

White Paper  
**Stephen  
Blair-Chappell**  
Technical Consulting  
Engineer  
Intel Corporation

# Upgrading to an Intel<sup>®</sup> Multicore Ecosystem Keeps a Car Simulator Running in the Fast Lane

January 2009



## *Executive Summary*

---

In this paper we describe how by upgrading both the hardware and software development environment of an embedded product, significant performance improvement can be obtained. Our observations are based on our experiences with Pi Shurlok PiAutoSim\*, a bench-top simulator that is intended for hardware-in-the-loop testing of automotive electronic control units.

---

*We show here how by upgrading to the latest multi-core architecture and by using the latest Intel® software tools we were able to gain a significant speed-up.*

---

Internally PiAutoSim\* uses a real-time operating system running on dedicated embedded controllers. We show here that by upgrading to the latest multi-core architecture and by using the latest Intel® software tools we were able to gain a significant speed-up of the control software.

We show how we used the VTune™ Performance Analyzer to sample and view the runtime behavior of the simulation. The simulation had some hard real-time constraints; by using the optimization features of the Intel® C/C++ Compiler we were able to improve the performance of the relevant sections of code.

One challenge was how to best use VTune™ and the Intel® Compiler into the embedded development environment. We describe here some of the steps we undertook to take full advantage of the new tools.



## ***Contents***

---

Introduction .....	4
The Pi Shurlok PiAutoSim* .....	4
The Software Stack .....	5
The Application .....	5
Hard time constraints .....	6
Upgrading for Performance.....	7
Updating the Software .....	8
Updating the Hardware .....	8
Levering the SIMD Extensions.....	9
Measuring and Tuning .....	10
The Benchmarks .....	10
Performance gains .....	11
Using Intel® VTune to Analyze the Runtime Performance .....	11
Conclusion & Next steps .....	14
References and Acknowledgements .....	14

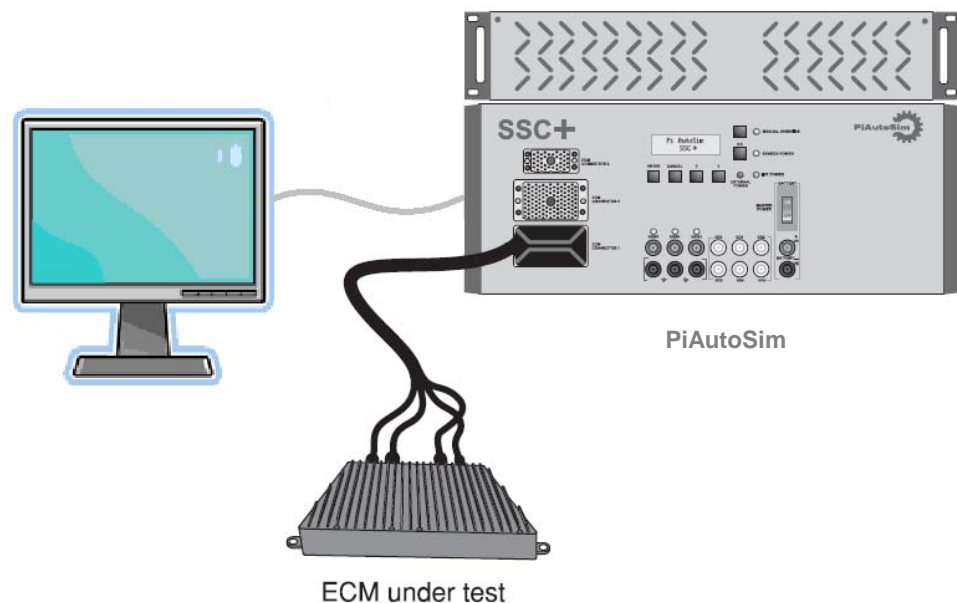


## Introduction

### The Pi Shurlok PiAutoSim\*

The PiAutoSim\* is a bench-top simulator that is intended for hardware-in-the-loop testing of automotive electronic control units. The unit is used by car manufacturers to test their Engine Control Modules (ECM) without having to fasten the ECM to an engine. *Engine Models* are built in software and used to test and validate the behavior of the ECM under test.

Figure 1. The PiAutoSim\*



The simulator is capable of mimicking the behavior of the engine under normal running and stress conditions. The PiAutoSim\* connects to the ECM under test and can generate and read a wide variety of analogue and digital signals (see [Figure 1](#)).

The PiAutoSim controller boards are based on the Intel® Pentium® processor. The configuring and running of the simulator is controlled via a mouse, keyboard and screen.

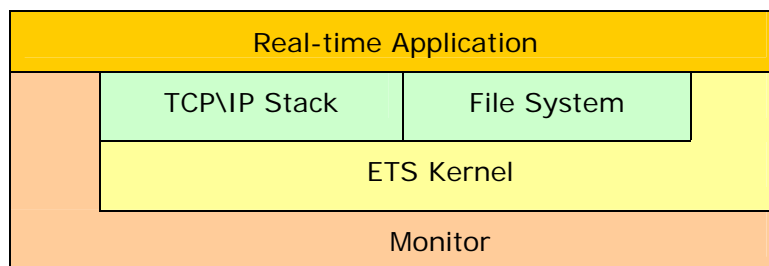


## The Software Stack

Internally PiAutoSim\* uses IntervalZero's ETS\* a real-time operating system running on IA32 architecture. The OS comes with support for the WIN32 API, which makes host-target development very easy. The software stack includes file system, TCP/IP support, the ETS kernel and a monitor ([Figure 2](#)).

Debugging can be done remotely via the monitor, with connections being made by Ethernet or serial cable.

**Figure 2. The Software Stack**

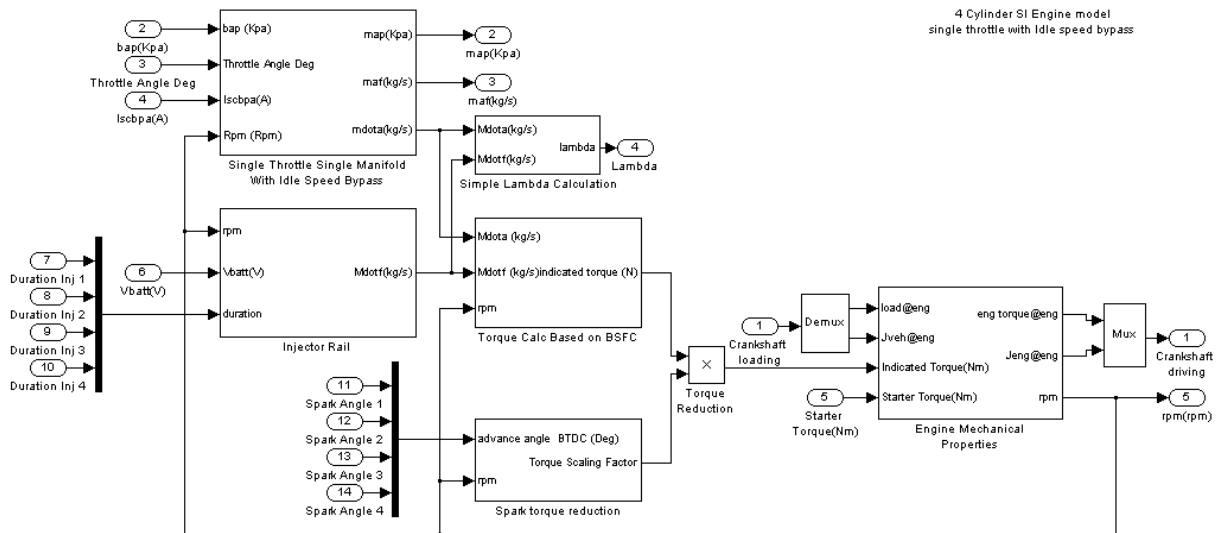


## The Application

The application is used to simulate the behavior of the plant physics of an engine. The C code for the engine simulation is created using Mathworks MATLAB RTW\* code generator, which is in turn is compiled using Microsoft\* compiler. The design of the engine can be seen in [Figure 3](#).



Figure 3. The MATLAB Single Engine Design



## Hard Time Constraints

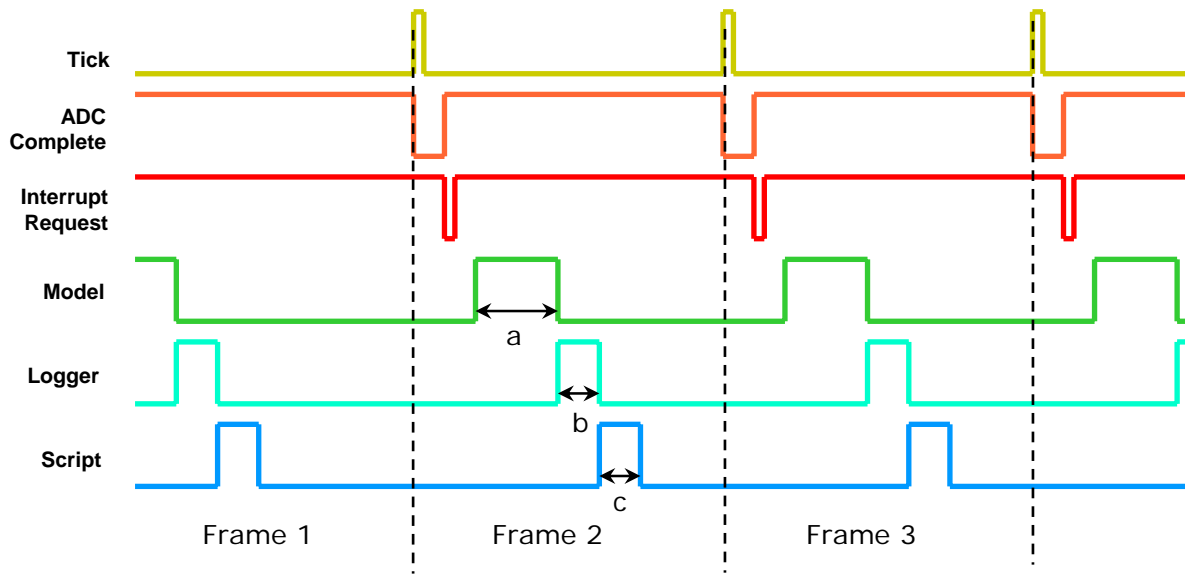
The simulator is designed to be deterministic in its behavior. A hardware generated clock tick is used to trigger a sequence of events. The clock tick or frame is programmed to have a time period of a few milliseconds.

In each frame of execution hardware interrupts and ADC conversions are first handled. There then follows time three slots for running the simulation model, logging the results and finally executing the next commands in a script file. Detail of the time frames and time slots can be seen in [Figure 4](#).

In our optimization work, we focused on making the code for the *Engine Model* (slot a) to run as quickly as possible. It was important that the full simulation did not overrun the allotted time period.



Figure 4. Simulation time frames



While for most situations the PiAutoSim\* has ample time to simulate the engine model, we undertook the optimization work with an eye to the future. With the increase in engine complexity it is inevitable that there will come a time when additional processing time will be needed.

The engine model parameters included information on speed, load, accelerator position, engine inertia, engine friction coefficient, engine stiction, engine efficiency, and ignition state.

## Upgrading for Performance

We upgraded the ecosystem in two phases. First we updated the software environment to use the Intel® C++ Compiler 11.0 running the newly built code on the existing system. We then upgraded the processor to an Intel® Pentium® processor with Intel® dual-core technology and pursued new opportunities for optimization based on the features of the new Intel® Core™ microarchitecture.

**Note:** We did not change the design of the application, or introduce parallelism into the application.

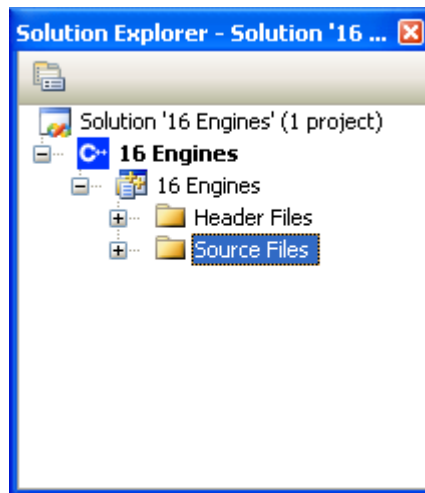


## Updating the Software

The original simulator project uses Microsoft Visual Studio\*. We installed the Intel® C++ Compiler 11.0 which is a plug and play replacement for the Microsoft\* C++ Compiler.

The integration was seamless and took only a short while to perform. When a visual studio project is configured to use the Intel® C++ Compiler 11.0, a new icon is displayed (Figure 5). The project can be reverted to use the Microsoft\* C++ Compiler at the press of a button.

Figure 5. The Visual Studio Project



Once the build environment was updated, we rebuilt and ran the engine simulation tests on the original hardware. We observed an immediate 20% performance improvement. The original project was built using Microsoft\* C++ Compiler version 6.0.

Once we made the transition to the Intel® C++ Compiler 11.0 we did not migrate back to the Microsoft\* compiler. We chose to concentrate on what the Intel® C++ Compiler 11.0 gave us when using some of the advanced Intel® architecture-specific optimizations that the Intel® C++ Compiler 11.0 brings to the table.

## Updating the Hardware

The original embedded hardware used Intel® Pentium® III and early Intel® Pentium® 4 devices. The plan was to future proof the kit by upgrade to the latest Intel® Core™ microarchitecture.



We experienced an immediate performance boost from the new hardware on three counts: first, due to a straight forward increase in clock speed; second, an increase in the number of execution units that were available in the processor; and last, a number of Single Instruction Multiple Data (SIMD) architectural extensions that can radically cut the execution time.

## Leveraging the SIMD Extensions

In order to take advantage of the SIMD instructions we configured the build so that the Intel Compiler used a technique called *auto-vectorization*.

In auto-vectorization the Intel compiler looks for opportunities to replace traditional floating point calculations with the more efficient SIMD instructions. The compiler analyses the code and looks for locations where floating point calculations are within loop constructs. By using the SIMD instructions, the compiler can effectively reduce the number of times such loops have to be executed

When building the simulation code we saw that the compiler succeeded in vectorizing, emitting the following message:

```
Engine.cpp  
.\Engine.cpp(129): (col. 2) remark: LOOP WAS VECTORIZED.
```

An extract from the code disassembly for the vectorized and non-vectorized code can be seen in [Figure 6](#). It can be observed that the vectorized code has its loop unrolled, and SSE instructions are used to do the calculation.



Figure 6. Disassembly of Vectorized Code

<pre>jmp      EngCyc+25h faddp   st(5),st mov     dword ptr [esp],eax fild   dword ptr [esp] fadd   st,st(3) fmul   st,st(2) add    eax,1 fmul   st,st(0) fadd   st,st(4) fdivr  st,st(1) cmp    eax,5F5E100h jl     EngCyc+23h</pre> <p style="text-align: center;"><i>Standard Code</i></p>	<pre>movaps  xmmword ptr [esp],xmm0 padd   xmm5,xmm6 addpd  xmm7,xmm3 mulpd  xmm7,xmm2 add    eax,8 mulpd  xmm7,xmm7 movaps  xmm0,xmmword ptr ds:[406770h] addpd  xmm7,xmm1 divpd  xmm0,xmm7 cvt dq2pd  xmm7,xmm5 padd   xmm5,xmm6 addpd  xmm4,xmm0 movaps  xmm0,xmmword ptr ds:[406770h] addpd  xmm7,xmm3 mulpd  xmm7,xmm2 mulpd  xmm7,xmm7 addpd  xmm7,xmm1 divpd  xmm0,xmm7 movaps  xmm7,xmmword ptr [esp] addpd  xmm7,xmm0 cvt dq2pd  xmm0,xmm5 movaps  xmmword ptr [esp],xmm7</pre> <p style="text-align: center;"><i>Vectorized Code</i></p>
---	--

## Measuring and Tuning

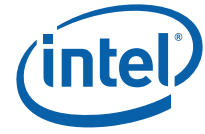
### The Benchmarks

Because both our target and host hardware were using the Intel<sup>®</sup> architecture it gave us the advantage that we could choose to run most of the tests on our host machine. We did our initial tests on a standard workstation, using the results to help understand the gains we would make on the embedded Intel<sup>®</sup> architecture platform.

We used two benchmarks to exercise the simulator. The first consisted of an engine simulation. In order to get a decent workload we actually simulated 16 engines at once – with this workload we were able to push the simulator its limits. The second benchmark was a single engine simulation enclosed in a 100,000 count loop.

We used the first benchmark for the initial test of the Intel compiler, which as stated earlier, gave a performance boost of 20%. We used the second test – the loop simulation – for the remainder of our testing. In this paper we concentrate on the loop test results.

All of our benchmarks were measured using the system clock and then converting the time elapsed into seconds.



## Performance gains

The results of the tests can be seen in [Table 1](#).

The tests were run on two machines, one containing an Intel® Pentium® 4, the other an Intel® Core™ 2. When we ran the tests on the Intel® Pentium® 4, we used the /QxK option which enables automatic vectorization for CPUs supporting SSE2. We could have chosen to use a more advanced vectorization option, but we chose this option because the results would be more representative of what we would achieve on the original embedded target containing the Intel® Pentium® III.

When running tests on the Intel® Core™ 2, we used the compiler option /QxT which enables vectorization for CPUs supporting SSE4.

**Table 1. The 100k Loop Simulation Results.**

	O2	O2 & QxT\K**	Speedup
<b>P4 **</b>	39.344 <sup>(1)</sup>	21.9	<b>1.80</b>
<b>Core 2</b>	5.546	0.515 <sup>(2)</sup>	<b>10.77</b>
<b>Speedup</b>	<b>7.09</b>	<b>42.52</b>	

**Note:** \*\* the P4 test was configured to support SSE2

The Speedup in the right-hand column represents the speedup we obtain when turning on vectorization for the two platforms. The P4 gave a speedup of 1.8. The Intel® Core™ 2 provided a speedup of over 10. Although a speedup of between 2 and 3 is not unusual in a customer's application, it is not often that we observe speedups in the order of 10 or more.

The speedup figures in the bottom row represent the gain we achieved by moving from Pentium® III to Intel® Core™ 2. On non-vectorized code we achieved a gain of 7; on vectorized code we achieved a gain of 42.

To calculate the performance improvement when upgrading from the *Pentium® 4 without vectorization* to the *Intel® Core™ 2 with vectorization* we use the values marked with (1) and (2) to work out the ratio. This gives a total speedup of over 76. Note that this is not a percentage but a raw multiplier.

## Using VTune™ to Analyze the Runtime Performance

We used the VTune™ Performance Analyzer to confirm the timing measurements we had achieved, and to give a clearer insight into where the origins of the performance improvements were.



Intel® VTune is a tool that can be used for measuring the performance of a running application. It relies on configuring the Performance Monitoring Unit (PMU) of the CPU to record certain kind of hardware events.

After the results of the application have been collected the can be displayed in the Intel® VTune GUI.

We configured Intel® VTune to capture a number of events which are described in the table below in [Table 2](#).

The actual values we recorded for these events can be found in [Table 3](#).

**Table 2. Captured CPU Events**

Event	Description
CPU_CLK_UNHALTED.CORE	A counter that is incremented every cycle. We use this to measure cycles consumed by the running program
INT_RETIRED.ANY	The number of instruction that have been executed to completion.
X87_OPS_RETIRED.ANY	The number of floating point instructions that have been executed to completion.
SIMD_INST_RETIRED	The number of SIMD instructions that have been executed to completion.

**Table 3.Events captured on the loop host-based simulation**

CPU events	Optimize for speed (/O2)	Optimize for speed (/O2) & Auto-vectorization (/QxT)
CPU_CLK_UNHALTED.CORE	16,641,000,448	1,548,000,000
INT_RETIRED.ANY	3,308,999,936	1,395,000,064
X87_OPS_RETIRED.ANY	250,000,000	0
SIMD_INST_RETIRED	0	763,000,000

### CPU\_CLK\_UNHALTED.CORE

This value is incremented every cycle and is used as raw measure of cycles consumed by the running program. We can see that the number of cycles with vectorization turned off is about 10 times that when vectorization is



switched on. This confirms that the elapsed time of 5.546 seconds *and* 0.515 seconds that we recorded in [Table 1](#) look right.

## INST\_RETIRED.ANY

This is the number of instructions that have been executed to completion, i.e. they have been fetched from the memory subsystem, decoded, dispatched, executed and the results written back to a register. We can see that the number of instructions executed when there is no vectorization is almost three times that of when vectorization is enabled.

## X87\_OPS\_RETIRED.ANY and SIMD\_INST\_RETIRED.

These two events record the number of floating point and SIMD calculations that were executed to completion in the life of the program. Traditional floating point instructions are expensive to execute. On the Intel® Core™ 2 architecture there are two floating-point execution units – one for arithmetic and the other for divide. The Intel® Core™ 2 has three SIMD execution units, with each execution unit being able to perform four single-floating point operations or two double-floating point operations in one cycle. This means that in a given cycle we can potentially execute the equivalent of 12 floating point instructions when using SIMD. In our program *one* floating point instruction was, on average, replaced by *three* SIMD instructions.

## Cycles Per Instruction Retired (CPI)

Even though we had three times as many SIMD instructions as floating point instructions, they were still performed much quicker. We can get a measure of the efficiency by calculating the CPI (see [Table 4](#)). A value of 1.36 is very respectable, whereas 5.02 is poor. A high CPI is usually regarded as an indication that optimization opportunities exist in the application.

**Table 4. CPI of the loop host-based simulation**

Option	CPI
/O2	5.02
/O2 /QxT	1.36



## Conclusion and Next steps

---

In this paper we showed that by upgrading the software development platform and the hardware, we were able to bring a significant performance boost to an embedded application

Using the Intel® architecture gave us a number of benefits that made our task easier.

- Using Intel® architecture embedded target meant we could test code on Host platform prior to running code on target.
- Using the latest Intel® C++ Compiler we were able to produce code that was optimized for the Intel® architecture embedded target.
- By upgrading to the latest multi-core processors we were able to harness the latest features leading to a significantly improved performance.
- Running the existing application on the new hardware has left us with lots of headroom for further optimization. In particular we anticipate that future versions of the ETS\* operating system will support SMP, thus giving us further optimization opportunities by way of parallelism.

By upgrading the ecosystem to the latest multicore we achieved

- An initial speedup of 20% by using the Intel® C++ Compiler on the original hardware
- A final speedup of over 76 (i.e. 7600%), which consisted of:
  - 10 times speedup due to enabling auto-vectorization.
  - 7 times speed up due to hardware upgrade.

## References and Acknowledgements

We'd like to record our thanks to the Pi Shurlok office in Cambridge for their support and help. Further information on the PiAutoSim\* can be found at [www.pi-shurlok.com](http://www.pi-shurlok.com)

Information on the Intel® software tools including a free evaluation download can be found at [www.intel.com/software](http://www.intel.com/software)

Details about the IntervalZero ETS\* realtime operating system can be found at [www.IntervalZero.com](http://www.IntervalZero.com)



### **Authors**

**Stephen Blair-Chappell** is a Technical Consulting Engineer with Intel Compiler Labs at Intel Corporation.

### **Acronyms**

ADC	Analogue to Digital Conversion
API	Application Programmer Interface
CPU	Central Processing Unit
ECM	Engine Control Modules
ECU	Engine Control Unit
ETS	IntervalZero's realtime operating system
SIMD	Single Instruction Multiple Data
SMP	Symmetric Multiprocessing
SSE	Streaming SIMD Extension



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.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, visit Intel Performance Benchmark Limitations

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.