is not multi-tasking: a multi-tasking system has one operating system with several programs running in parallel:

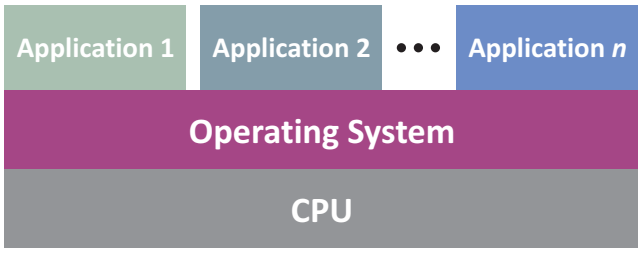


Figure 2- Multi-tasking

Virtualization is not hyper-threading: a hyper-threaded system simulates two CPUs for each physical CPU, allowing for balancing of performance of a single operating system using SMP (Symmetric Multi-Processing), and these two CPUs cannot be used totally independently of each other:

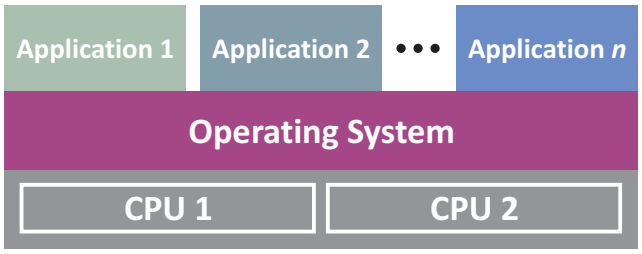


Figure 3- Hyper-threading

On a system with virtualization, there are two (or more) independent operating systems running in parallel on its own "virtual CPU" or "virtual machine", with each operating system able to run multiple programs:

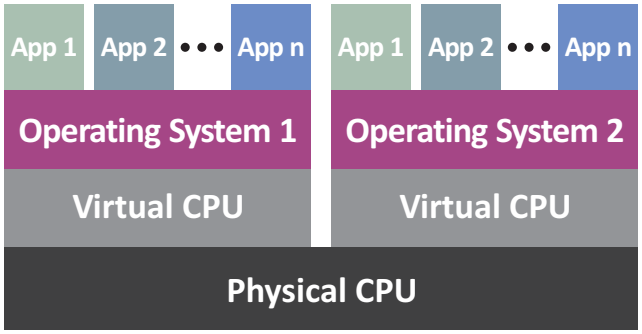


Figure 4- Virtualization

If a system has both hyper-threading and virtualization, each virtual CPU will appear to the operating system as if two CPUs are available on the system for symmetric multi-processing:

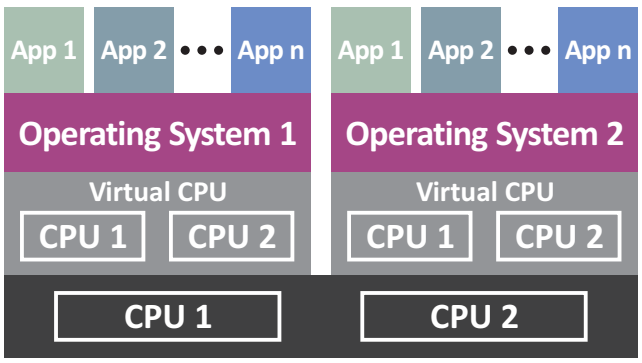


Figure 5- Virtualization with Hyper-Threading

Virtualization technology is not new in the computing industry; however, it's viability for use in low power embedded systems has been significantly moved forward by the architecture of Intel® Atom™ processor. There are numerous software packages available today that enable virtualization through software control of access to the physical hardware of the machine. Among these are VMware*, Microsoft* Virtual PC, and many others. There are three types of virtualization:

Paravirtualization is where the guest operating system is recompiled to use calls into the virtual machine software to emulate non-virtualizable instructions. This presents problems where the operating system source code is not available.

Full virtualization relies on software runtime binary translation to trap and virtualize the execution of non-virtualizable instructions. With this approach, the operating system code does not need to change as critical instructions are discovered and redirected with traps into software emulation routines.

Both paravirtualization and full virtualization incur large performance overhead, consumed machine time, due to fact that software must emulate non-virtualizable instructions. For low power embedded systems every clock cycle counts – for overall system performance and power consumption.

Hardware-assisted virtualization moves the handling of non-virtualizable instructions into the processor hardware, and adds new instructions to the instruction set to control virtualization. This greatly simplifies the virtual machine software, and allows the guest operating systems to run with higher performance and greater power efficiency.

Embedded Applications for Virtualization

Virtualization is typically considered a feature in server implementations – so, what does virtualization bring to the low power embedded system world? Here are four potential applications:

- Multiple applications written for different operating systems running on the same hardware at the same time - eliminates costly and timely porting of existing solutions and provides quicker time-to-market for systems involving legacy applications.
- Failure containment and high reliability applications – failure of one application or operating system will not halt or corrupt the whole machine. This is a key characteristic of many mission critical and 24/7 usage embedded systems.
- Remote access management or sideband system diagnosis and hardware monitoring – allows for continuous, tightly controlled, and fully "aware" runtime management of system.
- Operating system and application security from communication networks – second virtual machine can control access to the network and host any antivirus or network security applications independent of the main application virtual machine.
- Multi-level secure data management – as embedded systems evolve from standalone compute or control devices into fully networked environments such as financial, medical, military, and security systems, the requirements for secure data processing are rapidly increasing.

Compiler Optimizations

The Intel® Atom™ Processor represents a new generation of low-power IA-32 based Intel® processors and while it is compatible with IA-32 compiled code, there are new micro architecture features which can be additionally exploited with optimization in application code compilation. There are three general categories of compiler optimizations for the Intel® Atom™ Processor: in-order execution, new or preferable instructions added to the instruction set, and advanced features like SSE3 instructions and the big-endianness support.

Optimizations can be achieved using the Intel® C++ Compiler, versions 10.1 and 11.0 where the same techniques can be employed to generate code for both Microsoft Windows and Linux software stacks. This tool is a highly optimizing compiler for Intel® architecture and compatible processor technologies. It can be installed into an existing Microsoft Visual Studio* build environment or into an existing GNU* GCC installation. The compile optimizations for Intel® Atom™ Processor are enabled with the following switch settings:

- Intel® C++ Compiler 10.1
  - –xL (Linux*) or /QxL (Windows*)
- Intel® C++ Compiler 11.0 adds also the
  - –xSSE3_ATOM (Linux*) or /QxSSE3_ATOM (Windows*)

The most general difference in architecture to other Intel® processors is the in-order instruction scheduler. The scheduler feeds the instruction pipeline in exactly the order in which instructions are fed to it by the binary code of application. There is no instruction re-ordering done in the processor hardware. While the scheduler is much more power efficient than in other processors, there can be sensitivities to instruction latency and dependency stalls resulting from poor

scheduling by the source compiler. An example of ordering dependency stall:

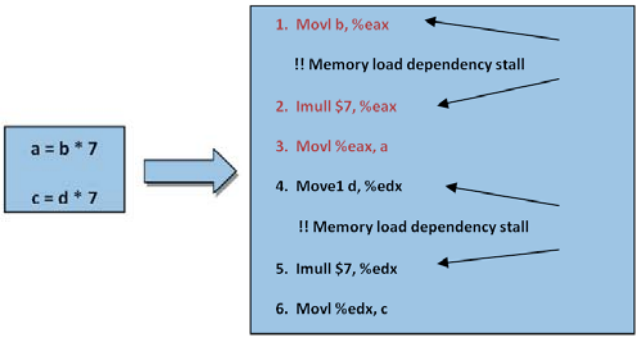


Figure 6- Ordering Dependency Stall

Result of compiler re-ordering to eliminate dependency stall:

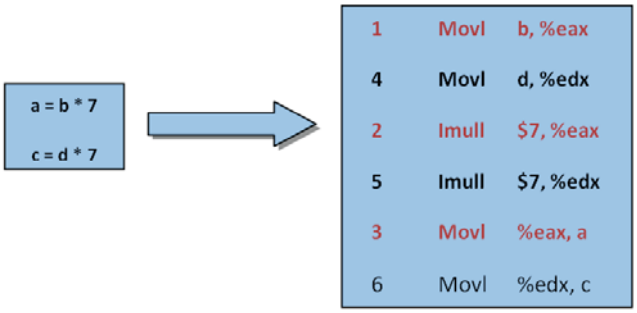


Figure 7- Re-ordered, No Dependency Stall

Another optimization point for the compiler is associated with memory access address generation. In order to minimize latencies, the compiler will output code targeted to speed the execution of generating and handing over memory access addresses. This is accomplished by using the LEA assembly instruction instead of EAX instruction so as to minimize memory accesses during the address generation phase and thereby minimizing the associated access latency. Example of memory access address generation:

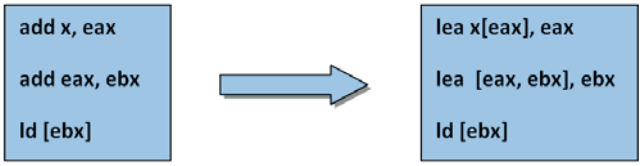


Figure 8- LEA Instruction Usage

Instruction flow optimization to avoid dependency stalls can also be accomplished with careful use of microcode or atomic instructions. The Intel® Atom™ Processor also supports byte-wise multiplication and division. Using byte-wide operations wherever the full width of integer operation is not required will promote parallelism and more efficient use of registers.

Embedded systems, especially storage and networking applications, often have many peripheral devices which require direct interface. One feature of the Intel® Atom™ Processor which can be very useful in support of the different peripheral devices of different “endianess“ is the MOVBE instruction which allows swapping of the low and high bits of a long value during the move operation. This instruction can also be used in arithmetic transformations. The Intel® C++ Compiler with –xSSE3_ATOM compile option enabled will automatically utilize the MOVBE instruction unless explicitly disabled via additional switch.

The SSE3 Instruction for Single Instruction Multiple Data execution for loop structures provides parallelizing execution of the loops and can significantly improve the execution flow of computational algorithms. This vectorization optimization is particularly useful in multimedia applications and large data streams often associated with graphics, gaming or encryption.

Additionally, there are higher level optimizations which are performed in multi-step process. Often the optimizations applied are unique to either the targeted processor architecture or the source code construction. Two of these optimization processes are the interprocedural optimization and profile-guided optimization. Care should be taken when using the compiler switches that enable these functions as there are implications for the size of resulting code files and the object file linking process.

System Power Management with Intel® Atom™ Processor

The Intel® Atom™ Processor is architected with advanced power management capabilities. For the low power embedded systems market, when coupled with the Intel® companion system controller hub, US15W, it represents an excellent example of a highly integrated and yet dynamically power managed platform. Below are some of the key power management features of this combined platform which can be enabled with advanced firmware and operating system components:

- Dynamic clock stepping and sub-system power down
- Power rail voltage stepping for processor core power
- Dynamic processor cache re-sizing and set associativity management
- Advanced processor "state" management including new ultra low power state with fast resume capability
- Dynamic I/O port enumeration, power up/down and re-enumeration
- Integrated thermal sensing and management hardware

When selecting an Intel® Atom™ Processor computer module to build your low power embedded system, it is critical to consider the power management and demonstrated operational characteristics of the computer module design. While there may be many offerings with basic functionality, the difference in real world application performance and particularly power management can vary widely.

For example, while an advanced COM using Intel® Atom™ Processor may be capable of running full Linux operating system with 1GB DDR2 memory and process 720p HD video at 3W or less, total COM power consumption, others may consume more than 6W to accomplish the same task. This is often due to the unique operating points and dynamic nature of the core logic power rails and sub-system power management.

Also, there is potential for significant differences in power management functionality provided by system BIOS, embedded controller architectures and OS level drivers and configuration. Care should be taken when selecting a COM or system provided for Intel® Atom™ Processor in order to achieve the best combination of user control over system power and performance trade-off.

Summary

The Intel® Atom™ Processor offers exciting new performance opportunities for embedded systems with a computing efficiency and level of optimization not available prior to this new generation of low power IA-32 based Intel® processors. Utilizing new performance techniques such as application processing parallelism thru multithreading and Intel® Hyper-Threading Technology, designers can create highly efficient and feature rich applications.

System architectures employing efficient hardware accelerated virtualization can now extend to the small form factor and low power devices, helping to create more highly reliable and secure computing environments and bringing forward legacy applications. Finally, optimized software applications for the Intel® Atom™ Processor need to be mated with reliable and power efficient computer modules or systems to gain the greatest leverage from this new technology and bring new heights of advanced computing to low power embedded system solutions.