

---

**Publisher**

Richard Bowles

**Managing Editor**

Andrew Binstock

**Content Architect**

Muntaquim Chowdhury

**Program Manager**

Stuart Douglas

**Technical Editor**

Marian Lacey

**Technical Illustrators**

InfoPros

**Technical and Strategic Reviewers**

Per Hammarlund

Dave L. Hill

Ronak Singhal

Muntaquim Chowdhury

## Intel Technology Journal

Copyright © 2011 Intel Corporation. All rights reserved.  
ISBN 978-1-934053-33-1, ISSN 1535-864X

Intel Technology Journal  
Volume 14, Issue 3

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4744. Requests to the Publisher for permission should be addressed to the Publisher, Intel Press, Intel Corporation, 2111 NE 25th Avenue, JF3-330, Hillsboro, OR 97124-5961. E-mail: [intelpress@intel.com](mailto:intelpress@intel.com).

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold with the understanding that the publisher is not engaged in professional services. If professional advice or other expert assistance is required, the services of a competent professional person should be sought.

Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

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

Third-party vendors, devices, and/or software are listed by Intel as a convenience to Intel's general customer base, but Intel does not make any representations or warranties whatsoever regarding quality, reliability, functionality, or compatibility of these devices. This list and/or these devices may be subject to change without notice.

Fictitious names of companies, products, people, characters, and/or data mentioned herein are not intended to represent any real individual, company, product, or event.


Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Intel, the Intel logo, Celeron, Intel Centrino, Intel Core Duo, Intel NetBurst, Intel Xeon, Itanium, Pentium, Pentium D, MMX, and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

**For more complete information about performance and benchmark results, visit** [HYPERLINK "http://www.intel.com/benchmarks"](http://www.intel.com/benchmarks) [www.intel.com/benchmarks](http://www.intel.com/benchmarks)

This book is printed on acid-free paper. 

Publisher: Richard Bowles  
Managing Editor: Andrew Binstock

Library of Congress Cataloging in Publication Data:

Printed in United States of America

10 9 8 7 6 5 4 3 2 1

First printing: July, 2011

INTEL® TECHNOLOGY JOURNAL  
THE TICK TOCK BEAT OF MICROPROCESSOR  
DEVELOPMENT AT INTEL

Articles

Foreword ..... 5

Preface ..... 7

The Next-generation Intel® Core™ Microarchitecture ..... 8

The Uncore: A Modular Approach to Feeding the High-performance Cores ..... 30

Energy-efficient Computing: Power Management System on the Nehalem Family of Processors ..... 50

The Feeding of High-performance Processor Cores—QuickPath Interconnects and the New I/O Hubs ..... 66

Architected for Performance—Virtualization Support on Nehalem and Westmere Processors ..... 84

Circuit and Process Innovations to Enable High-performance, and Power and Area Efficiency  
on the Nehalem and Westmere Family of Intel Processors ..... 104

The Road to Production—Debugging and Testing the Nehalem Family of Processors ..... 128

The Toolbox for High-performance CPU Design ..... 148



## FOREWORD

by **Reynold D'Sa**

General Manager

Converged Core Development Organization

Intel Corporation

It is a singular honor to introduce this special edition of the *Intel Technology Journal* (ITJ) dedicated to the recent spate of products designed and developed by the Converged Core Development Organization (CCDO) at Intel. In the last few years, CCDO, in conjunction with our partners in the Architecture and other silicon-development organizations, has delivered a wide array of products that have been well received by our customers and the technical press: these comprise our flagship products. From a distance, it would seem that these products are delivered effortlessly very much in cadence with the tick-tock drumbeat of the Intel® product roadmap. However, behind each product stands teams of engineers that have spent years working on the product from inception to tape-in. I would like to talk about how we conceive, execute, and deliver our products and briefly explain the role of the many and varied technologists that are involved with every step of our design process.

Each of our products goes through three distinct phases of the product development cycle: product definition, execution, and post-Si debug. During product definition, we evaluate a highly-complex matrix of technology trends and innovations to determine what our partners and customers will need from us and what they will want four years down the road. At this initial stage, our technologists (architects, process engineers, and design engineers) work together to create a portfolio of technical innovations. These innovations span the spectrum of technical possibilities, from new architecture and circuit technologies to break-through platform capabilities. In parallel, we collaborate with our business-group partners and their planning teams to understand our customers' needs and the development trajectory of their products. The product that we plan to build stems from the confluence of these two streams—Intel innovation and customer requests. This is our recipe for defining leadership products that are constantly pushing the technological boundaries of our industry.

The execution phase requires a monomaniacal focus on CPU design and tape-out. The entire design team, along with our partners in Process Technology and Architecture, goes into full gear to transform the abstract product specification into the fabric of interconnected transistors—often billions in numbers—that actually implement the product. In the course of execution, we go through various levels of abstractions, starting with a register transfer level (RTL) model that is eventually transformed into the layout, which in turn defines the actual silicon implementation. In addition to “building” the product, we also validate the correctness of the implementation across many dimensions ranging from

logical correctness, adherence of the design to stringent process requirements, fidelity of the end-product to projected performance, and so on. This large and complex array of activities has to be executed flawlessly. The execution phase culminates in the tape-in of the actual die. This step also starts the final phase of our product development cycle, which includes various post-silicon activities. The articles in this issue of the ITJ do not cover this phase of the development cycle.

Another challenging aspect of our design cycle is that at any given time, we have multiple products under development. In the last two years, we have had five major products in concurrent development. We have achieved this breakthrough level of execution efficiency and productivity as a result of the “one-team” execution model that was first introduced by CCDO. In this execution paradigm, a common brain-trust is leveraged over multiple concurrent projects owned by the same team. This common-mode harvesting of expertise enables resource-efficient execution across not one but five CPU projects.

I hope this foreword has given you a glimpse of our intricate and complex product development cycles. The articles in this issue of the ITJ provide a more in-depth look into different facets of our products, our development philosophy, and the tools and flows used in our design. I hope you find the articles and their contents informative and instructive.

Reynold D’Sa

## PREFACE

In this special edition of the *Intel Technology Journal*, we take a holistic look at the development of CPUs at Intel. By now the “tick-tock” beat of CPU development at Intel is well known. The “tick” and “tock” form a strongly-coupled pair of consecutive generations of CPUs that straddle two process technologies. The tock CPU is a major inflection point in architectural development; whereas the tick provides linear architectural improvement over the tock, but marries that improvement to the latest process offering from Intel. Each of these designs carries its own challenges. The primary challenge of the tock is the introduction of radically new platform and CPU features; whereas the primary challenge of the tick is to bring the latest generation of Intel process technology into full production mode.

The common denominator to both tick and tock design is a highly sophisticated design flow and a tightly integrated compendium of CAD tools and flows that provide the design team with the unique capability to deliver CPUs that are constantly pushing the boundaries of the state of the art.

In this issue of the *Intel Technology Journal*, we focus on the Intel® microarchitecture code name Nehalem (45nm Tock) and the Intel® microarchitecture code name Westmere (32nm Tick) generation of microprocessors. The articles in this issue can be broadly divided into two categories: those that focus on the architecture innovations and those that focus on our design flow and capabilities. Our hope and expectations are that, taken in tandem, these two categories of articles will provide valuable technical insight into how modern complex CPUs are designed and implemented.

# THE NEXT-GENERATION INTEL® CORE™ MICROARCHITECTURE

## Contributors

**Martin Dixon**  
Intel Corporation

**Per Hammarlund**  
Intel Corporation

**Stephan Jourdan**  
Intel Corporation

**Ronak Singhal**  
Intel Corporation

## Index Words

Nehalem  
Westmere  
Core  
Power consumption  
Loop Stream Detector  
New Instructions  
Physical Addressing

*“We added features only if they added performance in a power-efficient manner. If features were beneficial for performance, but were not power efficient, we did not pursue them for this design.”*

## Abstract

The next-generation Intel® microarchitecture was designed to allow products (under the *Intel® microarchitecture code name Nehalem* and *Intel® microarchitecture code name Westmere*) to be scaled from low-power laptops to high-performance servers. The core was created with power efficiency in mind and offers performance improvements for both lightly-threaded and highly-threaded workloads, while also adding key segment-specific features. We describe the innovative techniques used to achieve these goals in the core, including the development of traditional microarchitecture enhancements, new instructions, the addition of Intel Hyper-Threading Technology, innovative power-management schemes, and other performance and power improvements throughout the core.

## Introduction

The Intel® microarchitecture that appears in Nehalem and Westmere processors is the follow-on to the successful Intel® Core™ and Intel® Core 2 products and forms the basis of the Intel® Core™ i3, Intel® Core™ i5 and the Intel® Core™ i7 series of chips and the Intel Xeon® 5500/5600/7500 CPUs [1]. Each of these products is built by combining a processor core that contains the instruction processing logic and an “Uncore” that contains the logic that glues together multiple cores and interfaces with the system beyond the processor. The core is common to all products, while Uncores are designed for specific market segments. An example Uncore is described in another article in this issue of the *Intel Technology Journal* [2]. In this article, we detail the key features of the core that forms the basis of all the Nehalem and Westmere family of products. Given that the processor core had to scale from low-power laptops to high-end servers, we designed it with the overall goal of power efficiency for a wide range of operations, while achieving improved performance over the prior generation of processors.

## Guiding Principles

Given the range of products to be covered, the focus on power efficiency required a new mindset in choosing which features to include. Even though our goal was still to provide high performance, we added features only if they added performance in a power-efficient manner. If features were beneficial for performance, but were not power efficient, we did not pursue them for this design. For this design, the starting point was the Intel Core 2 microarchitecture (code-named Penryn).



The rule of thumb for judging the power efficiency of any particular performance feature was to compare performance without the feature versus performance with the feature within the same power envelope. Total power consumed by a component includes dynamic power plus leakage power. Dynamic power is equal to the capacitance multiplied by the frequency and the square of the voltage. Features that increase capacitance require the frequency and voltage to be dropped to remain at the same power level. Making the assumption that frequency and voltage are linearly related and that leakage is directly proportional to dynamic power, then power is cubically related to frequency. Therefore, our basic rule of thumb is that a feature that adds 1% performance needs to do so by adding less than 3% power. For our analysis, we make the assumption that the interesting power-scaling range starts at the maximum frequency achievable at the minimum supported voltage. To drop below that frequency, the voltage is held constant, so there is only a linear relationship in that range. The power efficiency that needs to be achieved in that linear scaling range is much more strict and therefore a much less power-efficient operating point, one that we seek to avoid. Consequently, if a performance feature is worse than cubic in its power-performance relationship, then we are actually losing performance in a fixed power envelope. Overall, for this microarchitecture generation, the majority of the features added were much better than this 3:1 ratio, with most features approaching 1:1.

*“Our basic rule of thumb is that a feature that adds 1% performance needs to do so by adding less than 3% power.”*

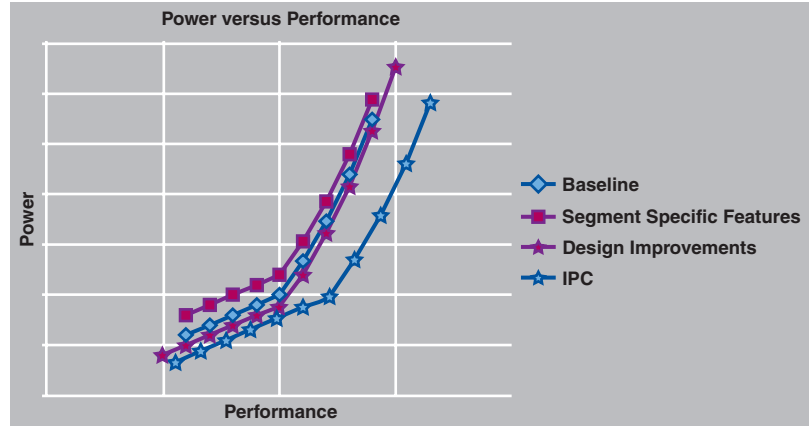
Beyond these power-efficient performance features, we also needed to add market segment-specific features. These were features that were critical for one of our target segments (mobile, desktop, server) but not in other segments. For instance, the amount of memory that needs to be addressed is manifested in the number of physical address bits that are supported in the core. The server segment has a much higher demand for memory capacity than either desktop or mobile products. It was important to find a balance between the needs of each market segment in order to maintain our power efficiency while still hitting our segment targets.

Figure 1 shows the power-performance characteristics of the core changes with the addition of segment-specific features, design and microarchitecture improvements, and performance work. Starting with a baseline curve (Baseline), we add the segment-specific features (Segment-Specific Features). With these features, the curve moves up because the power increases without a performance change. The design and microarchitecture enhancements (improvements) do three things to the curve:

- The enhancements move the curve down due to power reduction work.
- They extend the operating point towards the left, by enabling operation at lower voltages.
- They extend the operating point to the right by enabling higher frequency operation.

*“We end up with a core that yields higher performance at lower power envelopes and that covers a wider range of power envelopes.”*

The power-efficient performance features increase the instructions per clock (IPC) and move the curve up and to the right. Overall, we end up with a core that yields higher performance at lower power envelopes and that covers a wider range of power envelopes.



**Figure 1:** Power-performance for different changes to a processor design

Summing up our approach for this microarchitecture, the guiding principles we used in designing the core microarchitecture were these:

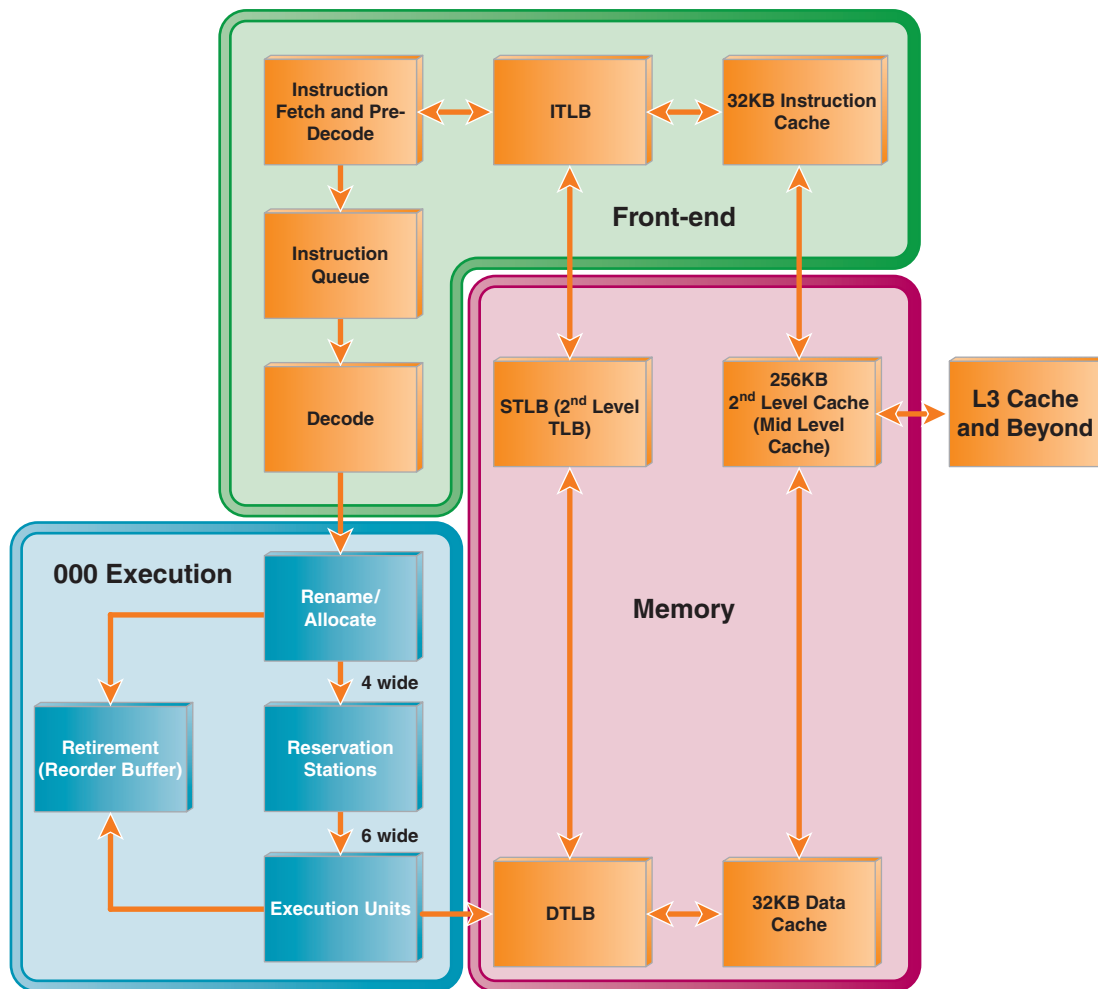
- Simply aggregating more and more cores is not sufficient to provide a well-balanced performance increase. We must also deliver a per-core performance increase since not all software has been, or even can be, upgraded to use all available threads. Moreover, even highly parallel software still has serial sections that benefit from a faster core. The need for increased per-core performance has been echoed by many of our key customers.
- Customers should not have to choose between high performance when all cores are active or high performance when only some cores are being used. Prior to this generation of processors, customers had to make a tradeoff between number of cores and the maximum frequency that cores could run at. We sought to eliminate the need for this tradeoff.
- Power envelopes should remain flat or move down generation after generation. This enables smaller form factors in laptops, while it also addresses critical power constraints that face servers today. Therefore, while we strived for higher performance, we could not do it by increasing power.

## Microarchitecture Overview

The Core 2 microarchitecture had a theoretical maximum throughput of four instructions per cycle. We chose to maintain that maximum theoretical throughput, so our focus was on how to achieve a greater utilization of the possible peak performance.

The core consists of several sub-blocks, known as clusters, that are responsible for key pieces of overall functionality. In each cluster, a set of new features was added to improve the overall efficiency of the core. As seen in Figure 2, there are three basic clusters that make up the core. The front end is responsible for fetching instruction bytes and decoding those bytes into micro-operations ( $\mu$ ops) that the rest of the core can consume. The Out of Order (OOO) and execution engine are responsible for allocating the necessary resources to  $\mu$ ops, scheduling the  $\mu$ ops for execution, executing them, and then reclaiming the resources. The memory cluster is responsible for handling load and store operations.

*“The core consists of several sub-blocks, known as clusters, that are responsible for key pieces of overall functionality.”*



**Figure 2:** Block diagram of the core

### Front End

One goal for the front-end cluster is to provide a steady stream of  $\mu$ ops to the other clusters. Otherwise, the rest of the machine starves waiting for operations to execute. The focus for performance improvements is providing higher effective throughput through the front end while keeping latencies low.

*“Many fundamental portions of the front end remain unchanged from previous generations of processors.”*

*“The instruction cache size remains at 32 kilobytes and is organized in 64-byte lines.”*

*“With each generation, branch prediction is typically near the top of that list. The rationale for this is simple: more efficient branch prediction gives better efficiency with no other changes to the machine.”*

*“We also added a Renamed Return Stack Buffer (RSB). This idea was first implemented in the Core 2 Penryn Processor family.”*

*“In prior microarchitecture generations, mispredicting branches could corrupt the RSB.”*

Many fundamental portions of the front end remain unchanged from previous generations of processors. Specifically, the instruction cache size remains at 32 kilobytes and is organized in 64-byte lines. Similarly, there are four decoders that are used to translate raw instruction bytes into  $\mu$ ops for the rest of the machine to consume.

One area in the front end that has traditionally been improved in each generation of processors is the accuracy of branch prediction. At the beginning of each project, we look at what areas of the processor have the greatest leverage for improving overall performance. And with each generation, branch prediction is typically near the top of that list. The rationale for this is simple: more efficient branch prediction gives better efficiency *with no other changes to the machine*.

Even though the predictors today have a very high accuracy rate, improving the prediction accuracy still has a significant impact on performance, because of the opportunity cost of a mispredicted branch. When a branch is mispredicted, the impact of flushing the pipeline increases with each generation of processor as we improve the throughput and increase the speculation depth of the core.

Accurate branch prediction is also critical for power efficiency. Each mispredicted branch represents a case where instructions were fetched, decoded, renamed, allocated, and possibly executed, and then thrown away. Because they are thrown away, the work provides almost no benefit, yet it costs power. More accurate branch prediction reduces speculative operations and can result in higher power efficiency.

In this microarchitecture generation, we make several notable improvements to our branch predictors. First, we add a second-level (L2) branch predictor. The purpose of this predictor is to aid in improving the prediction accuracy for applications that have large code footprints that do not fit well in the existing predictors. This addition is in line with providing better performance for server workloads, like databases, that typically have large code footprints.

We also added a Renamed Return Stack Buffer (RSB). This idea was first implemented in the Core 2 Penryn Processor family [2]. CALL instructions are branches that are typically used to enter into functions, and RET (Return) instructions are branches used to exit functions and return back to where the function CALL occurred. An RSB is a microarchitecture feature that implements a simple stack. The content of the stack is maintained such that when CALLs occur, the address of the CALL is pushed onto the stack. When a RET occurs, it pops an address off the stack and uses the popped address as its prediction of where the RET is branching to. However, in prior microarchitecture generations, mispredicting branches could corrupt the RSB. For instance, a CALL instruction would push an address onto the stack, but

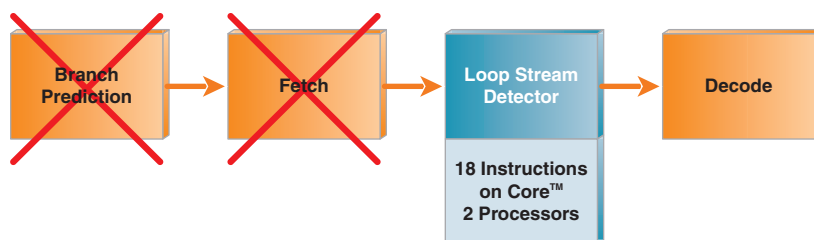
then would itself be flushed from the machine due to a mispredicted branch. A subsequent RET would then pop the address from the mispredicted CALL off the stack and would also then mispredict. With the renamed RSB, we are able to avoid these corruption scenarios.

Beyond branch prediction, another area of improvement in this generation is increasing the number of *macrofusion* cases. Macrofusion is a capability introduced in the Core 2 microarchitecture where a TEST or CMP instruction followed by certain conditional branch instructions could be combined into a single  $\mu$ op. This is obviously good for power by reducing the number of operations that flow down the pipeline, and it is good for performance by eliminating the latency between the TEST/CMP and the branch.

In this generation, we extend macrofusion in two ways. First, Core 2 would only macrofuse operations in 32-bit mode. In this generation, macrofusion can be applied in both 32-bit and 64-bit mode. Second, the number of conditional branch cases that support macrofusion was increased. Newly supported macrofusion cases in this generation are a compare (CMP) instruction followed by these instructions:

- JL (Jump if less than)
- JGE (Jump if greater than or equal)
- JLE (Jump if less than or equal)
- JG (Jump if greater)

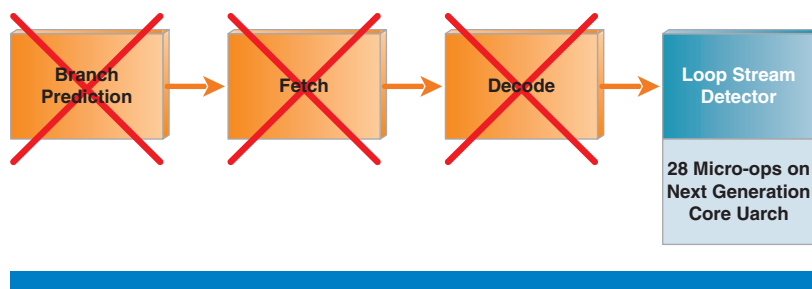
Another area in the front end where we sought to improve both power efficiency and overall performance was in the Loop Stream Detector (LSD). The motivation behind the LSD is fairly straightforward. Short loops that execute for many iterations are very common in software. In a short loop, the front end is fetching and decoding the same instructions over and over, which is not power efficient. The LSD was created to capture these short loops in a buffer, reissue operations from that buffer, and then reduce power by disabling pieces of logic that are not needed—since their work is captured in the state of the buffer. Figure 3a shows the LSD as it existed in the Core 2 microarchitecture. The branch prediction and instruction fetch sub-blocks are powered down when running out of the LSD.



**Figure 3a:** Loop Stream Detector in the Core 2 microarchitecture

*“Macrofusion is a capability introduced in the Core 2 microarchitecture where a TEST or CMP instruction followed by certain conditional branch instructions could be combined into a single  $\mu$ op.”*

*“In a short loop, the front end is fetching and decoding the same instructions over and over, which is not power efficient. The LSD was created to capture these short loops in a buffer, reissue operations from that buffer, and then reduce power by disabling pieces of logic that are not needed.”*



**Figure 3b:** Loop Stream Detector in this microarchitecture generation

*“In the prior architecture generation, the LSD could cover loops of up to 18 instructions. Now, in this generation, loops of up to 28.”*

In this microarchitecture generation, we made two key improvements to the LSD. First, we move the LSD to a later point in the pipeline, as seen in Figure 3b. By moving it after the decoders, we can now turn off the decoders when running from the LSD, allowing even more power to be saved. This change also provides for higher performance by eliminating the decoders as a possible performance bottleneck for certain code sequences. Second, by moving the location of the LSD, we can now take advantage of a larger buffer. In the prior architecture generation, the LSD could cover loops of up to 18 instructions. Now, in this generation, loops of up to 28  $\mu$ ops in length can be executed out of the LSD. Our internal studies show that, on average, the number of  $\mu$ ops per instructions is nearly 1, so the 28  $\mu$ op deep buffer is virtually equivalent to a 28-instruction deep buffer.

### Out of Order and Execution Engine

The OOO and Execution Engine cluster are responsible for scheduling and executing operations. In this generation, we looked to improve overall performance by exploiting greater levels of parallelism.

*“We increased the size of the OOO window that the hardware scans by 33% from 96 operations in the Core 2 microarchitecture to 128 operations.”*

To exploit greater parallelism during execution, we need the processor core to scan across a greater range of operations to identify cases that are independent and ready to execute. In this generation of processors, we increased the size of the OOO window that the hardware scans by 33% from 96 operations in the Core 2 microarchitecture to 128 operations. This window is implemented as the Reorder Buffer (ROB), which tracks all operations in flight. Because the pipeline depth in this processor generation is effectively the same as in the prior generation, the entire benefit of this increase is used for performance instead of for simply covering a deeper pipeline.

Additionally, when increasing the size of the ROB, we need to increase the size of other corresponding buffers to keep the processor well balanced. If we do not make those increases, all we would be doing is shifting the location of the performance bottleneck and not actually getting any value out of the larger

ROB. Consequently, in this generation, we also increased the load buffer by 50% and the store buffer sizes by more than 50%. The increase in the size of these buffers was proportionally greater than the increase in the ROB size, based on data we collected that showed that performance bottlenecks in the previous technology generation were too often caused by the limited number of load and store buffers rather than by the ROB. Table 1 summarizes the sizes of these structures in this generation of products as compared to those in the prior product generation.

Structure	Core 2	Next-Generation (Nehalem/Westmere)	Comment
Reservation Station	32	36	Dispatches operations to execution units
Load Buffers	32	48	Tracks all load operations allocated
Store Buffers	20	32	Tracks all store operations allocated

**Table 1:** Size of Key Structures

Operations are scheduled from our unified reservation station (RS). “Unified” means that all operations, regardless of type, are scheduled from this single RS. We increased the size of the RS from 32 to 36 entries in this processor generation as another means of expanding the amount of parallelism that can be exploited.

The RS is capable of scheduling an operation every cycle on each of its six execution ports:

- Ports 0, 1, 5: Integer operations plus Floating Point/SSE operations.
- Port 2: Load operations.
- Port 3: Store Address operations; store operations are broken into two pieces: an address operation and a data operation.
- Port 4: Store data operation.

This number of execution ports is the same as in the prior generation microarchitecture. The RS and its connection to the execution units are shown in Figure 4.

*“We also increased the load buffer by 50% and the store buffer sizes by more than 50%.”*

*“Operations are scheduled from our unified reservation station (RS).*

*“Unified” means that all operations, regardless of type, are scheduled from this single.”*

*“The RS is capable of scheduling an operation every cycle on each of its six execution ports.”*



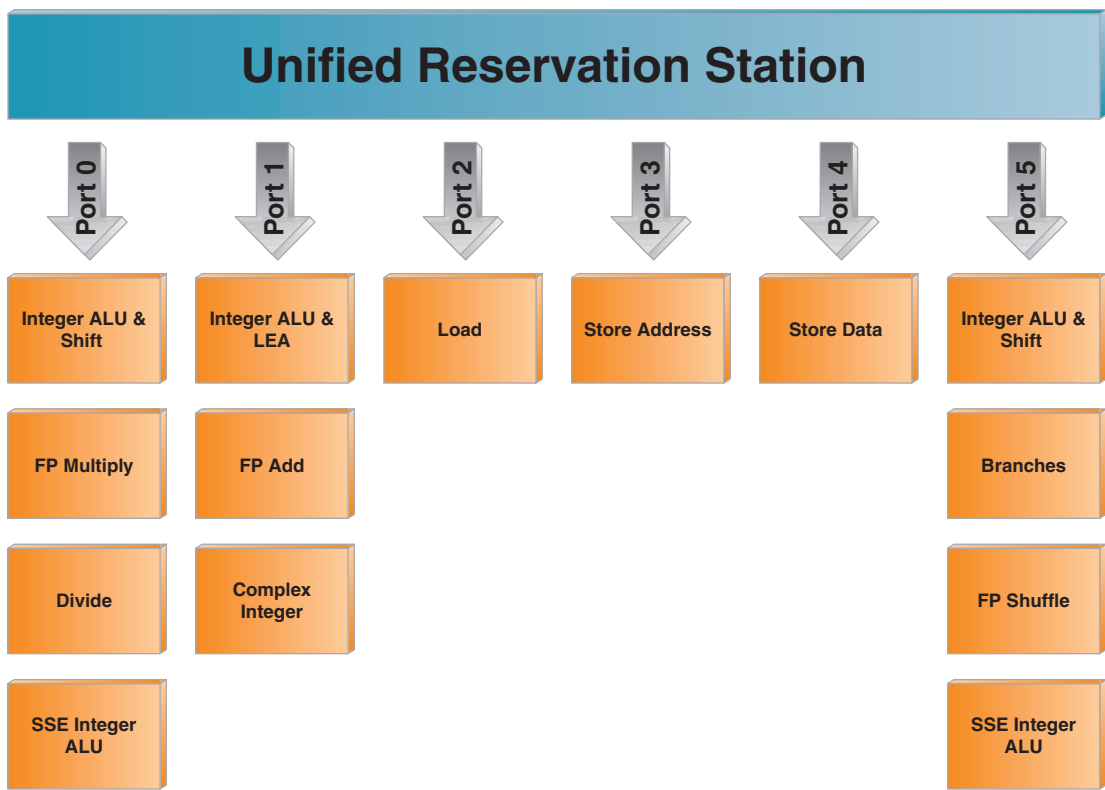


Figure 4: Reservation station and execution units

*“In the prior generation microarchitecture, the new stream of instructions that is fetched on the correct path after a mispredicted branch was not allowed to allocate into the ROB, until the mispredicted branch in question was retired.”*

Another major performance improvement made in this microarchitecture generation was to reduce the cost of mispredicted branches. Mispredicted branches still have to flush instructions from the wrong path from the pipeline and cause a new stream of instructions to be fetched, so the minimum latency impact from a mispredicted branch is not affected. However, in the prior generation microarchitecture, the new stream of instructions that is fetched on the correct path after a mispredicted branch was not allowed to allocate into the ROB, until the mispredicted branch in question was retired. In this generation of microarchitecture, we remove that restriction and allow the new stream to allocate and execute without regards to whether the mispredicted branch in question has retired. By removing this limitation, significant latency can be saved in certain cases. For example, if a load operation misses all on-die caches and has to read memory, it may take hundreds of cycles to complete. Following that load may be a branch instruction that is not dependent on the load. If that branch mispredicts, in the prior generation microarchitecture, the new stream of instructions could not execute until the branch retired, which meant that the long latency load also had to complete, effectively creating a dependence between the branch and the load. In this new technology generation, no such dependence is created. This allows the new stream of instructions to start executing in parallel with the completion of the load that missed the caches.



## Memory Subsystem

The memory cluster is responsible for handling load and store operations. The most noticeable change in the memory subsystem is in the cache hierarchy. In the prior generation of microarchitecture, the first level of cache was private to a core while the second level of cache was shared between pairs of cores. In this generation, we moved to having two levels of cache that are dedicated to the core and a third level that is shared between cores and is located in the Uncore.

The first-level data cache remains 32 kilobytes in size and continues to use a writeback policy. Also, we left the cache line size and associativity the same—64 bytes and 8-ways, respectively. The load-to-use latency of the first-level data cache increased from three cycles to four cycles in this generation. We added a new second level, or mid-level, cache (MLC) that is 256 kilobytes in size, also with 64-byte lines and 8-way set associative. It is also a writeback cache. The MLC was designed to achieve high performance through a low, 10-cycle, load-to-use latency. The MLC is a unified cache, holding both instructions and data. Cache misses from both the first-level instruction cache and from the first-level data cache look up the MLC on cache misses.

The rationale for adding a second level of cache dedicated to each core was twofold:

- We decided to move to having a last-level cache shared between all cores. Having such a shared cache allows the entire cache to be used by any subset of the cores, in line with our goal of not penalizing applications that cannot take advantage of all cores. Therefore, we needed the MLC to buffer the last-level shared cache from a high rate of requests coming from all of the cores. This bandwidth buffering also enables greater scalability in core counts, so that as we build products with larger core counts, we do not necessarily need to make any changes to the CPU core.
- Provide a middle latency alternative between the very fast first-level cache and the third-level cache that would be much slower. By adding this cache, we optimize the overall effective latency (weighted latency average) across a wide range of workloads.

Beyond the caching hierarchy, we also updated the Translation Lookaside Buffer (TLB) hierarchy inside the core with the addition of a second-level TLB (STLB). The STLB is 512 entries in size and can hold both instruction-page and data-page translations. The workloads that benefit most from this structure have large data and code working sets, for example, sets often seen in high-performance computing and database workloads. By adding the STLB, numerous page walks can be eliminated, resulting in a performance gain—as page walks can be costly operations.

*“In this generation, we moved to having two levels of cache that are dedicated to the core and a third level that is shared between cores.”*

*“We added a new second level, or mid-level, cache (MLC) that is 256 kilobytes in size.”*

*“Cache misses from both the first-level instruction cache and from the first-level data cache look up the MLC on cache misses.”*

*“We also updated the Translation Lookaside Buffer (TLB) hierarchy inside the core with the addition of a second-level TLB (STLB).”*

*“By adding the STLB, numerous page walks can be eliminated, resulting in a performance gain.”*

*“This microarchitecture (Westmere family) also adds support for 1GB pages.”*

In addition to the STLB, the 32nm version of this microarchitecture (Westmere family) also adds support for 1GB pages. Prior to this technology generation, the page sizes supported were 4KB, 2MB, and 4MB. With the appropriate operating system support, larger page sizes offer the opportunity for higher performance by again reducing the number of page walks that are needed to read a given page. Table 2 details each level of the TLB hierarchy.

	# of Entries
<b>first Level Instruction TLBs</b>	
Small Page (4k)	128
Large Page (2M/4M)	7 per thread
<b>first Level Data TLBs</b>	
Small Page (4k)	64
Large Page (2M/4M)	32
<b>New 2nd Level Unified TLB</b>	
Small Page Only	512

**Table 2:** TLB Hierarchy Description

Another set of memory cluster optimizations made in this microarchitecture generation revolved around unaligned memory accesses to the first-level data cache. Specifically, two optimizations were made to improve performance on these types of operations.

The first optimization relates to 16-byte SSE vector load and store operations. The SSE architecture defines two forms of 16-byte memory accesses, one that can be used when the memory location being accessed is aligned on a 16-byte boundary (for example, the MOVDQA instruction), and a second form that allows any arbitrary byte alignment on these operations (for example, the MOVDQU instruction). The latter case is important because compilers often have to be conservative when generating code; they cannot always guarantee that a memory access will be aligned on a 16-byte boundary. Prior to this, the aligned form of these memory accesses had lower latency and higher throughput than the unaligned forms. In this new microarchitecture generation, we optimized the 16-byte unaligned memory access instruction to have the same latency and throughput as the aligned version, for cases that are aligned. By doing this, compilers are free to use the unaligned form in all cases and not have to worry about checking for alignment considerations.

*“We optimized the 16-byte unaligned memory access instruction to have the same latency and throughput as the aligned version, for cases that are aligned.”*

The second optimization in the first-level data cache is for memory accesses that span a 64-byte cache line. As vectorization becomes more pervasive, we are seeing the need to improve the performance on these operations, as the compiler, again, cannot guarantee alignment of operations in many cases. With this generation of processors, we took significant steps to reduce the latency of these cache-line split accesses through low-level, microarchitectural techniques.

*“We took significant steps to reduce the latency of these cache-line split accesses through low-level, microarchitectural techniques.”*

The final area of optimization in the memory subsystem was synchronization latency. As more threads are added to systems, and as software developers recognize that there are significant performance gains to be had by writing parallel code, we want to ensure that such code is not severely limited by the need to synchronize threads. To this end, we have been working to reduce the latency of cacheable LOCK and XHCG operations, as these are the primitives used primarily for synchronization. In this processor generation, we reduce both the latency and side-effect costs of these operations. The significant reduction in latency was achieved through careful re-pipelining. We also worked to minimize the pipeline stalls that these synchronization operations caused. Prior to this technology, all memory operations younger than the LOCK/XCHG had to wait for the LOCK/XCHG to complete. However, there was no architectural reason that we had to be this conservative, so this time around, we allow younger load operations to proceed even while an older LOCK/XCHG is still executing, as long as the other instructions do not overlap with the LOCK/XCHG in the memory they are accessing. This improvement does not show up in the latency of the LOCK/XCHG instructions, but it does show up in overall performance by allowing subsequent instructions to complete faster than in previous processors.

## Intel® Hyper-Threading Technology

Even with all of the techniques previously described, very few software applications are able to sustain a throughput near the core's theoretical capability of four instructions per cycle. Therefore, there was still an opportunity to further increase the utilization of the design. To take advantage of these resources, Intel Hyper-Threading Technology, which was first implemented in the Intel Pentium® 4 processor family, was re-introduced to improve the throughput of the core for multi-threaded software environments in an extremely area- and power-efficient manner.

The basic idea of Intel Hyper-Threading Technology is to allow two logical processors to execute simultaneously within the core. Each logical processor has its own software thread of execution state. Because a single software thread rarely fully exploits the peak capability of the core on a sustained basis, the central idea was that by introducing a second software thread, we could increase the overall throughput of the CPU core. This design yields an extremely efficient performance feature, since a minimal amount of hardware is added to provide this performance improvement.

Several key philosophies went into the design of Intel Hyper-Threading Technology in this processor generation:

- When Hyper-Threading Technology is enabled, but only a single software thread is scheduled to a core, the performance of that thread should be basically identical to when Intel Hyper-Threading Technology is disabled. This behavior was achieved by making sure that resources that are shared or partitioned between logical processors can be completely used when only one thread is active.

*“We have been working to reduce the latency of cacheable LOCK and XHCG operations, as these are the primitives used primarily for synchronization.”*

*“Prior to this technology, all memory operations younger than the LOCK/XCHG had to wait for the LOCK/XCHG to complete.”*

*“Very few software applications are able to sustain a throughput near the core's theoretical capability of four instructions per cycle.”*

*“Intel Hyper-Threading Technology, which was first implemented in the Intel Pentium® 4 processor family, was re-introduced.”*

*“Because a single software thread rarely fully exploits the peak capability of the core on a sustained basis, the central idea was that by introducing a second software thread, we could increase the overall throughput of the CPU core.”*

*“We do not unnecessarily constrict throughput by ping-ponging between the threads unless both threads are active and have work to do.”*

*“By partitioning structures, we guarantee each thread a set of resources to achieve reasonable performance.”*

- There are points in the processor pipeline where the logic has to either operate on one thread or the other. When faced with these arbitration points, design the selection algorithm to achieve fairness between the threads. Therefore, if both threads are active and have work to do, the arbitration points will use a “ping-pong” scheme to switch between the two threads on a cycle-by-cycle basis.
- When faced with an arbitration point in the pipeline, if only one thread has work to do, allow that thread to get full bandwidth. It is often the case that at an arbitration point in the pipeline, only one thread has work to do. If that case occurs, we designed the core to give full bandwidth to that thread until the other thread has work to do. In other words, we do not unnecessarily constrict throughput by ping-ponging between the threads unless both threads are active and have work to do.

Given these principles, decisions still had to be made on how various structures are handled in the face of Intel Hyper-Threading Technology. Four different policies were available for managing structures and are summarized in Table 3:

- **Replicated:** For structures that were replicated, each thread would have its own copy of the structure. If only a single thread were active in the core, the structures for the other thread would be unused. The most obvious example of this type of scheme is in the case of the architectural state. Each thread must maintain its own architectural state so the structures that hold that state, such as the retired RF, must be replicated. Cases that are replicated represent a real area and power cost for Hyper-Threading Technology, but the number of cases where structures are replicated is limited.
- **Partitioned:** For structures that are partitioned, when two threads are active, each one is able to access only half of the structure. When only a single thread is active, we make the entire structure available to that thread. Prime examples where we partitioned structures are the various control buffers: reorder, store, and load. By partitioning structures, we guarantee each thread a set of resources to achieve reasonable performance. Moreover, partitioning of structures typically comes at only a small cost for managing the structures, but without any increase in the area or the power of the structures themselves.
- **Competitively shared:** For structures that are competitively shared, we allow the two threads to use as much of the structure as they need. The best example of this scheme is the caches on the processor. In the caches, we do not limit a thread to a percentage of the cache. Therefore, if two threads are active, we allow one thread to occupy a majority of the cache, if that is what its dynamic program behavior demands.
- **Unaware:** Finally, there are parts of the core that are completely unaware that Hyper-Threading Technology exists. The execution units are the best example of this, where the computed result is not affected by which thread is doing the computation.

Policy	Description	Examples
Replicated	Duplicate logic per thread	Register State Renamed RSB Large Page ITLB
Partitioned	Statically allocated between threads	Load Buffer Store Buffer Reorder Buffer Small Page ITLB
Competitively Shared	Depends on thread's dynamic behavior	Reservation Station Caches Data TLB 2nd level TLB
Unaware	No impact	Execution units

**Table 3:** Comparison of Intel® Hyper-Threading Technology Policies

The key arbitration points in the pipeline that we needed to consider when implementing Hyper-Threading Technology are these:

- **Instruction Fetch:** We support only one read of the Instruction Cache per cycle; therefore, we need to arbitrate between the two threads to decide which one gets to read the cache in any given cycle.
- **Instruction Decode:** In any given cycle, the bytes from only one thread can be decoded into  $\mu$ ops.
- **Allocation:** We can allocate resources (ROB entries, store/load buffer entries, RS slots) only to a single thread each cycle.
- **Retirement:** Similarly, when we retire  $\mu$ ops and reclaim their resources, we can only work on a single thread in a given clock cycle.

Note that the RS is not a point of arbitration between the threads. The RS schedules  $\mu$ ops to the execution units without regard to which thread they belong to. Instead, its decisions are based upon which instructions are ready to execute and then choosing the “oldest,” as measured by when they allocated into the RS.

The overall performance benefit of Hyper-Threading Technology varies depending on the nature of the software application in question. Some applications do not benefit from this technology. For example, applications or benchmarks, like Streams, that are memory bandwidth-bound when not using Hyper-Threading Technology will likely not see a performance benefit when Hyper-Threading is enabled, because no additional memory bandwidth

*“The overall performance benefit of Hyper-Threading Technology varies depending on the nature of the software application in question.”*

*“Citrix Systems has shown that Hyper-Threading Technology provides a 57% increase in the number of users that could be consolidated onto a server.”*

*“New instructions improve overall power efficiency by completing in a single instruction a task that previously was handled by multiple instructions.”*

*“Seven new instructions were branded as SSE4.2.”*

is made available by using this technology. Another category of applications that will not benefit are those that already operate near the peak throughput of four instructions per clock or near the peak of another execution's resource, since that means that no idle resources can be exploited by the other thread. Linpack\* is a primary example of a benchmark that falls into this category [3].

However, since there are very few workloads that operate near the CPU's peak IPC capabilities, there is ample opportunity for software to benefit from Intel Hyper-Threading Technology. Various reports on real-world workloads have shown that the performance boost can vary significantly based on the workload. Facebook has shown 15% higher throughput on one of their production workloads due to the use of Hyper-Threading Technology [4]. Citrix Systems has shown that Hyper-Threading Technology provides a 57% increase in the number of users that could be consolidated onto a server [5]. Additionally, measurements on the SpecMPI2007\* benchmark suite, a high-performance computing proxy, have shown gains ranging from -2% to 35% depending on the specific benchmark with an overall 9% impact on the final benchmark score [6].

## New Instructions and Operations

Another means of improving performance in a power-efficient manner is through the addition of new instructions that software can exploit. New instructions improve overall power efficiency by completing in a single instruction a task that previously was handled by multiple instructions. We can achieve higher performance at lower power through the addition of these new instructions. Of course, exploiting the benefit of the new instructions requires software to be rewritten or re-compiled. Pre-existing software will not see a benefit from these instructions.

New instructions were introduced in both the 45nm (Nehalem) and 32nm (Westmere) versions of the core. The instructions were chosen because they offered significantly higher performance on critical and common operations in computing today.

In the 45nm version of the Nehalem microarchitecture, seven new instructions were added for performance, as well as new virtualization functionality and new timestamp counter (TSC) functionality. Full specification details on these instructions can be found in *Intel® 64 and IA-32 Architectures Software Developer's Manual Volumes 2A and 2B* [7]. The seven new instructions were branded as SSE4.2 and comprise the following:

- PCMPSTRI, PCMPSTRM, PCMPISTRI, PCMPISTRM: These four instructions accelerate manipulation operations on text strings. Such operations can be helpful, for instance, when parsing a stream of XML or doing regular expression comparisons. These instructions offer powerful



capabilities such as identifying substrings within a string, finding a specific character in a string, or finding a range of characters, such as finding all numbers, in a string. The instructions operate on the 128-bit SSE register set, allowing for processing of 16 bytes at a time per operation. An example usage has shown roughly a 25% average throughput improvement on parsing XML by use of these new instructions [8].

- **POPCNT:** This instruction returns the number of 1's that are set in an integer register. This is a common operation in recognition and search algorithms.
- **CRC32:** This instruction accelerates the computation of a Cyclic Redundancy Check (CRC) by using the iSCSI polynomial, which is used in several networking protocols [9]. Additionally, this type of operation can also be used to provide a fast hash function.
- **PCMPGTQ:** This instruction compares two SSE integer vector registers and checks for a "greater than" condition.

Two other architecture-visible features were added to the Nehalem microarchitecture. Specifically, support was added for a constant time stamp counter, which aids timing across deep sleep states. Additionally, enhanced page table (EPT) support was added that allows virtualization vendors to avoid shadow page tables, removing a source of performance overhead when running in virtualized environments. Coupled with EPT, two instructions (INVVPID, INVEPT), were added to the architecture to support invalidations of EPT translation caches.

In the Westmere architecture, seven new instructions were added that focus on providing instructions to accelerate the performance of cryptographic operations. These operations are prevalent today in computing in areas such as Web commerce and in setting up secure transactions between a host and a client. The seven new instructions are as follows:

- **AESENC, AESENCLAST, AESDEC, AESDESCLAST, AESKEYGENASSIST, AESIMC:** This collection of six instructions can be used together to accelerate encryption and decryption by using the Advanced Encryption Standard (AES) [10]. With these instructions, performance gains three to ten times better than previous-generation technology performance have been achieved on commercial software [11].
- **PCLMULQDQ:** This instruction performs a carryless multiply operation. A Galois (binary) field multiply consists of the carryless multiply followed by a reduction. Galois fields are common in cryptographic operations, such as AES as well as cyclic redundancy checks, elliptic curve cryptography, and error correcting codes. AES Galois Counter Mode is probably the most widely known application of PCLMULQDQ [12].

*"This instruction accelerates the computation of a Cyclic Redundancy Check (CRC) by using the iSCSI polynomial, which is used in several networking protocols [9]. Additionally, this type of operation can also be used to provide a fast hash function."*

*"In the Westmere architecture, seven new instructions were added that focus on providing instructions to accelerate the performance of cryptographic operations."*

*"Galois fields are common in cryptographic operations, such as AES as well as cyclic redundancy checks, elliptic curve cryptography, and error correcting codes."*

- Also in the Westmere processor architecture, the page table support, when running in virtualized environments, was further enhanced to support real-mode guests. This design substantially reduces the time required to boot virtual machines through BIOS as well as reduces the time to execute device option ROMs.

These new instructions focus on accelerating specific usage models for critical and common tasks. Going forward, we will continue to look for opportunities for such targeted acceleration opportunities, as well as more general-purpose instruction-set additions, such as the upcoming Intel Advanced Vector Extensions (AVX). Full details on AVX are available in the *Intel Advanced Vector Extensions Programming Reference* [13].

## Segment-Specific Features

In this section, we detail a few examples of new segment-specific features, and we discuss some of the tradeoffs that were made in their design.

### Physical and Virtual Addressing

The client segment platforms, mobile and desktop, typically support a maximum of 4GB-8GB of main memory; large server systems need to support very large memory capacities, on the order of 100 GBytes or more. This wide range of requirements provides a very stark set of tradeoffs. Additional address bits needed for large memory systems have little value in the client segments, but they do consume extra power. In addition to increasing power consumption and adding area, additional address bits can cause stress on the timing of critical paths in the machine, such as address generation for cache look-ups. To strike a balance between how many address bits to add, we settled on a modest two physical address bits over the Core 2 microarchitecture, but we did not increase the virtual address bits from the number in previous-generation technology. Our rationale was that physical address bits are critical to enable certain large systems, and that the number of virtual address bits was already adequate to enable operating systems for the server systems that needed to be built.

*“We settled on a modest two physical address bits over the Core 2 microarchitecture, but we did not increase the virtual address bits from the number in previous-generation technology.”*

*“Because the Westmere core would be used mostly in the same platforms as the Nehalem core, we opted against adding additional addressing bits in the Westmere version.”*

### Reliability Features

To satisfy large server systems, we enhanced the reliability features found in the prior-generation architecture. Reliability features come in two major categories: enhancements to the Machine Check Architecture (MCA) and enhancements to the design and microarchitecture. To achieve the targets for “soft error rates” we set to enable scaling to the socket and processor counts needed. For both of these feature additions, it is important to keep in mind that the cores were designed to meet the requirements of high-end, multi-processor servers with 8-10 cores per socket and up to 8 sockets. Clearly the requirements set by these high-end server systems far exceed the ones that are set by the client segments.



For machine-check enhancements, we added more reporting capabilities for various failure modes to increase coverage and insights into state for diagnostic software. The cost of MCA features is not high in power, area, or design effort; however, due to its complexity it is a fairly substantial cost against the project's validation budget. A reasonable balance of added features included looking at the tradeoffs between value and validation cost.

We also added Soft Error Rate (SER) features to meet scalability requirements. Deciding what SER features to add was arrived at by carefully modeling possible failure behavior and also looking at a careful modeling of possible and critical failure behaviors to determine the protection of specific micro-architectural assets. The SER features we added included items such as parity on various structures. To further add high-end server reliability, we also added a mode where the first-level data cache operates at half the size with additional data reliability protection enabled. This cache mode is a customer-enabled option for the expandable server (EX) product line, where customers are willing to trade some performance for additional reliability.

### Power Features

Low average power is important to battery life for traditional productivity workloads and for battery life for specific usages, like video, DVD, or streaming media playback usages. For this generation of processors, we modeled the impact of both of these usage cases when making feature decisions. At a high level, there are two aspects to achieving a good average power behavior: 1) being able to run at a low power for a minimum performance point, and 2) being able to transition in and out of low power states quickly, making sure that the power consumed in these low power states is as low as possible.

Achieving a low-power operating point is valued by the server segment, where it enables more cores to be aggregated in a socket. To achieve this operating point, we typically work on lowering the power consumption in general and specifically lowering the minimum voltage at which the core can operate (MinVCC). Having a low MinVCC provides a very efficient way to reduce the power due to the cubic scaling we discussed earlier. Due to the cubic benefit from lowering the operating voltage, even small changes have very beneficial effects. There is typically an area tradeoff involved in the device sizing needed to achieve a low MinVCC, which is a factor to take into account for the overall product cost.

Efficient low-power states, also known as C-states, and quick entry and exit into and out of those states are features that are fairly uniquely valued by the client mobile segment. At a high level, the lower the power is for a C state, the more power we save. Moreover, the faster we can enter and exit these states, the more often we can utilize them without harming overall system performance characteristics. We added new low-power states, made the power in those states lower, and optimized entry and exit latencies.

*“For machine-check enhancements, we added more reporting capabilities for various failure modes to increase coverage and insights into state for diagnostic software.”*

*“Cache mode is a customer-enabled option for the expandable server (EX) product line, where customers are willing to trade some performance for additional reliability.”*

*“Efficient low-power states, also known as C-states, and quick entry and exit into and out of those states are features that are fairly uniquely valued by the client mobile segment.”*

*“We added new low-power states, made the power in those states lower, and optimized entry and exit latencies.”*

## Integration and Scalability

We also added features to make it easier to integrate cores into a large scalable processor and to better utilize the power for such multi-core systems.

To achieve better performance scaling in systems with high core counts, we added the previously mentioned MLC to reduce the bandwidth requirement of the core-Uncore interface. The benefit of the MLC is two-fold. It adds performance by, on average, improving the perceived latency of memory accesses through its low latency and by not overtaxing the bandwidth at the shared last-level cache in the Uncore.

Additionally, the Nehalem generation of processors supports buffers that allow the core to independently run at a different frequency than the Uncore they are attached to. This is a key power-efficiency feature where we can adjust the frequency of the core and the Uncore individually to operate at the best power/performance efficiency point. For example, we can set the Uncore frequency to match the desired performance of external interfaces, such as memory speed, while we let the cores independently run at the frequency that is demanded by the task, as asked for by the operating system.

*“We can set the Uncore frequency to match the desired performance of external interfaces, such as memory speed, while we let the cores independently run at the frequency that is demanded by the task.”*

*“The power to the cores can be completely turned off when the core is not being used.”*

*“The frequency of active cores can scale up if there is power headroom.”*

Building on the ability to dynamically run the cores at a different frequency, and isolating a core from the Uncore, this generation of microarchitecture also supports power gates, where the power to the cores can be completely turned off when the core is not being used. This feature is important for mobile average power as it prevents draining of the battery when the core is doing nothing. Power gates are also key enablers for Intel Turbo Boost Technology which dynamically allows active cores to run at higher frequencies. Fundamentally, the turbo boost feature allows products built with this generation of cores to dynamically use the full-power envelope without artificially limiting the performance of the cores. With this technology, the frequency of active cores can scale up if there is power headroom. Power gates help this by eliminating the power overhead of idle cores. However, even when all cores are active, Intel® Turbo Boost Technology can kick in and increase the frequency of all cores if the workload being run is not power intensive. This technology helps satisfy the philosophy of not penalizing customers whose software does not take advantage of all cores.

## Conclusion

The Nehalem and Westmere architecture is designed to work efficiently for a wide range of operating points, from mobile to high-end server. High-value, segment-specific features were added, while at the same time taking care that the power overhead could be absorbed by all segments. Intel achieves the needed increase in efficiency of both power and performance through appropriate buffer size increases and more intelligent algorithms. These changes were made without fundamentally affecting the pipeline latencies of the baseline design. The changes deliver exciting features and a superior performance increase inside equal or lesser power envelopes.

## References

- [1] Harikrishna Baliga, Niranjana Cooray, Edward Gamsaragan, Peter Smith, Ki Yoon, James Abel, Antonio Valles. “The Original 45-nm Intel® Core™ 2 Processor Performance,” *Intel Technology Journal*. Available at <http://download.intel.com/technology/itj/2008/v12i3/paper4.pdf>
- [2] David L Hill, Derek Bachand, Selim Bilgin, Robert Greiner, Per Hammarlund, Thomas Huff, Steve Kulick, and Robert Safranek. “The Uncore: A Modular Approach to Feeding the High-Performance Cores,” *Intel Technology Journal*, Volume 15, Issue 01, 2011.
- [3] Pawel Gepner, Michal F. Kowalik, David L. Fraser, Kazimierz Wackowski. “Early Performance Evaluation of New Six-Core Intel® Xeon® 5600 Family Processors for HPC,” *ispdc*, pp. 117–124, 2010 *Ninth International Symposium on Parallel and Distributed Computing*, 2010.
- [4] *Real-World Application Benchmarking*. Available at [http://www.facebook.com/note.php?note\\_id=203367363919](http://www.facebook.com/note.php?note_id=203367363919)
- [5] *Nehalem and XenServer Raise the Bar for XenApp Performance*. Available at <http://community.citrix.com/pages/viewpage.action?pageId=73564465>
- [6] Bugge. H. “An evaluation of Intel’s core i7 architecture using a comparative approach,” *Computer Science—Research and Development*, Vol. 23(3–4), 203–209, 2009.
- [7] *Intel® 64 and IA-32 Architectures Software Developer’s Manual Volumes 2A and 2B*.
- [8] Chris Newburn. “High Performance XML Processing with Intel SSE4.2 Hardware and Software Algorithms,” *XML in Practice*, 2008.
- [9] Vinodh Gopal, et al. “Fast CRC Computation for iSCSI Polynomial Using CRC32 Instruction.” Available at <http://download.intel.com/design/intarch/papers/323405.pdf>
- [10] Gueron, S. “Advanced encryption standard (AES) instructions set,” *Technical report*, Intel Corporation, 2008. Available at <http://softwarecommunity.intel.com/isn/downloads/intelavx/AES-Instructions-Set WP.pdf>
- [11] Refer to <http://www.tomshardware.com/reviews/clarkdale-aes-encryption,2538-7.html>

- [12] Gueron, S., et al. “Intel Carry-Less Multiplication Instruction and its Usage for Computing the GCM Mode,” *White Paper*, Intel Corporation, 2010.
- [13] *Intel Advanced Vector Extensions Programming Reference. Available at <http://software.intel.com/file/19151>*

## Authors' Biographies

**Martin Dixon** is a Principal Engineer in the Intel Architecture Group, working closely with software partners to develop and enhance the overall instruction set. He received his B.S. degree in Electrical and Computer Engineering from Carnegie Mellon University.

**Per Hammarlund** is a Senior Principal Engineer in the Intel Architecture Group working on microarchitecture development, Intel Hyper-Threading Technology, power efficiency, and performance modeling. Per started at Intel in 1997 working on the Willamette processors (Pentium 4). He received his PhD degree in Computer Science from the Royal Institute of Technology (KTH) in 1996.

**Stephan Jourdan** is a Senior Principal Engineer in the Intel Architecture Group.

**Ronak Singhal** is a Senior Principal Engineer in the Intel Architecture Group working on microarchitecture development, Instruction Set Architecture development, and performance analysis. He joined Intel in 1997 and has worked on multiple generations of processors, starting with the initial Intel Pentium 4 processor. He received his B.S. and M.S. degrees in Electrical and Computer Engineering from Carnegie Mellon University.

## Copyright

Copyright © 2011 Intel Corporation. All rights reserved.

Intel, the Intel logo, and Intel Atom are trademarks of Intel Corporation in the U.S. and other countries.

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

Requires an Intel® HT Technology enabled system, check with your PC manufacturer. Performance will vary depending on the specific hardware and software used. Not available on Intel® Core™ i5-750. For more information including details on which processors support HT Technology, visit <http://www.intel.com/info/hyperthreading>

Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM). Functionality, performance or other benefits will vary depending on hardware and software configurations. Software applications may not be compatible with all operating systems. Consult your PC manufacturer. For more information, visit <http://www.intel.com/go/virtualization>

Requires a system with Intel® Turbo Boost Technology capability. Consult your PC manufacturer. Performance varies depending on hardware, software and system configuration. For more information, visit <http://www.intel.com/technology/turboboost>

# THE UNCORE: A MODULAR APPROACH TO FEEDING THE HIGH-PERFORMANCE CORES

## Contributors

- David L Hill**  
Intel Corporation
- Derek Bachand**  
Intel Corporation
- Selim Bilgin**  
Intel Corporation
- Robert Greiner**  
Intel Corporation
- Per Hammarlund**  
Intel Corporation
- Thomas Huff**  
Intel Corporation
- Steve Kulick**  
Intel Corporation
- Robert Safranek**  
Intel Corporation

## Index Words

- Uncore  
In-Die Interface  
Credit Tokens  
Last-Level Cache Controller  
Crossbar  
Chip Debug Port

*“The Uncore includes all the chip components outside the CPU core: system memory controllers, socket-to-socket interconnects, multi-core memory coherence controller, large shared cache, and the chip-level clocking, power-delivery and debug mechanisms that tie the various on-die components together.”*

## Abstract

The Intel® microarchitecture code name Nehalem and Intel® microarchitecture code name Westmere platforms include a new generation of Intel products spanning Intel® Core™ mobile laptops, to Intel Xeon® multi-processor socket server systems. These products delivered significantly improved performance due to platform repartitioning and a new focus on power efficiency. The platform repartition included introducing the QuickPath Interconnect (Intel® QuickPath Interconnect (Intel® QPI)) architecture and moving the memory controllers from the chipset to the processor sockets. QPI replaced the Frontside Bus (FSB), which had been the interconnect technology for more than a decade in Intel products. The QPI and the integration of the memory controller onto the die provided market-leading memory bandwidth and latency characteristics for the platform. The focus on power efficiency allowed the Nehalem and Westmere platforms to provide significant performance improvements within equal or lesser thermal power envelopes. This article describes the Intel Xeon-5500/5600 Uncore, which is the first Nehalem-Westmere modular multi-core implementation of this new platform repartition and power efficiency focus.

## Introduction

The Intel processor projects, codenames Nehalem and Westmere, are targeted at multiple product segments, spanning single-socket, low-thermal-power mobile clients to high-performance dual- or multi-socket, high-thermal-power servers. The portion of the processor design that ties together the high-performance cores with the platform to span this range of product is called the “Uncore.” The Uncore includes all the chip components outside the CPU core: system memory controllers, socket-to-socket interconnects, multi-core memory coherence controller, large shared cache, and the chip-level clocking, power-delivery and debug mechanisms that tie the various on-die components together.

Building a single design that competitively covers so wide a range of products may seem impossible, but examination of the problem shows the requirements across the product range to be similar and synergistic. For example, the entire range requires the following:

- Power-efficient, scalable implementation of high bandwidth, low-latency memory solutions.
- Aggressive system power management.
- A modest set of high-value platform segment-specific features.

The Nehalem and Westmere Uncore microarchitecture and implementation approach addresses these requirements. From an implementation point of view, the goals were achieved by designing a modular Uncore and core. The modular design supports both efficient design reuse and the ability to proliferate a significant number of products from the same base design.

The Nehalem-Westmere platforms introduced several significant changes to achieve high performance:

- An integrated memory controller. The system memory controllers now reside in the processor sockets and not in the chipset.
- A new high-bandwidth, reliable, full-duplex, point-to-point system interconnect, QPI, that allows the Intel® Xeon® multiple-socket platform topologies to communicate between sockets.
- A modular, more easily scalable, multi-core and cache hierarchy, along with multiple on-die power planes.
- A programmable integrated microcontroller solely dedicated to executing dynamic power control algorithms for the multiple power planes and interfaces.

In concert, these advances allow a very significant range of core counts, cache sizes, and platform-differentiated, power-performance optimizations. In this article we discuss the first of two Uncore base designs that enabled 45nm, 4-core Nehalem and 32nm, 6-core Westmere, dual-socket products, known as the Intel Xeon 5500/5600 Efficient Performance (EP) products.

We divide this article in sections, each one describing the major Uncore micro-architecture components, that together with the CPU cores, constitute the internal elements of a processor socket. Other processor components are discussed in other articles in this issue of the *Intel Technology Journal*.

## Overview and Modular Philosophies

The Nehalem-Westmere processor family consists of a set of client and server processors that were built using a modular die-component design methodology. In the following sections we describe the primary elements and concepts of this design methodology as applied to the Intel Xeon 5500/5600 products.

### Modular Converged-Core

The Nehalem-Westmere Uncore and the CPU core are built up as separate modular design databases. One CPU core design known as a “converged-core” is used across all product segments such that the family of products provides CPU core implementation consistency for the system software and applications. A converged core also enables the design to be used again and again, thereby capitalizing on the many hundreds of person-years it takes to create the complex CPU core, enabling a modular instantiation capability as needed to create multi-core products.

*“The Nehalem-Westmere Uncore and the CPU core are built up as separate modular design databases.”*



*“A flexible interface was defined with a unique ability to cross voltage, frequency, and clock-skew boundaries between core and Uncore.”*

*“The first Uncore design covered mobile client through dual-socket Xeon-EP segments.”*

*“There are eight or more client and server product variations at the time of this writing, all based on modest design alterations of the two base Uncore implementations.”*

### **Core-Uncore Cutline**

Nehalem Architecture, Design, Validation, Debug, and Manufacturing teams coordinated their efforts extensively in defining the details of a core-Uncore cutline. A “black-box” approach to core was taken, with all power-delivery, design-for-testability, clocking, and other physical connection attributes defined as required to instantiate the core “macro” into multiple product implementations within a process generation.

For power optimization, segment flexibility, and schedule adaptation reasons, a flexible interface was defined with a unique ability to cross voltage, frequency, and clock-skew boundaries between core and Uncore with a small latency penalty. The interface is described further in a later section. Nehalem uses this capability extensively in the product family both for statically configuring core frequency and voltage-operating setpoints within platform-dependent constraints, and for dynamically moving operating cores to optimal power-efficiency points based on workload, ambient environment, and power/thermal conditions within freedoms negotiated via sensors, BIOS, and the operating system. The functionality provided by the interface is a key enabler in ensuring the Nehalem- and Westmere-generation platforms can optimize performance within their allotted socket-thermal power envelopes.

### **Nehalem-Westmere Uncore Proliferations**

For the 45nm Nehalem products, we built one converged core and two base Uncores. This work became the basis for the 32nm Westmere products. The first Uncore design covered mobile client through dual-socket Xeon-EP segments. With modest alterations of the base Uncore design, we created proliferation products for embedded and high-end client applications. The second Uncore design covered higher core count, higher socket count, higher reliability, multi-socket server Xeon-7500 segment products. Altogether, there are eight or more client and server product variations at the time of this writing, all based on modest design alterations of the two base Uncore implementations.

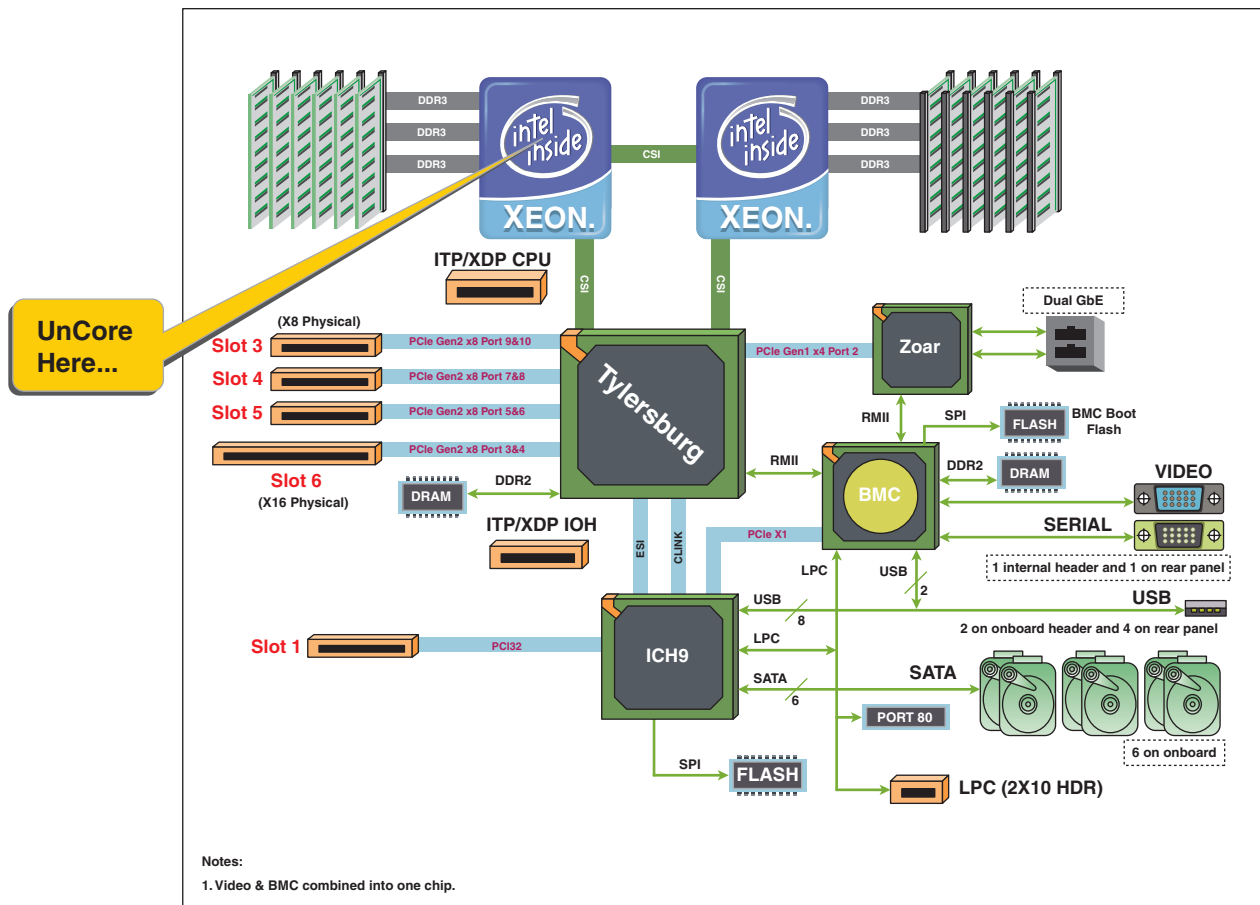
By adhering to our philosophy regarding the 32nm Westmere design and architecture, we ensured a continuation of Nehalem product compatibility, both in the extensions visible to software and at the platform level. The goal of the Westmere designers was to enable a drop-in, socket-compatible refresh of the OEM’s Nehalem platform investment, typically with BIOS-only changes, to a platform designed to Intel specifications.



## Uncore Within the Platform-level View

Figure 1 shows the platform-level view of an example Intel Xeon-5500/5600 system. An Uncore is inside each processor socket, carrying out the function of system interface from this platform-level viewpoint. Functionality that used to be in the chipset has now moved to the processor socket, and it is the new central interface point for the platform. In particular, the platform-visible QPI links and DDR channel implementation are carried out by the Uncore, along with reset, initialization, and voltage-selection sequences that involve the platform components.

*“An Uncore is inside each processor socket, carrying out the function of system interface from this platform-level viewpoint.”*



**Figure 1:** Uncore within the platform view

## Uncore Components and Modularity

In this sub-section, we look at the challenges of the architecture of previous-generation products, examine our modular solution to those challenges, and describe the Xeon 5500/5600 Uncore components.

### Prior-Architecture Challenges

Ten years ago, while designing the first multi-core products, Intel engineers and managers realized that the ability to create powerful, multi-core server

products, uniquely different in performance capability from the cost-sensitive smaller client platforms, was limited by the inherent complexity of assembling and converging a single massive design database for each product, containing all cores and system logic. This “monolithic database” design style had been the standard for many years. Since unique client and server products each required multiple, multi-hundred person teams, and each team needed key critical, senior design expertise, finding and funding such teams on the desired product schedule cadence became a problem. An additional challenge was keeping the large design and architecture teams in sync over the several-year parallel design cycle, such that software-visible, implementation nuances of new processor instructions and design timing changes would not severely affect operating frequencies between the family of client and server production parts.

The key opportunity we pursued was to reuse with minimal changes or instantiate multiple unaltered copies of the highly complex and software-visible CPU core design between client and server products. We thereby achieved both better architectural consistency, and, furthermore, we met the differentiated product requirements, with lower overall implementation effort.

### **Modular Solution**

The Nehalem team addressed this opportunity of re-use by creating a modular database strategy upfront in the concept phase of the project. In Nehalem-Westmere products, cores and Uncores are their own design databases, placed structurally within a die floorplan on separate power and clock domains, such that product design teams can instantiate a converged core “blackbox” style to create the necessary client and server core count product variations. Four identical instances of a Nehalem converged core plus an Uncore became the Xeon 5500 server, while two instances of the Westmere converged core plus a subset of the Xeon Uncore design were optimized to become the mobile client processor, resulting in very satisfactory leverage of key design expertise, due to the modular re-use of cores and Uncore components. Xeon 5600 similarly comprised six instances of a 32nm Westmere converged core, with a 32nm Uncore altered modestly in feature set from the Xeon-5500 Uncore. This methodology resulted in multiple products that covered a greater range of power-performance capabilities being designed in less time than previously achieved. The modularity also allowed parallel core and Uncore teams to focus on their respective definition and implementation objectives.

### **Xeon-5500/5600 Uncore Components**

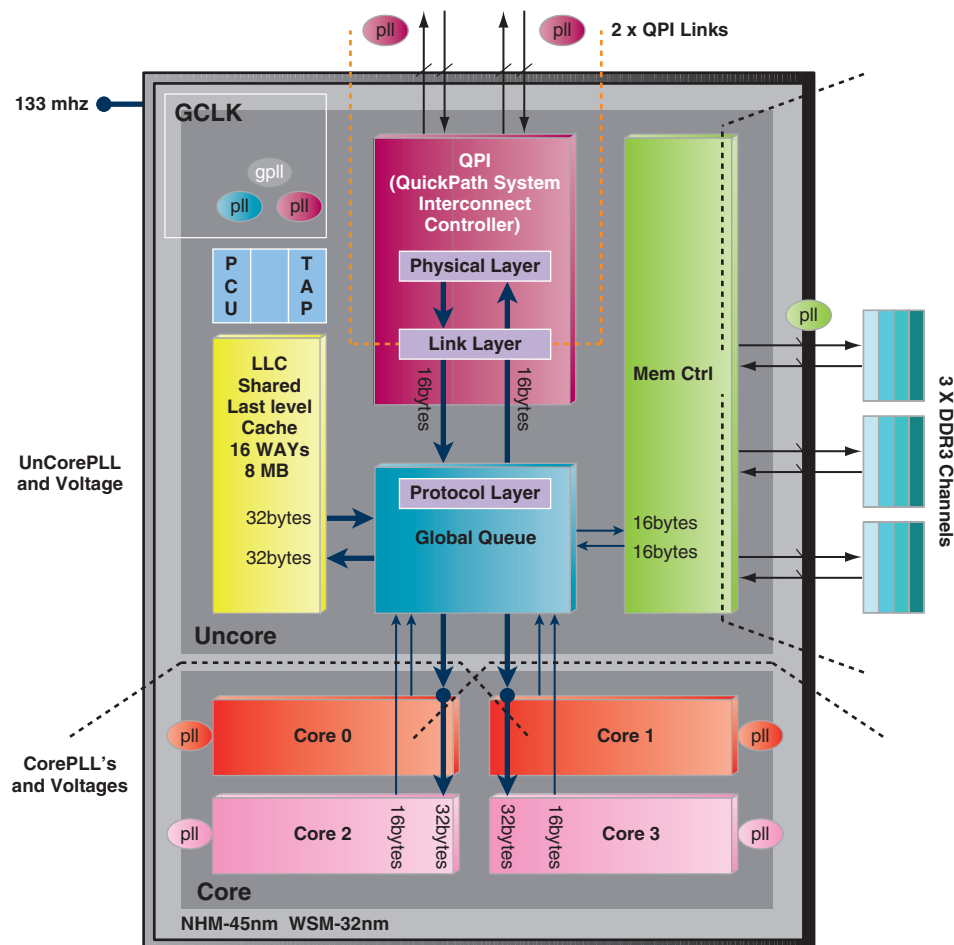
The Xeon-EP 5500/5600 Uncore design database comprises eight major components:

- A five-way intelligent crossbar called a Global Queue (GQ) which operates the Last Level Cache (LLC), a QPI protocol layer, and on-die, multi-core interfaces.
- An 8-12MB shared cache called the LLC.

- The QPI Link-level and Physical (PHY) layer logic.
- DDR3 Memory Controllers (MC).
- A micro-programmable Power Control Unit (PCU).
- Synchronous Clock-generation infrastructure (CLK).
- Chip Analog I/O (QPI, DDR and Misc I/O).
- Chip-level debug equipment (Jtag TAP).

We describe the GQ/QPI, LLC, and MC components later in this article. The PCU and power-management features are described in the power-management article in this issue of the *Intel Technology Journal* [1].

Figure 2 shows the components and rough plumbing of the Intel Xeon-EP 5500 processor component.



**Figure 2: Four Intel® Xeon®-5500 cores plus Uncore rough plumbing**

*“Each QPI link is in effect a 2-byte wide payload in each direction, that may run at speeds of up to 6.4GHz.”*

*“Cores have no direct connections to other cores.”*

*“The maximum Uncore frequency is 3.2GHz.”*

*“QPI and DDR are each running at frequencies that are derived from their native interface operating frequencies.”*

## Major Interfaces

The processor sketch in Figure 2 has socket-level pinouts for two sets of QPI interfaces, one of which connects to the sibling processor socket in a two-socket platform or dual-processor (DP) platform, and the other of which connects to the Tylersburg I/O Hub socket (as shown in Figure 1). Each QPI link is in effect a 2-byte wide payload in each direction, that may run at speeds of up to 6.4GHz. There are also socket-level connections for three industry-standard DDR3 channels, each of which is an 8-byte wide, bi-directional channel operating at speeds of up to 1.3GHz.

Within the die, there are major communication paths between the GQ and each core, the LLC, the MC, and the QPI controller. The GQ is the sole interface to each core, and it acts as a queued “crossbar” between the mentioned components. Cores have no direct connections to other cores. The GQ interfaces operate at a fixed frequency and voltage that are fused into the product at manufacturing; the actual values are dictated by the segment requirements for power and performance. The maximum Uncore frequency is 3.2GHz. While the GQ operates at a per-product fixed frequency, it has a unique, low-latency variable voltage and frequency interface mechanism (described in a subsequent section) to the CPU cores that enables the PCU to change core voltage and frequency operating points dynamically in 133MHz increments. Elements of this interface mechanism are also used in the Uncore design to interface between GQ and QPI or GQ and DDR. QPI and DDR are each running at frequencies that are derived from their native interface operating frequencies. The PCU coordinates initialization of the various interface frequencies and the dynamic frequency and voltage adjustments for cores. The GQ crossbar interfaces are either 16 bytes- or 32 bytes-wide, operating at the GQ clock rate, as noted in Figure 2.

## Global Queue (GQ) Component

This section describes the Global Queue: a five-way intelligent crossbar that interconnects and operates the LLC, the QPI protocol layer, and on-die multi-core interfaces.

## Voltage and Frequency Adjustable Interfaces

A significant piece of Nehalem-Westmere power-performance improvement comes from the ability to both statically and dynamically change the frequency and voltage operating points of the core or Uncore components. As mentioned previously, the Uncore operates at a fixed voltage and frequency, but that voltage and frequency differs depending on client or server product SKU. The client mobile parts and low-wattage servers, for example, operate the Uncore at lower frequencies and voltages than do the higher-wattage server parts, in order to optimize power at the expense of peak performance.

The Uncore team created a low-latency interface mechanism to allow cores to dynamically change the operating point to frequencies lower, equal to, or higher than the fixed Uncore operating frequency. The interface is referred to as In-die Interface (IDI), and it is key to the modular core strategy.

The IDI has a fair amount of implementation flexibility. In the Nehalem and Westmere generation, the IDI has primary 16-byte datapaths running from each core to Uncore (C2U), and a 32-byte datapath from Uncore to core (U2C). Each core has an IDI, with the multiple core IDIs converging at the GQ queued crossbar. At the boundary of the core, the C2U IDI jumps from the core clock and voltage domain to the Uncore clock and voltage domain (and vice versa for U2C paths) by using a small queue-like structure and voltage-level shifters. The queue structure allows an adjustable amount of clock skew to be tolerated between the cores and Uncore, and also allows the core frequency to be dynamically adjusted up or down in 133mhz increments.

The PCU and various sensor mechanisms on-die, along with the IDI ability to jump voltage and frequency domains with low latency, enables the Turbo Boost Technology mechanism. This mechanism allows actively operating cores to opportunistically run at higher than standard frequency, thereby maximizing power-performance within limits of the platform-defined socket power envelope. The IDI also enables the PCU to negotiate cores running at lower voltage and frequency points, or to completely turn off a core to conserve power when the operating system determines workloads are less demanding, and when the user is willing to operate at those power-conserving system states.

Allowing a subset of cores to run at higher frequency, up to the processor power limitations, is a key enabler for improving performance on single-threaded or lightly-threaded workloads, while the lower operating voltages and frequencies in less demanding compute scenarios reduce system energy consumption and extend battery life for mobile usages. With the use of the IDI, the Uncore provides a key mechanism that enables Nehalem and Westmere products to have well-balanced performance characteristics across a wide operating range.

### Credits

GQ and IDI use a credit-based interface approach. This approach means a core cannot make an IDI request to GQ (asking for memory or I/O service for example) unless the core has a credit available to use a GQ request queue entry. The crediting scheme allows deterministic flow control backpressure for correct operation with the frequency jump mechanics to ensure the cores and Uncore do not overrun one another's resources. Credited GQ and QPI interfaces and the resulting smooth backpressure on aggressive request sources are an important part of the Uncore designers' approach to maintaining low latency under heavily loaded conditions.

### Cache Controller

The GQ also acts as the LLC controller. Requests entering the GQ headed to the LLC are from the core IDI links and from the external pin QPI interfaces. The GQ itself is also a requester to the LLC, since the GQ manages the fills of new data and evictions of old cache data.

The cache controller function is latency-sensitive, as the core execution and code-fetch pipelines cannot maintain throughput if they are waiting for

*“In the Nehalem and Westmere generation, the IDI has primary 16-byte datapaths running from each core to Uncore (C2U), and a 32-byte datapath from Uncore to core (U2C).”*

*“The PCU and various sensor mechanisms on-die, along with the IDI ability to jump voltage and frequency domains with low latency, enables the Turbo Boost Technology mechanism.”*

*“A core cannot make an IDI request to GQ (asking for memory or I/O service for example) unless the core has a credit available to use a GQ request queue entry.”*

*“The most recent accessor of shared data owns the job of forwarding the F-state copy.”*

*“The LLC keeps a bit for each local CPU core with each cache line. The GQ uses the bit to keep track of which internal cores might have cached copies of a system address.”*

*“This policy greatly reduces the number of snoop disruptions to core execution pipelines and lowers the latency by as much as 30% for most LLC cache lookup accesses. This is because GQ can resolve the address coherency simply by looking up the address in LLC and by checking the GQ queue entries for same address operations in-flight.”*

responses from the large cache or memory subsystem. To facilitate low latency, the GQ has request age- and priority-based schedulers as well as forward-progress mechanisms to determine which of the multiple request sources should be serviced next in the LLC.

The GQ supports the QPI MESIF protocol cache states. “MESIF” are the well-known Modified, Exclusive, Shared, Invalid states, plus a new state called Forwardable. F-state is equivalent to a system Shared state, except the F-state identifies which caching agent in the system is responsible for forwarding a shared cache-line copy, if a new requester appears. The most recent accessor of shared data owns the job of forwarding the F-state copy. This F-state support reduces latency, because a cached copy may be supplied to a new requester. The prior generation of Intel architecture required a shared copy to come from system memory at a higher latency cost.

#### Cached Memory Coherency

The GQ uses a single point in the cache controller pipeline to resolve conflicts for multiple accesses within the socket to the same physical address. Because all cacheable request sources come through the GQ for access to an address, this is a convenient location to carry out the coherency checks. The LLC keeps a bit for each local CPU core with each cache line. The GQ uses the bit to keep track of which internal cores might have cached copies of a system address. Using these per-core bits, and an inclusive-caching policy, the GQ is able to rapidly determine whether any snoops to internal cores are required in order to allow a next user of any system address. The inclusive policy means that the GQ ensures that any system cache line address that might be cached in a core, also appear as a tag-directory entry in the LLC. This policy greatly reduces the number of snoop disruptions to core execution pipelines and lowers the latency by as much as 30% for most LLC cache lookup accesses. This is because GQ can resolve the address coherency simply by looking up the address in LLC and by checking the GQ queue entries for same address operations in-flight. To minimize overall latency, the GQ checks core-generated requests for system address map and preliminary DRAM channel map properties in parallel with the LLC lookup latency, such that immediate dispatch can occur to a QPI or local DRAM controller port once LLC HIT/MISS becomes known.

The GQ also uses the per-core bits to maintain an internally shared state. If multiple cores internally want to share a cache line that is exclusively owned by the socket, the GQ tracks this internal sharing with the per-core markers, while maintaining system level E or M state.

#### Crossbar and Schedulers

Before getting to GQ transactions that MISS in the LLC, we cover the crossbar and schedulers. When requests enter the GQ, they may end up being serviced by the LLC, the local DDR controller, or the requests may need to go out of socket on QPI to either the sibling processor socket (for access to its GQ LLC or DDR controller), or the I/O hub socket to reach PCIe or other system devices. The GQ has two important mechanisms to deal with routing



and prioritizing this request traffic and responses (such as read data or write-completions) required to satisfy the requests. The credited queue structures, which house all pending GQ requests, have request age- and priority-based markings. As mentioned earlier, the GQ selects requests for the LLC lookup pipeline based on these markings. If the requests need service in the socket-local, MC or on QPI, the GQ must obtain the credits to request those services. The GQ schedulers thus use age, priority, and next-level credit availability as inputs to decide which request to dispatch next. There are multiple schedulers operating in this manner, in parallel, on the GQ outbound paths to LLC, QPI, DDR, and the core U2C IDI paths. The GQ schedulers must also ensure forward progress, such that no request or request source is accidentally starved by the complex scheduler policies.

### Crossbar

The primary crossbar portion of GQ is the data queuing switch. When data moves from one location to another within the socket, or across QPI, or to the DDR memory, it does so as full cache lines, 64 Bytes at a time. The GQ data buffer essentially operates as a multi-ported, 64-deep cache-line buffer. The ports are each 32-Bytes wide and may all operate simultaneously, moving cache lines core-to-core, LLC to or from QPI, cores or DDR memory, or DDR to and from QPI, LLC, or cores. The GQ's request age- and priority-based schedulers keep track of incoming cache lines and are ready to move outgoing cache lines from this buffer. There are also data queue bypasses for minimal latency movement of data to cores from LLC or local DDR channels.

### QPI Protocol and Link Layers

When requests to the GQ are not serviceable locally in the LLC or memory controller, they must be scheduled to QPI for service in the other system sockets. The QuickPath Interconnect is the subject of another article in this issue of the *Intel Technology Journal*; a high-level description, therefore, is sufficient for our purposes [2]. The GQ implements the protocol layer of QPI and also acts as the protocol-layer gateway to the QPI link-layer. The key tasks of the QPI protocol layer from a GQ perspective are to track the protocol layer credits required to generate a QPI request or cache snoop to another socket, and to generate the QPI snoop or completion response to any incoming QPI snoop or Interrupt request created by other sockets within the platform.

In the Xeon 5500/5600 systems, the QPI protocol-layer credits for access to a destination socket's resources are managed by the GQ. Specifically, for cacheable accesses to DRAM in a dual-socket Xeon-5500/5600 system, there are conceptually two HOME agents as defined by the QPI specification. (A HOME agent is the logical controller responsible for responding to QPI requests for use of cache-coherent DDR memory.) There are two HOME agents in Xeon-5500/5600 systems, one in each processor Uncore. The address space for all of DRAM is interleaved between these two HOME agents such that for any given 64-byte cache line address, exactly one HOME agent is responsible for responding to a QPI request for reading or writing

*“The GQ schedulers must also ensure forward progress, such that no request or request source is accidentally starved by the complex scheduler policies.”*

*“When data moves from one location to another within the socket, or across QPI, or to the DDR memory, it does so as full cache lines, 64 Bytes at a time.”*

*“In the Xeon 5500/5600 systems, the QPI protocol-layer credits for access to a destination socket's resources are managed by the GQ.”*

*“The DDR memory bandwidth capability local to a socket is higher than the socket-to-socket bandwidth capabilities across QPI links.”*

that address. The HOME agent controller is physically located in the Uncore of each processor socket near the MC. Each GQ is provided protocol-layer credits that give it rights to send a certain number of requests to the HOME DDR controller in the local socket Uncore, and in the remote processor socket Uncore. There are separate credit-pool depths for local HOME and for remote HOME. The depths are either 16 or 24 QPI protocol-layer credits for requests to a remote HOME, which in turn corresponds to an ability for each GQ to generate up to 16 or 24 outstanding requests to the sibling processor socket. A larger number of credits is provided for protocol-layer access to the local HOME within each Uncore. More credits are needed for higher bandwidths, and the DDR memory bandwidth capability local to a socket is higher than the socket-to-socket bandwidth capabilities across QPI links.

Entry to the QPI link layer is also a credited operation, managed by GQ. QPI multiplexes could be described as separate, unique, virtual connections onto a single physical link. This multiplexing of physical to “virtual” channels is called the message link layer of QPI. Each of the six message channels (HOM, SNP, DRS, NDR, NCS, and NCB) is described in [2], and two QPI virtual networks VN0 and VNA (whose function is also described in [2]) have their own guarantee of forward progress from source to destination, despite being multiplexed together onto the outbound, high-speed QPI link. GQ schedulers manage the credits to gain access to each of the QPI message channels and virtual networks for each of the two links.

QPI is thus a significant fraction of the GQ’s scheduler design, since both QPI protocol and link-layer credits must be included in the age- and priority-based schedulers and the forward progress aspects of the request scheduler implementation.

## Shared Cache Component

In this section we describe the LLC: a large cache shared by all on-die cores. The LLC caches instructions, data, and page-table entries.

### Last-level Cache Array

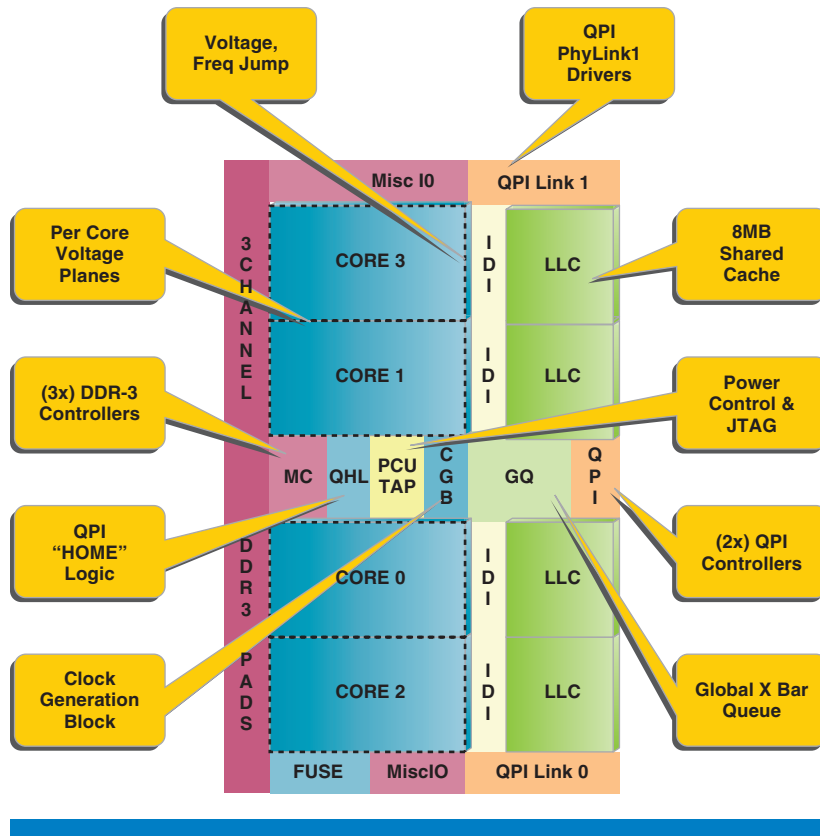
The LLC is modular in the die floorplan as a 2-MB array block that is physically, but not logically, associated with each core. All cores share the entire LLC. Figure 3 shows the physical floorplan of components. For the 4-core Xeon-5500, there are 8 MB of shared LLC. For the 6-core Xeon 5600, there are 12 MB of shared LLC available. Both generations’ LLC implementations are 16-way set associative. The cache is fully shared, which means any single core may fill or access the full LLC capacity (as in when the PCU has powered off all but one core). Code, data, and page-table entries may be cached in the LLC.

The LLC consists of a tag array, data array, and a state plus LRU (least recently used) array, which also includes the marker bits that the GQ uses for keeping track of which cores might be using any LLC cache-line address.

*“All cores share the entire LLC.”*

*“The cache is fully shared, which means any single core may fill or access the full LLC capacity.”*





**Figure 3:** Floorplan-centric view of Uncore components

### Cell Selection

A key piece of the modular architecture is the ability to separate the core power planes from those of the Uncore. This separation allows technology-dependent requirements of the cores to be isolated from the technology-dependent components of the Uncore, such as voltage-level operating constraints of the large cache memory cells. Since the program goals of the Nehalem-Westmere teams were to operate processor cores at the lowest possible voltage levels for power-performance reasons, it was advantageous to operate the Uncore power plane at a voltage that is optimal for the large LLC cache array and system interface functions. This design allowed an optimal small LLC cache cell with good leakage and reliability characteristics to be selected, one that didn't affect the minimum or maximum operating voltages and frequencies of the cores.

### Redundancy

The LLC uses a DECTED double-error correction, triple-error detection scheme for improved data reliability. In Intel Xeon products, DECTED provides a superior level of reliability compared with single error correction (SECTED) schemes. Providing this enhanced level of functional reliability for servers is a segment-specific feature that is valued by server system customers. It is also a reasonable feature for cost-sensitive client products, which get the

*“Placing the LLC in its own power domain also allows the Nehalem Turbo Boost Technology feature to raise core frequencies and voltage to optimal peak values.”*

*“When a core needs to wake up again, assuming the period of sleep has been short, the LLC will likely still have available a fair percentage of the execution context that was originally in the core caches.”*

*“The QPI link layer consists of six message channels on two virtual networks, for each of the two QPI links.”*

enhancement benefit in the form of a physically smaller cache array. This leads to lower product cost, yet is reliable enough to operate at very power-efficient voltage levels.

#### **LLC Implications for Core Power-performance**

Having an area- and power-optimized LLC cache and cores on the same power plane would imply a voltage floor that limits the ability to drop cores to their lowest possible voltage operating points, and hence be a source of significant leakage. Isolation of the LLC array design to the Uncore power plane allowed the core voltages to be moved to a lower minimum operating point during less computationally demanding time periods, which improved power efficiency.

Placing the LLC in its own power domain also allows the Nehalem Turbo Boost Technology feature to raise core frequencies and voltage to optimal peak values during computationally-demanding time periods without taking Uncore and LLC to those same levels. This leaves more thermal and current headroom for cores to invoke Turbo Boost Technology. Additionally, when one or more cores go into a deep power-savings mode (in which the core's voltage is removed) there is no state remaining in those cores. The architectural state required to reliably recover the core is saved in special regions of the LLC before the core powers down, such that it may be quickly and securely recovered. The ability to turn cores on and off is a large power-performance efficiency feature in the Nehalem-Westmere family of products.

The LLC also contributes to the performance of this power state function through the inclusive caching policy implemented by the GQ. When a core is to be powered off, any modified cache-line state in the core caches must be written back to either the LLC or DDR memory for functional correctness reasons. Because the LLC tag array is maintained inclusive of all cache lines that might be in the cores, these M-state write-backs are guaranteed to sink into the LLC at high bandwidth and low latency, rather than require a long latency and lower bandwidth trip to system DDR memory. When a core needs to wake up again, assuming the period of sleep has been short, the LLC will likely still have available a fair percentage of the execution context that was originally in the core caches. This LLC performance feature allows Nehalem-Westmere cores to rapidly go in and out of powered-off states, which improves their power-performance efficiency, because low-power states can be used more aggressively due to the short latency of entry and exit.

#### **QPI Link and PHY-Layer Unit Component**

While the GQ implements the QPI protocol layer, a separate portion of the Uncore, called the QPI unit, implements the link and physical layers for each of two QPI links. The QPI unit communicates primarily with the GQ, but also with the local HOME agent controller that responds to DDR memory requests. As described in the preceding GQ section, the QPI link layer consists of six message channels on two virtual networks, for each of the two QPI links.

Each QPI link within the QPI unit is treated by the GQ as a standalone device, which has its own set of link credits for each message class. Outbound messages on QPI links are delivered to the PHY layer in units called FLITs (QPI FLow control level unITs) and are 80 bits wide. The GQ moves data to and from the QPI units in 64-byte, cache-line granules, at 16 bytes per Uncore clock. Therefore, the FLIT-level data breakout is accomplished within the link layer QPI unit, where it is moved to the QPI physical layer. The GQ sends commands to the QPI unit, such as QPI requests or snoops, on a separate path from the data, but they are handled similarly to those in the QPI unit. The command is sent to the PHY layer as one FLIT, and data are sent to PHY as eight FLITs. The QPI unit PHY layer hardware then breaks up FLITs into 20-bit quantum known as PHITs (PHysical layer unITs) and reliably transmits them to the destination socket. PHITs are the physical width of the QPI link, in each direction.

The QPI unit sends incoming QPI messages from each of the link-layer virtual network message channels to either the GQ or the HOME agent. The HOME agent is a QPI term, but within Uncore the HOME agent is located at the front-end request queue to the MC unit. An example delivery from the link layer into Uncore would be an incoming request to use the MC in this socket. The QPI unit observes the message channel request for DDR and breaks it into two message pieces. The primary message is sent to the HOME agent MC front end, and this request starts functions within the MC to service the request. The message is also sent to the GQ as a snoop request for that same address, such that the GQ can interrogate the socket-internal cache hierarchy for a cached copy. The GQ responds to the MC HOME agent with its snoop results, and the MC HOME agent is responsible for eventually supplying a response back to the QPI fabric for that memory request.

Additional higher-level information on QPI operation is provided in [2]. It may be helpful to read that article to provide some context before reading this article on how Uncore implements the Xeon 5500/5600 subset of QPI's requirements.

## Memory Control Unit Component

The MC in the Xeon 5500/5600 was used for products that span the mobile to server market. Therefore, the controller supports a superset of server and client features. Top-level features include Small Outline-DIMMs (SODimm), Unbuffered DIMMs (UDIMM), and Registered DIMMS (RDIMM), with x16, x8, and x4 data width, Isochrony, device-failure correcting ECC, DRAM throttling, and independent or lockstep channel operation.

Three DDR channels provide server memory capacity and bandwidth. Each channel allows for one to three DIMMS per channel, depending on maximum channel frequency. The DDR channels typically operate as independent controllers, with requests originating in either the socket-local GQ or the QPI links.

*“The GQ sends commands to the QPI unit, such as QPI requests or snoops, on a separate path from the data.”*

*“PHITs are the physical width of the QPI link, in each direction.”*

*“Three DDR channels provide server memory capacity and bandwidth.*

*Each channel allows for one to three DIMMS per channel, depending on maximum channel frequency.”*

Focusing on optimizing power efficiency in this generation of chipsets led us to focus on lower-latency, open-page policies and independent channels as opposed to the lockstep, closed-page policies used in previous server chipsets.

Counters in the MC track the energy required for each DRAM command for the specific DIMMs populated. Commands are throttled when the energy counters exceed the configurable cooling capacity of the system.

Power optimizations include the DRAM CKE (clock enable) protocol, modulated according to package power state, and Self Refresh power modes with clocks powered off.

A front-end queue structure to the MC, called the QPI HOME Agent, enforces coherency ordering, such that the MCs operate in parallel on the stream of requests coming from the local Home Agent without needing to deal with memory ordering issues. The MC is therefore free to optimize access order for overall latency and best utilization of the memory devices.

## Power Control Unit

The Uncore contains the fully programmable PCU that coordinates Nehalem's power/performance advances. It provides flexibility and dynamic adjustment capability to make use of new power-saving technologies Intel developed for the 45-nm and 32-nm process families. The Uncore is a convenient location for the PCU, because the Uncore has access to all system interfaces, clocking, and core connections needed to sense and adjust for optimal operation. The programmable PCU architecture means product-specific Uncores can tailor PCU operation to platform- or segment-specific needs and enable algorithms to be improved over time.

The PCU within the Uncore also works closely with clock-generation units, fuses, and configuration register infrastructure to coordinate the reset and initialization sequencing within the socket. The PCU works in concert with the operating system and CPU core firmware to coordinate all movements in and out of power optimization states.

*“The PCU works in concert with the operating system and CPU core firmware to coordinate all movements in and out of power optimization states.”*

## Clocking

All PLLs, and the master clock grid that distributes primary clocks to cores and to Uncore elements, are contained in the Uncore. One architectural reason for making the chip-level clocking part of the Uncore is to enable modular, converged-core support, such that die-level clocking may be arranged to suit various levels of integration in future products. One lower-level architectural clocking item worth noting is the leverage of core-Uncore skew tolerance in the IDI interface to tolerate first-droop, power-supply sag. The circuit technology

article in this issue [3] explains this Adaptive Frequency System (AFS) mechanism in greater detail. From an architectural point of view, the core clock PLL may be allowed to wander a fixed amount, because the IDI interfaces can tolerate this level of clock skew between cores and the Uncore. The AFS mechanism allows improved efficiency because the cores can in effect operate at higher frequency for any given supply voltage.

## I/O

The Uncore chip I/O buffers include all QPI, DDR, initialization, and test port I/O. From an architectural perspective, the I/O is reusable in the modular paradigm between products.

## Chip Debug Port Mechanisms

The Uncore has the JTAG-compliant test access port (TAP) for access to internal socket debug hardware. From an architectural perspective, the Nehalem-Westmere processors are partitioned such that the modular, converged-core has a well-defined debug interface as part of IDI. The variable integration levels, and reset initialization functions owned by the Uncore, inclined the team to place the global debug mechanisms in the Uncore design domain.

## Performance

To achieve high performance in the Nehalem and Westmere Uncore designs, we focused especially on achieving low latency under high bandwidth conditions in the caches, MCs, and in the logical connections between sockets.

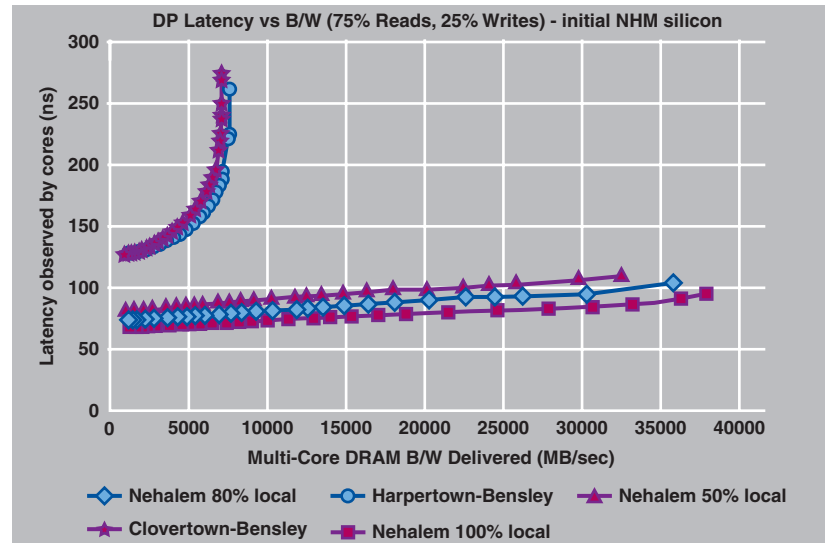
For the server parts, the Nehalem family idle Load-to-Use latency statistics for LLC and DRAM access are cut by more than 25% compared with prior-generation, FSB-based Xeon systems.\* This is a great achievement in and of itself and a significant contributor to the performance delivered by Nehalem systems. But more impressive and important are the relatively flat and greatly improved latency-under-loaded-bandwidth characteristics. They contribute most noticeably to the overall Nehalem platform strength. Figure 4 shows the prior-generation, FSB-based Bensley platform and Nehalem Xeon-5500 platform processor performance under various levels of bandwidth load. The improvements are a direct result of the QPI and Uncore credited flow control mechanisms and complex schedulers managing the traffic flow and timely forward progress.

*“The core clock PLL may be allowed to wander a fixed amount, because the IDI interfaces can tolerate this level of clock skew between cores and the Uncore.”*

*“The Nehalem-Westmere processors are partitioned such that the modular, converged-core has a well-defined debug interface as part of IDI.”*

*“More impressive and important are the relatively flat and greatly improved latency-under-loaded-bandwidth characteristics.”*

\*Based on Intel internal measurements



**Figure 4:** Xeon-5500 loaded-latency versus prior-generation Intel Xeon-Bensley

#### Maintaining Low Memory Latency

An issue observed in earlier, retry-based interconnect protocols is that under high bandwidth conditions significant queuing delays may occur in the interconnect and memory system. These delays are caused by traffic congestion and a lack of upstream resources such as the socket-to-socket interconnect bus, socket internal interconnects, or delays in the MCs. The core request sources could not “see” destination resources becoming overwhelmed until the requester would receive a “retry” response and realize the upstream resources were too congested. The retries and layers of arbiter-based flow control mechanisms involved would cause waves of jittery stop-start, flow-control propagation delays back through the congested interconnect traffic. The delays would amplify as more retries occurred to dramatically affect the overall latency and throughput of the loaded system. The memory service delays in turn would cause core pipelines to not be fed data and instructions, which then resulted in performance issues and poor multi-core, power-performance scaling. The large delay effects were often not seen in uncongested conditions, which is why some computer systems with excellent idle load-to-use latency behaved poorly under heavy bandwidth demand conditions.

The designers of the Nehalem and Westmere Uncores implement a new approach to feeding the cores that significantly improves the memory data movement latencies under these heavily loaded conditions. The new approach involves uniformly credited flow-control mechanisms across the platform, coordinated by the intelligent Uncore schedulers mentioned earlier for IDI, LLC, QPI and DRAM controller interfaces. The uniform traffic crediting

*“The large delay effects were often not seen in uncongested conditions, which is why some computer systems with excellent idle load-to-use latency behaved poorly under heavy bandwidth demand conditions.”*

mechanisms create a fairly linear backpressure effect across all QPI links, MCs, and internal interfaces back to the core requesters and prefetchers. This avoids significant oversubscription of the interconnect at the interconnect entry points, which in turn means traffic systematically makes progress from source to destination under heavily loaded scenarios.

As a result, Nehalem technology can achieve very high burst and sustained memory bandwidth utilizations with a shallow loaded latency deterioration curve. This contrasts noticeably to “hockey stick” queuing-delay response curves incurred in prior-generation products, as graphically shown in Figure 4.

In Figure 4, the basic low-latency design of the Nehalem Uncore is evident in the differences in latency versus those of previous platforms, which are shown at the low-bandwidth operating points to the left of the graph. More importantly for Nehalem-Westmere power-performance, the loaded latency, as cores demand strong memory performance, remains low, as seen on the right side of the graph, which means less core pipeline stalls and therefore higher core throughput. Figure 4 is the visual short summary of Uncore and QPI’s contribution to Nehalem-Westmere multi-core performance versus the architecture of prior systems. Core pipelines drain less often, because the Uncore keeps the pipes fed at reasonable latencies under demanding, multi-core, throughput conditions.

## Conclusions

The Nehalem-Westmere Uncore made substantial improvements in modular design, power efficiency, and performance. These improvements enabled carefully-tailored sibling products in less time, which were better optimized to their platforms while adhering to the stable converged-core software model.

*“Nehalem technology can achieve very high burst and sustained memory bandwidth utilizations with a shallow loaded latency deterioration curve. This contrasts noticeably to “hockey stick” queuing-delay response curves incurred in prior-generation products.”*

*“Core pipelines drain less often, because the Uncore keeps the pipes fed at reasonable latencies under demanding, multi-core, throughput conditions.”*



## References

- [1] Steve Gunther, Anant Deval, and Ted Burton. “Energy-Efficient Computing: Power-Management System on the Intel® Nehalem Family of Processors,” *Intel Technology Journal*, Volume 15, Issue 1, 2011.
- [2] Gurbir Singh, David L Hill, Ken Creta, Rob Blankenship, Nilesh Bhagat, Debendra Das Sharma, David Johnson, Robert A. Maddox, and Robert Safranek. “The Feeding of High Performance Processor Cores—QuickPath Interconnects and the New I/O Hubs,” *Intel Technology Journal*, Volume 15, Issue 1, 2011.
- [3] Kumar Anshumali, Terry Chappell, Wilfred Gomes, Jeff Miller, Nasser Kurd, and Rajesh Kumar. “Circuit and Process Innovations to Enable High-performance, and Power and Area Efficiency on the Nehalem and Westmere Family of Intel® Processors,” *Intel Technology Journal*, Volume 15, Issue 1, 2011.

## Authors’ Biographies

**Derek Bachand** is a Principal Engineer in the Intel Architecture Group working on microarchitecture and RTL development in the Intel® Xeon-EP and client processor Uncore. Derek joined Intel in 1992 after receiving his BS degree in Computer Engineering from Virginia Tech. He has worked on multiple generations of IA processors primarily in the bus interface units starting with P6 (Pentium® Pro) and he has led design implementation activities for the Xeon 5500/5600 cache, Global Queue, QPI protocol layer, and internal interconnects. His email is derek.t.bachand at intel.com.

**Selim Bilgin** is a Director of Engineering in the Intel Architecture Group. Selim led the design and implementation of Intel’s first QPI implementation for the Xeon 5500/5600 products. Selim joined Intel in 1997 after receiving his MS degree in Electrical and Computer Engineering from the University of Wisconsin-Madison. He worked on multiple generations of IA processors starting with the Pentium® 4 processor family. His email is selim.bilgin at intel.com.

**Robert Greiner** is an Architect Principal Engineer in the Intel Architecture Group. Robert has 40 years industry experience and developed the quad-pumped system bus, first used on Pentium® 4 processors, and the QPI interconnect and PCU infrastructure for the Nehalem family of products. He holds a BS degree in Mathematics from Michigan State University. His email is robert.greiner at intel.com.

**Per Hammarlund** is an Architect Senior Principal Engineer in the Intel Architecture Group working on micro-architecture development, hyper-threading, power efficiency, and performance modeling. Per started at Intel in 1997 working on the Pentium® 4 processors. He received his PhD degree in Computer Science from the Royal Institute of Technology (KTH) in 1996. His email is per.hammarlund at intel.com.



**David L Hill** is an Architect Senior Principal Engineer in the Intel Architecture Group. Dave joined Intel in 1993, and he has 30 years industry experience primarily in high-performance caches, interconnect controllers, and coherent memory systems. He was the Uncore lead architect of concepts and implementation for Nehalem and Westmere Xeon-5500/5600 and client products, and the Westmere product lead architect. Dave received his BS degree in Electrical Engineering from the University of Minnesota. His email is david.l.hill at intel.com.

**Thomas Huff** is an Architect Senior Principal Engineer in the Intel Architecture Group. Tom developed key portions of the Uncore and platform repartition concepts for Nehalem and Westmere products. He holds MS and PhD degrees in Electrical Engineering from the University of Michigan. His email is tom.huff at intel.com.

**Steve Kulick** is an Architect Principal Engineer in the Intel Architecture Group. Steve was the lead architect for the 8000 series server chipsets and was the lead memory controller architect for the Xeon-5500/5600 products. He holds a BS degree in Electrical Engineering from Drexel University. His email is steve.kulick at intel.com.

**Robert Safranek** is an Architect Principal Engineer in the Intel Architecture Group. Robert was a primary developer and author of the QPI specification, and he also led development of the processor QPI implementation architecture for Intel Xeon 5500/5600 products. Robert holds a BS degree in Electronic Engineering from the University of Nebraska and an MS degree in Computer Engineering from Portland State University. His email is robert.j.safranek at intel.com.

## Copyright

Copyright © 2011 Intel Corporation. All rights reserved.

Intel, the Intel logo, and Intel Atom are trademarks of Intel Corporation in the U.S. and other countries.

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

Requires an Intel® HT Technology enabled system, check with your PC manufacturer. Performance will vary depending on the specific hardware and software used. Not available on Intel® Core™ i5-750. For more information including details on which processors support HT Technology, visit <http://www.intel.com/info/hyperthreading>

Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM). Functionality, performance or other benefits will vary depending on hardware and software configurations. Software applications may not be compatible with all operating systems. Consult your PC manufacturer. For more information, visit <http://www.intel.com/go/virtualization>

Requires a system with Intel® Turbo Boost Technology capability. Consult your PC manufacturer. Performance varies depending on hardware, software and system configuration. For more information, visit <http://www.intel.com/technology/turboboost>

# ENERGY-EFFICIENT COMPUTING: POWER MANAGEMENT SYSTEM ON THE NEHALEM FAMILY OF PROCESSORS

## Contributors

**Steve Gunther**  
Intel Corporation

**Anant Deval**  
Intel Corporation

**Ted Burton**  
Intel Corporation

**Rajesh Kumar**  
Intel Corporation

## Index Words

Power Management  
Energy Proportional Computing  
Power Control Unit  
Dynamic Voltage Positioning

*“The art of power management is in the development of technologies that allow a product to meet all power-consumption constraints while maximizing performance that is valued by the end user.”*

## Abstract

Maximizing energy efficiency was a key goal on the design of the The *Intel® microarchitecture code name Nehalem*, which was conceived as a modular architecture with multiple cores that would scale from very small mobile platforms to very high-performance, server configurations. Building a scalable product while working within the voltage and leakage scaling physics required several innovative technologies including power gate transistors, dramatically improved Intel Turbo Boost Technology, and Dynamic Voltage Positioning. Power-plane partitioning, clocking strategies and cache, and memory and I/O power-management algorithms were all tuned to maximize energy efficiency while providing breakthrough product performance across a wide range of workloads. The resulting algorithmic complexity drove the inclusion of an embedded micro-controller (the Power Control Unit, or PCU) to implement the power-management decisions and control. In this article we describe the key power-management features implemented on the 45nm and 32nm Nehalem family of products.

## Introduction

Power consumption is no longer a concern for just mobile products; it is a concern for all processor market segments. In a mobile platform, the key concerns are to maximize the amount of work that can be accomplished with the energy stored in the battery and to manage higher thermal densities at the chassis level. In a desktop platform, typical concerns are the cost of building the platform (heat sink, air flow, and power-delivery costs) and the ability to meet consumer preferences for such things as the “Energy Star” label. Energy consumption is a component of the total cost of ownership, a significant consideration for a server platform.

Because every platform is power constrained in some fashion, power consumption and performance are essentially equivalent: getting higher performance frequently means reducing power consumption. Usually, an increase in performance is accompanied by an increase in power. The art of power management is in the development of technologies that allow a product to meet all power-consumption constraints while maximizing performance that is valued by the end user.

As Intel's flagship microprocessor, the goal of the engineers working on the family of processors, codename Nehalem, was premium performance across all workloads in all product segments. Achieving this goal required innovation in all areas ranging from architecture to circuits. The key power-management technologies introduced on the Nehalem processor and described in this article are these:

- *Dynamic voltage positioning.* This provides the necessary minimum voltage for correct operation in a given environment.
- *Development of a novel “power gate” transistor.* Power gates provide the capability to transparently remove unused power from a processor core while allowing other cores to continue executing instructions.
- *Voltage and clock partitioning of the processor.* This separates the computation engines (cores) from the support logic, including the large shared cache and the interfaces to the rest of the platform (the “Uncore”) thus optimizing power and performance.
- *Platform power-reduction mechanisms.* These reduction mechanisms minimize power consumption in memory, the links between the processor and other components in the platform, and the voltage regulator (VR) supplying power to the processor.
- *Intel Turbo Boost Technology.* This provides a performance boost by increasing operating frequency while staying within the processor power-delivery or thermal design constraints.
- *Introduction of an embedded microcontroller, the PCU.* The PCU implements the above-mentioned power-management technologies.

## Energy-Proportional Computing

The objective of energy-efficient computing is to consume the minimum energy needed to hit the performance targets for the currently executing workload. For a computation-bound workload; i.e., one in which performance scales well with the clock frequency of the cores, the goal is to minimize power consumption in logic blocks not directly involved with the computation, such as the memory and platform interfaces. For an application that does not utilize all computation engines on the processor, eliminating the power consumed by the idle computation engines is key. Because most applications do not fall into a single class, dynamic allocation of power to various functional blocks is critical for providing energy-efficient performance.

Nehalem technology lowers core active power consumption by voltage-frequency scaling that is based on workload demand, and it minimizes the voltage necessary to operate at a given frequency. Core idle power can be reduced in a variety of ways ranging from halting instruction execution to turning off clocks

*“The objective of energy-efficient computing is to consume the minimum energy needed to hit the performance targets for the currently executing workload.”*

*“On a multi-core die such as Nehalem, the capability to completely power-off idle cores with low entry and exit latency is crucial for increasing the power-performance efficiency.”*

*“The processor can also transition from a high to a low activity state in a few nanoseconds.”*

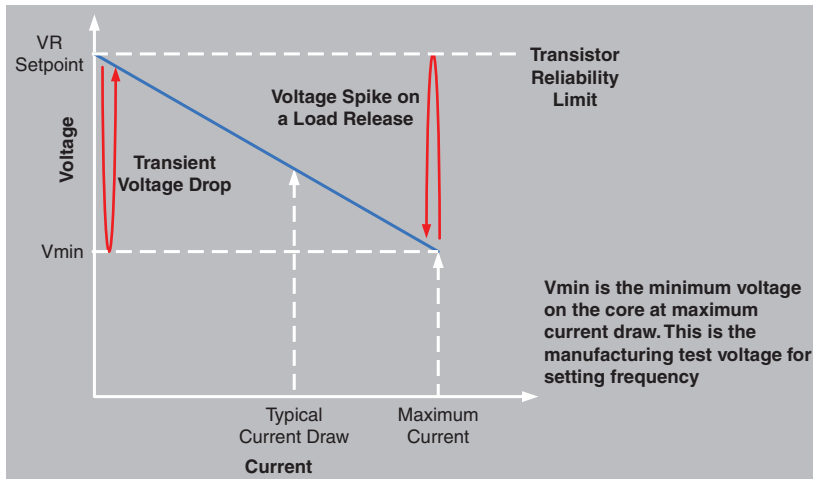
and lowering voltage. On a multi-core die such as Nehalem, the capability to completely power-off idle cores with low entry and exit latency is crucial for increasing the power-performance efficiency of the processor.

### **Dynamic Voltage Positioning**

The loadline is used in power-delivery systems to allow the voltage supplied to a processor to drop as the processor draws more current. There are three reasons to use a loadline: 1) to ensure that the processor continues execution in the presence of a transient voltage drop brought about by an increase in activity in the processor; 2) to ensure that the processor stays within the transistor reliability specifications; and 3) to reduce power consumption.

The dynamic current of the processor at a specific voltage and frequency is also dependent on the level of activity in the processor. The activity can change from a low to high level in a matter of a few nanoseconds. The impedance of a power-delivery system prevents the sudden delivery of charge; therefore, charge is drawn from the capacitance on the die, thereby causing a transient voltage drop. The processor continues to execute instructions through this event. With the Nehalem technology generation, we introduce a new technology (the Adaptive Frequency System) that allows the clocks to be modulated by the transient voltage drop. This new technology increases the frequency of the part at the same voltage level, thereby increasing efficiency.

The Adaptive Frequency System (AFS) is explained in detail in the circuit technology article also appearing in this issue of the *Intel Technology Journal* [1]. The processor can also transition from a high to a low activity state in a few nanoseconds. This event is called a load release. However, the power-delivery system continues to deliver higher charge after such a transition, which leads to a voltage spike. Transistor physics dictate the maximum voltage on the transistor at a given temperature in order to meet the reliability specification. In the event of a voltage spike, the voltage on the transistors of the processor should not exceed the maximum transistor reliability voltage. Figure 1 shows the power-delivery loadline for a core with the transient voltage drop and voltage spike events. At maximum current, the voltage on the part is  $V_{min}$ . The power of the part is computed at  $V_{min}$ . Typically, applications consume less than maximum current, providing an opportunity for power optimization.  $V_{min}$  is also the voltage at which manufacturing flows test the frequency of the part. The loadline is designed so that the transient voltage drop does not let the voltage go below  $V_{min}$  and does not let the voltage spike exceed the maximum reliability voltage.



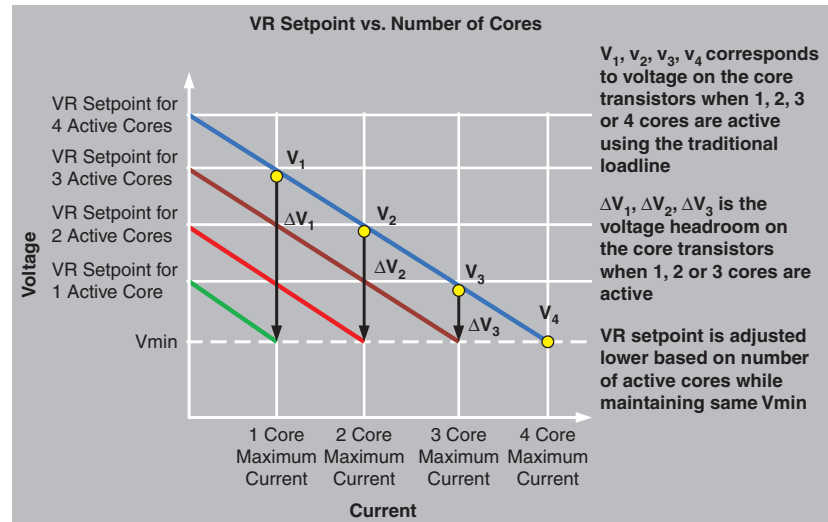
**Figure 1:** Power delivery loadline of a core

Historically, the operating voltage targets were based exclusively on the target operating frequency, via a frequency-to-voltage conversion table contained in the BIOS. Because these targets need to be sufficient to guarantee correct operation under worst-case operating conditions (such as maximum number of cores active), a voltage (power) guard band is maintained under more normal operating conditions. Nehalem's Dynamic Voltage Positioning system minimizes the voltage guard band by adjusting the VR setpoint as a function of the number of active cores, leakage, temperature, and platform loadline. When all cores are fully active at high temperatures, the maximum current is drawn. The voltage at the processor is lower than the output of the VR due to the impedance in the power-delivery system. Under maximum current draw conditions, the voltage at the processor is the lowest voltage in the system. In previous-generation processors, the VR setpoint is not adjusted based on the number of active cores or on the temperature, so the voltage seen at the transistor levels is higher than what is needed to support the frequency requirement.

In the Nehalem product generation, we lowered the VR setpoint when fewer cores are active, so the core transistors are at the minimum voltage necessary for the specified frequency of operation.

*“Nehalem's Dynamic Voltage Positioning system minimizes the voltage guard band by adjusting the VR setpoint as a function of the number of active cores, leakage, temperature, and platform loadline.”*

*“Under maximum current draw conditions, the voltage at the processor is the lowest voltage in the system.”*



**Figure 2:** Voltage Regulator setpoint as a function of number of cores active at a given frequency and temperature

Figure 2 shows the traditional loadline and the loadline with dynamic voltage-positioning enabled. With the traditional loadline, as the number of active cores decreases, the voltage on a core increases. This is represented by  $V_1, V_2, V_3, V_4$  corresponding to 1, 2, 3, or 4 active cores. When fewer cores are active, the capability to generate a large transient voltage spike and voltage drop is also reduced. Since the minimum voltage ( $V_{min}$ ) to run the part at a given frequency is fixed irrespective of the number of active cores, the VR setpoint can be lowered.  $\Delta V_1, \Delta V_2, \Delta V_3, \Delta V_4$  represent the amount by which the VR setpoint can be lowered without violations, for 1, 2, 3, or 4 active cores.

*“Processor temperature is monitored in the millisecond timescale.”*

*“Further power reduction in these idle processor cores means eliminating the non-switching power consumption—the power that leaks from the supply to ground when the circuit is idle.”*

The setpoint of the VR is also dynamically adjusted as the temperature of the processor changes. Processor temperature is monitored in the millisecond timescale, a slow enough rate to allow VR setpoint adjustments without interrupting instruction execution in the cores. Leakage current is exponentially correlated to temperature and is also dependent on the process technology; moreover, it varies from part to part. In the Nehalem generation, we tune the VR setpoint for each processor to maximize power-performance efficiency.

### Core C6 and the Power Gate Transistor

Previous microprocessors have done an excellent job of reducing the switching power consumption of idle computation engines, primarily by turning off the clocks on logic blocks that have nothing to do, and turning off the clock generation circuits like the Phase Locked Loop (PLL) on the entire processor core. Further power reduction in these idle processor cores means eliminating

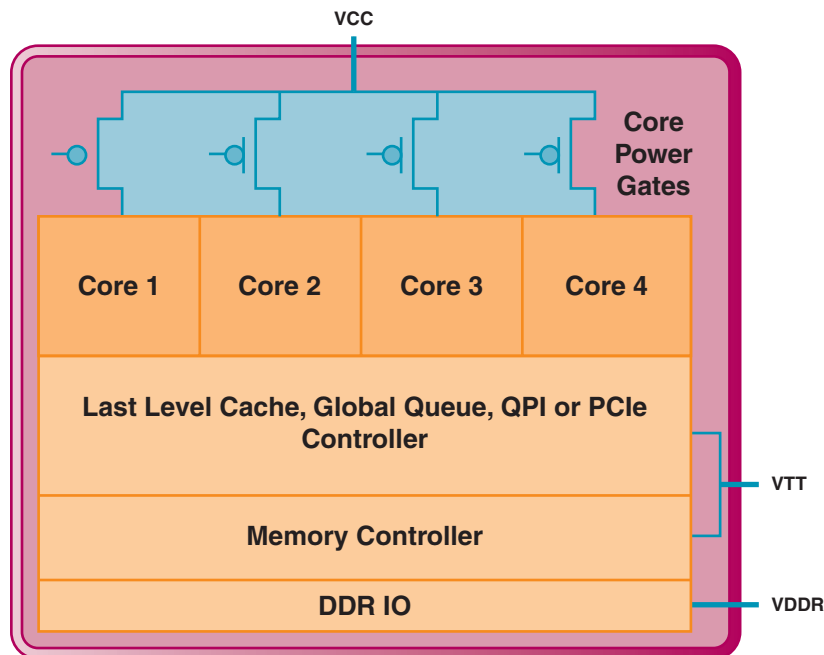
the non-switching power consumption—the power that leaks from the supply to ground when the circuit is idle. This leakage power consumption is a significant portion of the total consumption on a modern microprocessor, even with aggressive circuit and transistor technology optimizations.

By taking the voltage supply to 0 volts, leakage power consumption is eliminated entirely. While this is a simple concept, there are two inherent challenges.

The first challenge is the difficulty of removing voltage from the idle processor core. In a multi-core processor, all processor cores typically share a single VR on the motherboard. Adding VRs takes up significant space on the motherboard. This results in larger motherboards, which in turn puts pressure on the form factor and increases platform cost.

In the Nehalem family of processors we solve this challenge by integrating a power switch, or “power gate,” on the processor die between the VR on the motherboard and each processor core. Figure 3 shows the topology of the power gate transistors. When the associated processor core is executing, the power gate is “on,” providing a very low impedance path between the VR and the core. Minimizing the impedance is key to power reduction, as any voltage drop across the power gate consumes power and reduces performance.

*“In the Nehalem family of processors we solve this challenge by integrating a power switch, or “power gate,” on the processor die between the VR on the motherboard and each processor core.”*

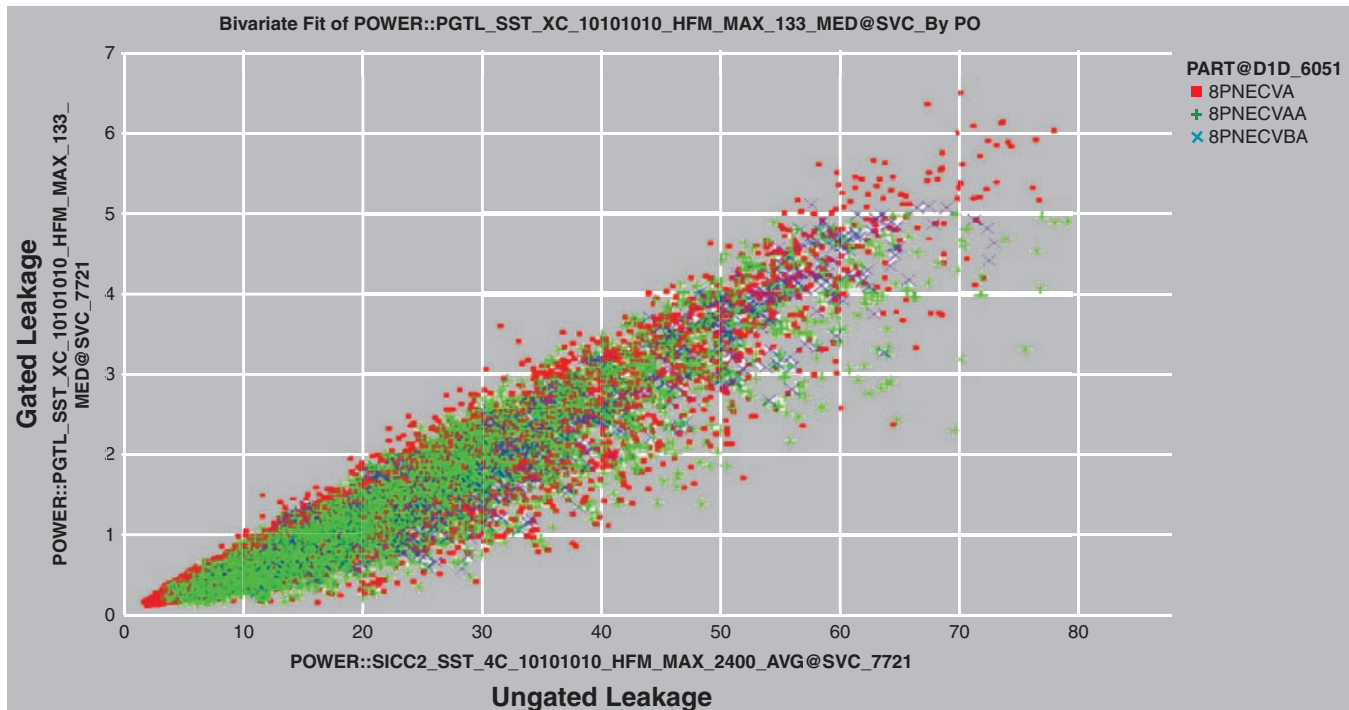


**Figure 3:** Power Plane Topology: each core has a power gate. “VCC” is the voltage regulator supply input for the cores; the rest of the processor is powered by a “VTT” VR and a “VDDR” VR



*“When the processor core is idle, the power gate is switched into an off state, which results in the transistors in the processor seeing the voltage supply drop to essentially 0. This virtually eliminates the leakage power consumed by the core.”*

When the processor core is idle, the power gate is switched into an off state, which results in the transistors in the processor seeing the voltage supply drop to essentially 0. This virtually eliminates the leakage power consumed by the core. Because there is almost no current through the power gate transistor in this state, there is also almost no energy loss in the power gate transistor. Figure 4 shows the distribution of leakage reduction achieved on a 4-core Nehalem part. The graph shows a reduction in leakage ranging from a factor of 10 to a factor of 30 when the cores are power-gated off.



**Figure 4:** Graph of the distribution of gated leakage (core supply is off) versus ungated leakage (core supply is on)

The second challenge to taking the voltage supply to zero volts is that before voltage can be removed from a processor core, its execution state must be saved to some form of non-volatile memory. The process of saving and restoring this state must be fast enough to prevent it from being noticeable by the operating system or application. This idle condition in which a processor core's execution state has been saved and voltage has been removed from the core is referred to as the “C6” state on Nehalem.

When the operating system has no work to schedule on a given processor core, it makes a request for that processor core to enter a lower-power idle state by executing the MWAIT instruction. When the C6 state is requested by the operating system, the core completes execution of all previous instructions in the code stream and stops fetching any new instructions. Because the voltage

on the core is going to 0, all caches on the processor core are flushed to the last level of shared cache. Next, all architectural states required to transparently resume execution are saved to an on-die SRAM that remains powered while the core is in the C6 state. Finally, notification is sent to the clock and voltage control logic to stop the PLL from supplying the clock signal to the core and to turn off the power gate. At this point, the power consumed by the processor core has dropped to approximately zero.

Exit from the C6 state is triggered by an interrupt directed to that specific core, or by an access to the memory address monitored as part of the MWAIT instruction executed to enter the C6 state. When either of these events occurs, the above process is reversed, and execution of the code stream continues with the instruction following the original MWAIT request. This process is transparent to the application, operating system, and other processor cores in the platform.

## Beyond the Processor Core—the Uncore

While most of the power consumed by the processor is consumed in the cores, the energy consumed in the Uncore, i.e., the remainder of the processor, cannot be ignored. The Uncore includes the shared last-level cache (LLC), the memory controller, the DDR interface, and the Quick Path Interconnect (QPI) or PCIe interconnect that connects the processor to the rest of the platform. Because the Uncore logic is shared by all processor cores, it needs to be alive when any processor core is executing instructions.

It should be noted that CPU performance is more sensitive to core frequency than Uncore frequency. This allows the Uncore to be optimized differently than the core from a power point of view. One such optimization is the decoupling of the voltage and frequency of the Uncore from the core, on Nehalem. Furthermore, individual sub-systems within the Uncore employ specific techniques to reduce active and idle power consumption.

## Voltage and Frequency Partitioning

Because the performance characteristics of the core and the Uncore are different, processor performance can be improved by separating the clock and voltage domains for these logic blocks.

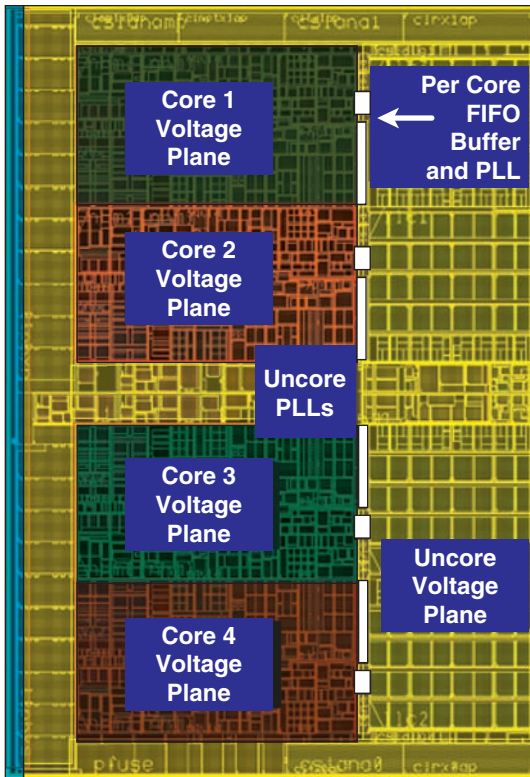
Separating the voltage domains allows scaling of core frequency and voltage, while leaving the Uncore voltage and frequency unchanged. Because the Uncore power is relatively constant, even though the core frequency is increased, more of the total power delivery budget can be spent in the core, thereby allowing higher core frequencies.

Separating the core and Uncore voltage also improves scaling down to lower voltages, because the LLC can remain at the higher voltage needed for SRAM cell stability, while the core scales to lower voltage and power. This option is especially important in the smallest form factors, where the core typically runs at a low frequency and voltage.

*“All architectural states required to transparently resume execution are saved to an on-die SRAM that remains powered while the core is in the C6 state.”*

*“The Uncore includes the shared last-level cache (LLC), the memory controller, the DDR interface, and the Quick Path Interconnect (QPI) or PCIe interconnect that connects the processor to the rest of the platform.”*

*“Because the performance characteristics of the core and the Uncore are different, processor performance can be improved by separating the clock and voltage domains for these logic blocks.”*



**Figure 5:** Nehalem floorplan showing various power and clock domains and the clock crossing FIFOs

*“The level of activity in the Uncore is typically much lower than the level of activity in the core.”*

Running the core and Uncore clocks at different frequencies created the need for a mechanism at the core-Uncore interface to synchronously transfer data between two frequency domains. This special FIFO buffer matches the bandwidth of the higher frequency domain to the lower frequency domain by allowing data transfer in certain time slots. Synchronous transfer of data through the FIFO buffer made it easier to debug and manufacture the part. Figure 5 shows the per-core power planes with per-core PLL on Nehalem. The Uncore has a separate set of PLLs to meet various frequency requirements.

### Ultra-Low Leakage Transistors

The level of activity in the Uncore is typically much lower than the level of activity in the core. This results in leakage power being a higher percentage of total power in the Uncore, relative to the core.

Separate transistor optimizations for the Uncore and core domains can be helpful. By providing a very low-leakage transistor type for use in the less performance-sensitive Uncore domain, leakage power in this domain can be reduced, even when faster transistors are still used in the core.

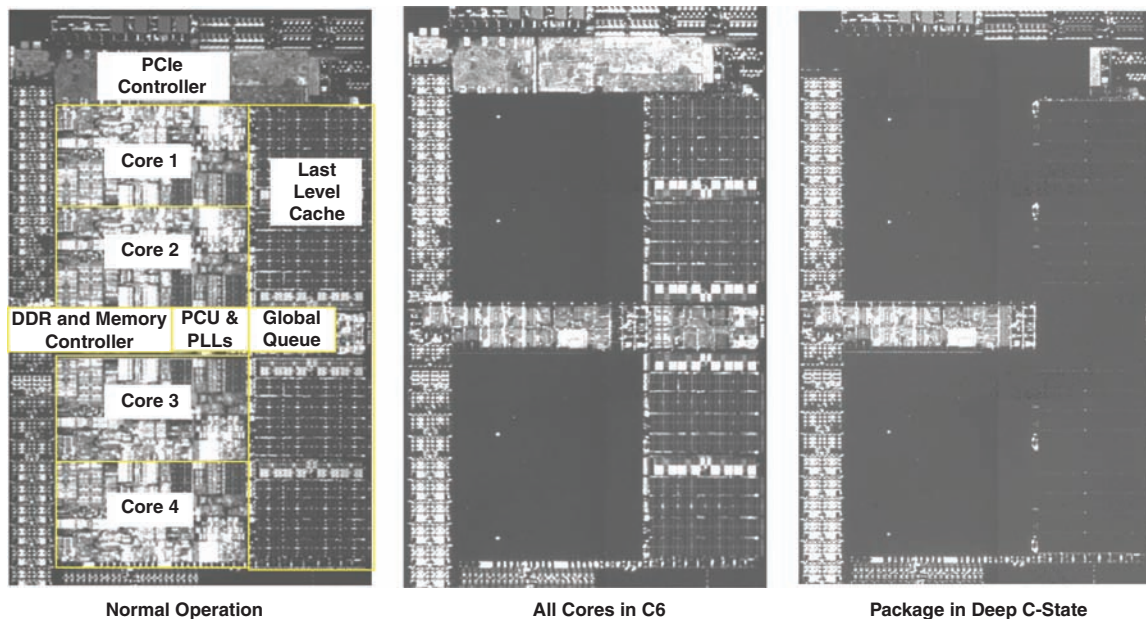
These low-leakage transistors were inserted in the design of the Uncore functional blocks, based on the timing margin available in various logic paths. By inserting the transistors in the Uncore functional blocks, Uncore leakage was reduced by more than a factor of two, relative to using only the nominal transistors.

### Sleep Transistors

In the case of the shared LLC, we inserted additional transistors (sleep transistors) between the supply side of the LLC memory cells, thereby creating a virtual supply. If there is no read or write access to the LLC, the entire cache is “sleeping.” In this mode, the sleep transistors lower the virtual supply to reduce the voltage across the memory cells, thereby reducing leakage. When an LLC access occurs, only the portion of the LLC that is being accessed is woken up. In other words, the virtual supply is raised to allow read or write access at normal operating frequency. The sleep transistors are controlled in such a way as to avoid adding latency to the LLC access.

Even after employing low-leakage transistors and LLC sleep transistors, there was an opportunity to further reduce leakage when the processor is idle.

We used sleep transistors to lower the voltage over the entire LLC, QPI PCIe digital logic blocks; thus, minimizing leakage power in the idle state. Figure 6 shows infrared emission microscope images for Lynnfield under various conditions. Lynnfield is a product based on Nehalem technology that integrated the PCIe controller. Under normal operating conditions, there is voltage on the entire die. When cores go into C6, the core power gates turn off the voltage to each core. When the package is in a deep idle state, the voltage on the parts of the Uncore—the entire cache and PCIe controller—is lowered.



**Figure 6:** Infrared emission images of the Lynnfield (a product based on Nehalem architecture) die during various power states

## Platform Power Optimizations

The microprocessor is only part of the story of energy-efficient computing; the power in the rest of the platform needs to be reduced too.

With the Nehalem processor, three areas of platform power consumption were addressed: reducing DRAM power, turning off the QPI or PCIe links, and improving the efficiency of the VR.

### DRAM Power Reduction

The power consumed by the memory subsystem can be significant, especially in a large server platform. However, DDR3 memory provides several power-reduction modes, which are utilized in a Nehalem platform.

During periods of operation where accesses to memory are infrequent, the local clock circuits on the DRAMs themselves are disabled, thereby reducing DRAM power consumption. The incremental latency required to read or write memory from this clock-disabled state is on the order of tens of nanoseconds, which is low enough not to noticeably impact performance, as long as the latency cost is not incurred frequently. To minimize the aggregate latency cost, logic is implemented to monitor accesses to DRAM, and the clock-disabled mode is entered only after a specified period in which no memory accesses have occurred.

For longer idle periods, when all cores have entered a low-power idle state, DRAM is placed in the self-refresh state. In this state, DRAM power

*“The microprocessor is only part of the story of energy-efficient computing; the power in the rest of the platform needs to be reduced too.”*

*“DDR3 memory provides several power-reduction modes, which are utilized in a Nehalem platform.”*



*“The deep idle state wake-up sequence was implemented such that the bulk of its latency is hidden behind other actions taken to wake the processor.”*

*“If one agent (processor or chipset) has nothing to transmit, it will negotiate with the receiver to enter a low-power state called L0s. In this state, the transmitter and receiver circuits are turned off.”*

*“The processor signals the VR when it can guarantee that it will remain at a low load level, and the VR uses this information to move to a more efficient state.”*

consumption is significantly reduced, and power consumption in the memory controller on the processor is reduced as well. While the latency to exit this state is substantial, the deep idle state wake-up sequence was implemented such that the bulk of its latency is hidden behind other actions taken to wake the processor (such as, transitioning the QPI or PCIe link to active state). The net benefit is a substantial reduction in the energy consumed by the memory subsystem, with no meaningful impact on system performance.

### **QPI and PCIe Link Power Management**

Both QPI and PCIe links are differential, point-to-point, high-speed links that connect a processor to another processor or connect the processor to a chipset. These links support power-management protocols for entering low-power modes. Normal link-operating mode is referred to as an L0 state. Logic monitors the inbound and outbound traffic in each link. If one agent (processor or chipset) has nothing to transmit, it will negotiate with the receiver to enter a low-power state called L0s. In this state, the transmitter and receiver circuits are turned off. In L0s, a transmit-receive pair can enter L0s independently of any other transmit-receive pair. When the processor wants to enter a deep idle state, it exchanges power-management messages with the other agents it is connected to through these links. In the deep idle state, the link is transitioned to an L1 state where the entire link and its associated PLL is turned off. To wake-up from this state, the agent needs to transmit switches on its transmitter, thereby developing a differential voltage on the link. Special circuits on the receiver detect the differential voltage and initiate the exit from the low-power state.

### **Improving the Efficiency of the Voltage Regulator**

It is worth noting, that the inefficiency in the power delivery system of the processor can meaningfully affect total platform energy efficiency. In particular, high-output current VRs can be very inefficient at lower load levels. To address this loss of efficiency, the processor signals the VR when it can guarantee that it will remain at a low load level, and the VR uses this information to move to a more efficient state. The Nehalem family of processors uses the Power Status Indicator to reduce the number of active phases in a regulator at low current draw. The processor can also request the regulator to go into “diode emulation” mode where the regulator will not pump charge into the power-delivery network unless the voltage falls below a certain threshold. The latency required for the VR to exit this higher-efficiency state is hidden in the sequence of operations taken when the processor changes its power state.

## **Performance When You Want It: Intel® Turbo Boost Technology**

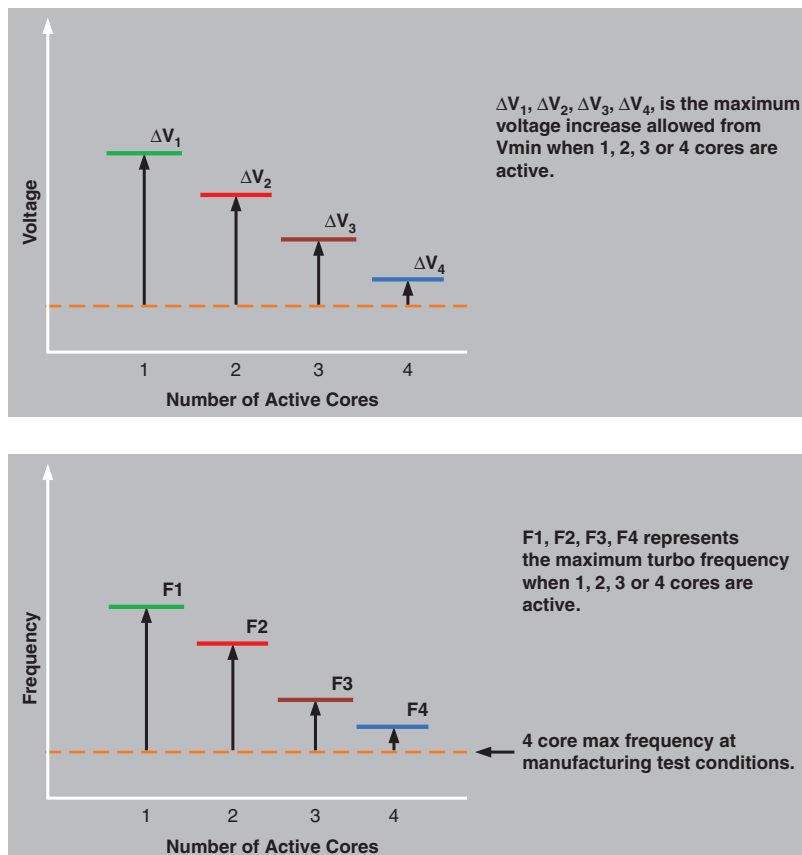
As mentioned earlier, the microprocessor is power limited in some fashion in all market segments. In addition to the limits on battery life, a processor may be limited at any point in time by thermal constraints on the processor silicon or in the platform, due to form factor. Limitations may also be imposed by the power-delivery system, namely the current capability and efficiency of the

voltage supplies. However, the most limiting factor at a point in time is often a function of the workload currently being executed.

The goal of the Nehalem designers was to maximize performance within these constraints on all workloads, whether that workload is exercising one core or all processor cores. Platform power-delivery and thermal solutions are built with the capability to run all cores at the highest frequency. This means that if the applications are not utilizing the processor fully, then there is power and thermal headroom that can be used to boost the performance of the processor. For multi-threaded applications, even if all cores are active, the temperature of the part may be lower than the maximum reliability temperature, because the application has lower core activity. Lower temperature translates to lower leakage power. Intel Turbo Boost technology translates this difference in power consumption from the platform limits to performance, by increasing the voltage and frequency of the active cores. Figure 7 shows the increase in voltage and frequency as a function of the number of active cores during turbo mode. When four cores are active, the voltage can be raised based on temperature; the voltage increase results in a corresponding frequency increase. When one core is active, there is more voltage headroom since three cores are off. The higher voltage headroom allows a single core with Turbo Boost technology to achieve higher frequencies.

*“Platform power-delivery and thermal solutions are built with the capability to run all cores at the highest frequency. This means that if the applications are not utilizing the processor fully, then there is power and thermal headroom that can be used to boost the performance of the processor.”*

Voltage and Frequency as a Function of Number of Active Cores in Turbo mode.



**Figure 7:** Voltage and frequency change when Intel Turbo Boost Technology is used

*“Single-threaded applications realize the maximum benefit from Turbo Boost technology, because the core has more power and thermal budget available to it to boost its frequency.”*

*“In the ultra-low voltage segments, the use of Turbo Boost technology can increase the frequency of a core by 80%.”*

*“For graphics-intensive workloads, the Graphics Engine frequency is boosted to deliver increased performance while staying within the thermal and power envelope.”*

Single-threaded applications realize the maximum benefit from Turbo Boost technology, because the core has more power and thermal budget available to it to boost its frequency. The more power-constrained the platform is, the higher the benefit from Turbo Boost technology, because the multi-core processor has to be voltage scaled to fit in the power envelope. For example, in the ultra-low voltage segments, the use of Turbo Boost technology can increase the frequency of a core by 80%.\*

To activate turbo mode, the operating system needs to request the maximum performance state (P0 state). When this state is requested, Nehalem computes the new target voltage and frequency based on the number of active cores, the core temperature, as well as power delivery and thermal constraints. Note that as the voltage is raised, the leakage power and the temperature of the core increase, thereby eating into the power headroom initially available: this is taken into account when choosing turbo frequency.

In client products, codename Westmere, the Graphics Engine and a two-core 32nm version of Nehalem are on the same package. This allows the package power to be shared between the Graphics Engine and the processor, depending on the workload. For graphics-intensive workloads, the Graphics Engine frequency is boosted to deliver increased performance while staying within the thermal and power envelope.

## **Bringing it All Together: The Power Control Unit**

The Power Control Unit (PCU) is responsible for all of the power-management activities on the die. The PCU monitors several things, such as the performance and idle state requests from multiple threads; core and Uncore temperature; and power consumed by the package. Based on various inputs, the PCU adjusts voltage and frequency of the cores; controls the clock distribution, power gates and sleep transistors; and manages all the package power actions. The algorithmic complexity of multi-core and package power-management was significant enough that in the Nehalem family of products, we implemented these operations in firmware (PCU code—Pcode). Pcode executes on a custom, lightweight, 16-bit, micro-controller. The micro-controller has fixed-length instruction decoding and contains a 16-bit multiplier. Pcode instructions are stored in an on-die 24 KB ROM. An 8KB SRAM provides storage for data. Hardware assist engines were added to reduce the latency of operations, such as a core frequency change or a package state action.

\*[http://ark.intel.com/products/49158/Intel-Core-i7-660UM-Processor-\(4M-Cache-1\\_33-GHz\)](http://ark.intel.com/products/49158/Intel-Core-i7-660UM-Processor-(4M-Cache-1_33-GHz))  
Nominal Clock speed 1.33 GHz; Turbo 2.4 Ghz



A time-multiplexed bus was added between each core, the Uncore, and the PCU to collect sensor information, such as temperature, core activity, and package current. Pcode uses this information in conjunction with values programmed into on-die, non-volatile storage (fuses). Values stored in fuses allow tuning of parameters that are process-dependent, such as leakage sensitivity to temperature and voltage, on a per-part basis.

To provide the ability to make Pcode updates after the part is manufactured, “patch” capability was added. Having patch capability was key to delivering all the planned benefits while reducing the cost of development for the product.

## Summary

Nehalem delivers several breakthrough technologies for energy-efficient computing. These technologies are the foundation of future processors. As form factors continue to shrink, and consumers demand higher performance with increased battery life, next-generation processors will face greater challenges in the mobile space. In the desktop and server segments, minimizing the cost of power delivery and cooling while packing in more computation horsepower continues to drive lower-power consumption requirements. Improving energy efficiency involves optimizations in several areas: process technology, circuit design, architecture, floorplan, package, platform, testing methodology, as well as software. As evidenced by the Nehalem processor, dynamic and adaptive power-management mechanisms will be key to maximizing performance in an energy-constrained world.

## Acknowledgements

The authors thank Bob Greiner, Stephan Jourdan, Jerry Shrall, Ray Ramadorai, Shaun Conrad, Craig Topper, Dave O’Brien, Arun Krishnamoorthy, Rajesh Kumar, Mark Balmer, Madhusudanan Seshadri, and Gary Jones for their invaluable contributions during the development of the Nehalem power-management system.

## References

- [1] Wilfred Gomes, et al. "Circuit and Process Innovations to Enable High-performance, and Power and Area Efficiency on the Nehalem and Westmere Family of Intel® Processors," *Intel Technology Journal*, Volume 15, Issue 1, 2011.

## Authors' Biographies

**Steve Gunther** was the lead power architect for the Nehalem family of processors. His work interests include all things related to silicon power consumption. Steve earned his BS degree in Computer Engineering from Oregon State University in 1987. Currently, he is a Principal Engineer in the Micro-processor Development Group and serves as the lead power-management architect for a next-generation Intel processor. Steve holds 20 patents, and he is the recipient of four Intel Achievement Awards.

**Anant Deval** was the PCU design lead on the Intel Nehalem family of processors. He is interested in low-power technologies from circuits to architecture. After graduating from Arizona State University with a MS degree in 1997, he worked on the Intel® Pentium® 4 processor. He is currently a Principal Engineer in the Microprocessor Development Group. He has several patents and received the Intel Achievement Award for the Lynnfield CPU. Anant can be reached at [anant.s.deval@intel.com](mailto:anant.s.deval@intel.com).

**Rajesh Kumar** is an Intel Fellow and Director of Circuit & Low Power Technologies. In this capacity he directed the development of circuit and power technologies as well as interface to process technology for the Nehalem & Westmere family of processors. He is currently directing similar work on Intel's 22nm TOCK and manages the circuit technology team in the Microprocessor Development Group. Rajesh received a Master's degree from the California Institute of Technology in 1992 and a Bachelor's degree from the Indian Institute of Technology in 1991, both in Electrical Engineering. His e-mail is [rajesh.kumar@intel.com](mailto:rajesh.kumar@intel.com)

**Edward (Ted) Burton** is a Senior Principal Design Engineer working on circuit and power technology development in the Intel Architecture Group (DEG), and he is currently working on advanced power-delivery technologies slated for use on all of Intel's future 32- and 64-bit microprocessors. Since coming to Intel in 1992, Ted has received six Intel Achievement Awards. He is listed as an inventor on over 40 patents, and he has authored or co-authored a handful of papers presenting ground-breaking work. A paper on metastability, which was published in the *IEEE Journal of Solid State Circuits* in 1999, is cited by 75 other published papers. A paper presented at the 1994 IEEE International Solid State Circuits Conference discussed a phase-tolerant data-communication router for multi-processor super-computer backplanes. An additional paper discussing phase-tolerant communications was presented at the 1995 IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing. Other papers appeared in electronics trade magazines EDN and Electronic Design. Ted served as a discussion panelist at the 2000 VLSI Technology Symposium.

Prior to joining Intel, Ted was a Senior Integrated Circuit Design Engineer for Signetics Corporation (1983–1992). One of his Signetics patents was awarded the \$10,000 “gold” prize as the company's most valuable patent of 1989. He received a Bachelor's degree in Applied Physics from BYU in 1988.

## Copyright

Copyright © 2011 Intel Corporation. All rights reserved.

Intel, the Intel logo, and Intel Atom are trademarks of Intel Corporation in the U.S. and other countries.

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

Requires an Intel® HT Technology enabled system, check with your PC manufacturer. Performance will vary depending on the specific hardware and software used. Not available on Intel® Core™ i5-750. For more information including details on which processors support HT Technology, visit <http://www.intel.com/info/hyperthreading>

Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM). Functionality, performance or other benefits will vary depending on hardware and software configurations. Software applications may not be compatible with all operating systems. Consult your PC manufacturer. For more information, visit <http://www.intel.com/go/virtualization>

Requires a system with Intel® Turbo Boost Technology capability. Consult your PC manufacturer. Performance varies depending on hardware, software and system configuration. For more information, visit <http://www.intel.com/technology/turboboost>

# THE FEEDING OF HIGH-PERFORMANCE PROCESSOR CORES—QUICKPATH INTERCONNECTS AND THE NEW I/O HUBS

## Contributors

**Gurbir Singh**

Intel Corporation

**Robert Safranek**

Intel Corporation

**Nilesh Bhagat**

Intel Corporation

**Rob Blankenship**

Intel Corporation

**Ken Creta**

Intel Corporation

**Debendra Das Sharma**

Intel Corporation

**David L Hill**

Intel Corporation

**David Johnson**

Intel Corporation

**Robert A. Maddox**

Intel Corporation

## Index Words

QuickPath Interconnect

QuickPath Interface

NUMA

Dual I/O Hub

*“The class of high-performance cores, used in the Intel® microarchitecture code name Nehalem and Intel® microarchitecture code name Westmere, consumes gigabytes of data per second under normal operating conditions.”*

## Abstract

The Intel family of processor technologies, delivers a new level of performance in a repartitioned platform. We repartitioned the platform based on the integration of the memory controller into the processor socket, the QuickPath Interconnect (link-based architecture), and a scalable I/O subsystem (based on the IO Hub). This article provides a high-level overview of both the QuickPath Interconnect and the I/O subsystem of this new generation of platforms. (The integration of the memory controller into the processor socket is not discussed in this article.)

## Introduction

The class of high-performance cores, used in the *Intel® microarchitecture code name Nehalem* and *Intel® microarchitecture code name Westmere*, consumes gigabytes of data per second under normal operating conditions. They require access to system memory and I/O with low latency and high bandwidth, and they depend upon a very capable support system that allows cooperative interaction with other processors for even higher performance (as in those typically used for server systems). In this article, we describe the QuickPath Interconnect (Intel® QuickPath Interconnect (Intel® QPI)) that enables multiple processor sockets to work together efficiently and communicate optimally with the I/O Hub system interfaces.

## The QuickPath Interconnect

The QPI is Intel’s latest interconnect technology for multi-processor systems. Prior to the Nehalem-Westmere processor generation, the Frontside Bus (FSB) had been Intel’s interconnect for all the Intel Architecture processors. The FSB interconnect dates back to the mid-90s and was designed to support up to four processors and one memory controller on a single bus.

Several trends in the evolution of technology made it evident that we had to take a fresh look at the platform architecture level of a microprocessor system. Intel’s latest generation of processors, with multiple cores on a single die, demanded higher data and instruction bandwidth. Multiple processors in a system required more memory bandwidth than a single memory controller could economically provide. In addition, in accordance with “Moore’s Law,” it had become economical to integrate memory controllers onto the same die as the processor cores and thereby create very modular computational units. These modular processor units (with memory-controller) can be connected together with an appropriate interface to create powerful multi-processor systems.

We took a fresh look at the needs of such multi-processor systems and designed the QPI taking into account the needs and characteristics of these platforms.

The key objectives of the new interface were these:

- *High performance and bandwidth.* The goal was to radically improve system bandwidth and meet the latency goals of the new generation of processors being designed. In addition, the interface also had to allow for an increase in bandwidth to meet future demands.
- *Software and processor family compatibility.* The interface had to be completely transparent to the operating system and user software once the system interface had been properly initialized. Moreover, this interface had to be usable in future platforms, and it had to support both the Itanium® and Intel Architecture family of processors.
- *Low cost.* The interface had to be economical to build and it had to make efficient use of signals, pins, and traces.
- *Scalability.* The interface had to be architected with an eye on all reasonable future system topologies and it had to provide the necessary capabilities to support these future needs.
- *Power efficiency.* The interface had to be power efficient as well as provide an infrastructure to support power control and management capabilities of different platforms.
- *Reliability and serviceability.* The interface had to provide the necessary features for robust, error-free operation of the system even in the event of signal interference and disruption. Moreover, the interface needed capabilities to support robust memory system features in the system, such as memory mirroring, DIMM sparing, memory migration, etc.
- *New technology support.* The interface architecture had to provide capabilities and headroom to support new traffic types with different characteristics. Examples of the new traffic are media streams with real-time requirements, encrypted data streams for advanced security, or new commands required to interact with node controllers for more scalable systems.

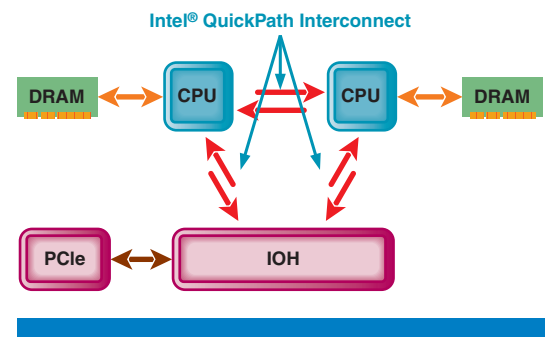
Our QPI was created to meet all these objectives.

### The Anatomy of an Intel QuickPath System

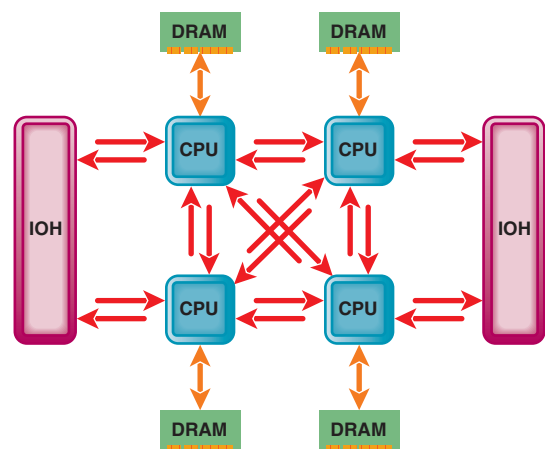
Figures 1 and 2 provide examples of two possible configurations—the former is a two-processor platform, the latter a four-processor design. Both can be built with the QPI. Each processor typically has a memory controller on the processor die, allowing systems to be more scalable (i.e., as processor capacity is added, memory capacity/bandwidth is increased). However, this is not essential and systems can have separate, discrete memory controllers. Similarly, the I/O subsystems can either be incorporated onto the same die as the processors or they can be built as separate components. (In the platforms shown in Figures 1 and 2, the I/O subsystem is based on the I/O Hub (IOH), codename Tylersburg.

*“The QPI is Intel’s latest interconnect technology for multi-processor systems.”*

*“The FSB interconnect dates back to the mid-90s and was designed to support up to four processors and one memory controller on a single bus.”*

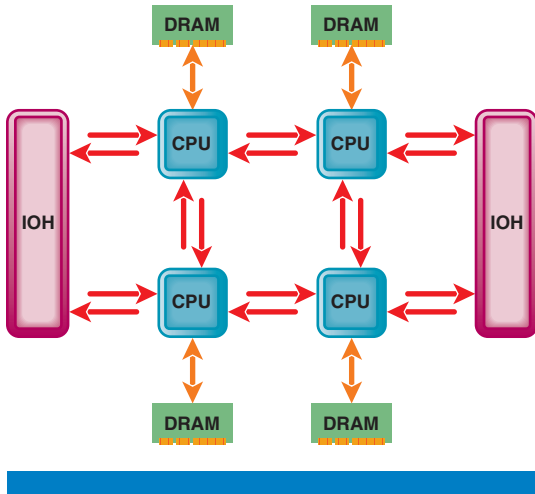


**Figure 1:** Platform with two processors employing Intel® QuickPath Interconnect technology (IOH = I/O Hub)



**Figure 2:** Platform with four processors employing Intel® QuickPath Interconnect technology

*“Each processor typically has a memory controller on the processor die, allowing systems to be more scalable (i.e., as processor capacity is added, memory capacity/bandwidth is increased).”*



**Figure 3:** Partially connected platform with Intel® QuickPath Interconnect technology

*“Each processor has a direct link to every other processor in the system.”*

*“The physical layer manages data transfer on the signal wires, including the electrical levels, timing, and it also manages logical issues involving sending and receiving each bit of information across the parallel lanes.”*

*“If a message handed up from the link layer is destined for an agent in another device, the routing layer forwards it to the proper link to send it on.”*

These platforms have multiple QPI links (indicated by the red arrow pairs in the figures) connecting the devices to each other. Each link can operate independently of the other links. The performance of such systems can be very high, particularly if the processors are allocated independent tasks, working on data that are optimally distributed across the different memory controllers and close to their own processors. Most current operating systems do implement such system configurations via Non-Uniform Memory Accesses (NUMA), in which the operating system attempts to place the data in memory physically “close” to the processor processing the information. Such link-based systems have much higher aggregate system bandwidth—and correspondingly higher performance.

The systems shown in Figures 1 and 2 are fully connected, meaning that each processor has a direct link to every other processor in the system. However, it is possible to build systems, using QPI, where the devices do not connect to all other devices. Figure 3 shows a four-processor system where each processor uses only two sets of links to connect to the other processors.

### Architecture Layers

The functions performed by QPI are logically grouped into four different layers, as illustrated in Figure 4. Following are the four layers of the QPI architecture:

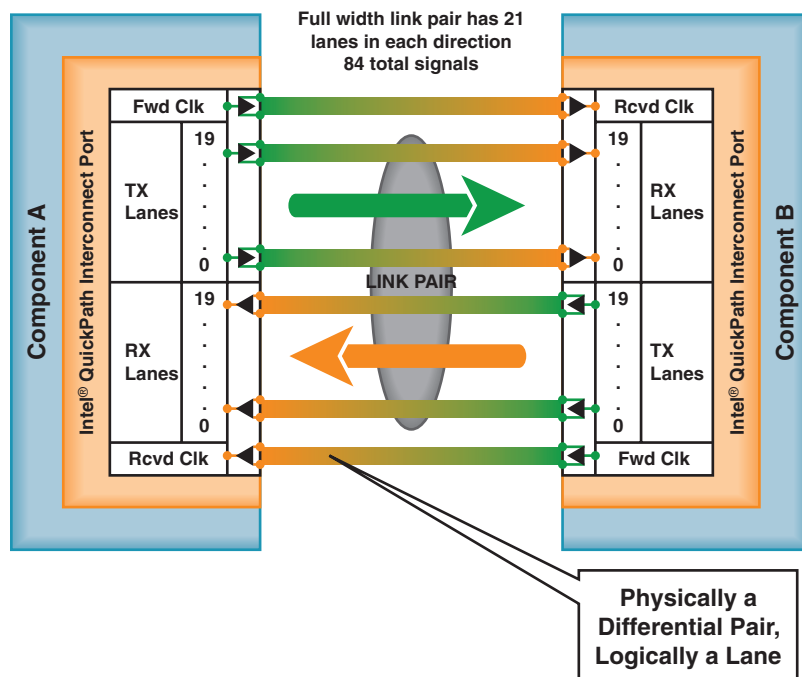
- The *Physical Layer*, which is responsible for dealing with details of the operation of the signals on a particular link between two agents. The physical layer manages data transfer on the signal wires, including the electrical levels, timing, and it also manages logical issues involving sending and receiving each bit of information across the parallel lanes.
- The *Link Layer*, which is responsible for handling flits of information. The link layer also manages the flow of these messages and handles errors that may occur during their transfer on that link.
- The *Routing Layer*, which is responsible for ensuring that messages are sent to their proper destinations. If a message handed up from the link layer is destined for an agent in another device, the routing layer forwards it to the proper link to send it on. All messages destined for agents on the local device are passed up to the protocol layer. The implementation details of this layer vary from one type of device to another. For example, processors that are not required to route traffic from one QPI link to another may not have a full routing layer.
- The *Protocol Layer*, which has multiple functions. This layer manages cache coherence for the interface by using a Writeback protocol. It also has a set of rules for managing non-coherent messaging and plays an integral part in cache coherency. Additionally, the protocol layer is responsible for system-level functions such as interrupts, memory mapped I/O, and locks. One major characteristic of the protocol layer is that it deals with messages across multiple links, involving multiple agents in multiple devices. Lower layers typically deal with only two directly connected devices.

In Figure 4 we introduce three terms that identify the granularity of the information being exchanged between the layers. These terms are phits, flits, and packets. We now examine these terms, and each layer's functions, in more detail.

Note that this layered architecture allows for a great deal of implementation flexibility and future growth, all within the scope of QPI. This modular approach also allows for extensions to be accommodated in an incremental manner. Therefore, in later sections of this article when we talk of certain characteristics of QPI, we are pointing out things that reflect current implementations. Those comments should not be construed as limitations on the overall scope of the QPI.

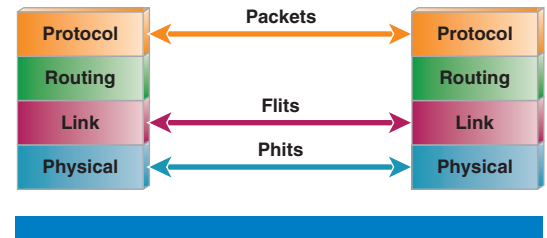
### The Physical Layer

The physical layer defines the operation and characteristics of the individual signals of a QPI link. We use high-speed differential signaling, with 20 differential pairs in one direction creating one link. A clock lane accompanies the set of 20 data lanes. One link in each direction completes the connection (referred to as a link pair). Figure 5 shows the signals of two QPI links, forming a link pair between the two devices.



**Figure 5:** Physical interface of the Intel® QuickPath Interconnect

Eighty-four pins are used to carry all the signals of one QPI link pair operating at its full width. In some applications, the link pair can also operate at half or quarter widths in order to reduce power consumption or work around failures. The unit of information transferred in each unit of time by the physical layer



**Figure 4:** Layers defined for the Intel® QuickPath Interconnect architecture

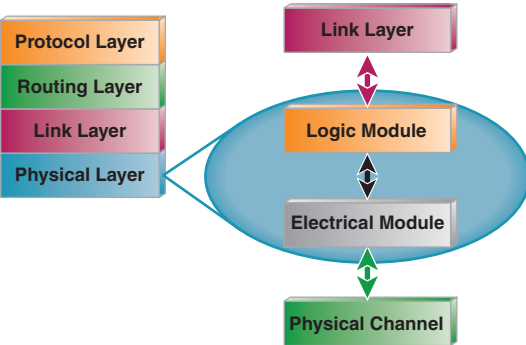
*“This modular approach also allows for extensions to be accommodated in an incremental manner.”*

*“We use high-speed differential signaling, with 20 differential pairs in one direction creating one link.”*

*“Eighty-four pins are used to carry all the signals of one QPI link pair operating at its full width.”*



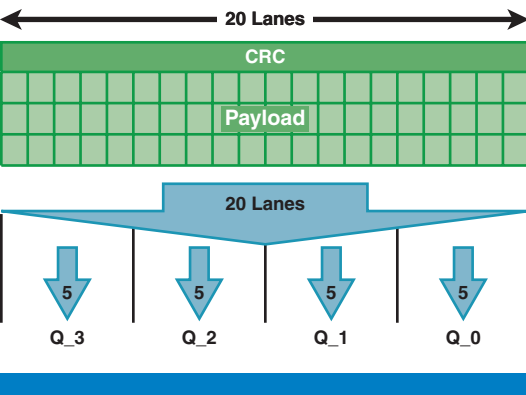
*“Typical signaling speeds of the link in current products calls for operation at 6.4GT/s for systems with short traces between components, and 4.8GT/s for longer traces like those found in large multi-processor systems.”*



**Figure 6:** Logical and electrical sub-blocks of the QPI physical layer.

*“Two connected link layers communicate with each other at the granularity of a flit, which is an acronym for flow control unit. In QPI, a flit is always 80 bits of information.”*

*“Groups of flits are acknowledged if they are free of errors.”*



**Figure 7:** Mapping 20 QPI lanes into four quadrants

is termed a phit, which is an acronym for physical unit. In the example shown in Figure 5, each phit contains 20 bits of information. Typical signaling speeds of the link in current products calls for operation at 6.4GT/s for systems with short traces between components, and 4.8GT/s for longer traces like those found in large multi-processor systems.

The physical layer is composed of digital and analog sections as shown in Figure 6.

The analog or electrical section manages the transmission of the digital data on the traces. This section drives the appropriate signal levels with the proper timing, relative to the clock signal, and then recovers the data at the other end and converts them back into digital data.

The logical portion of the physical layer interfaces with the link layer and manages the flow of information back and forth between them. It also handles initialization and training of the link and manages the width of operation.

**The Link Layer**

The QPI link layer controls the flow of information across the link and ensures that the information is transferred reliably. It also abstracts the physical layer into independent message classes and virtual networks that are required for the upper layers of the interface. We now look at these functions of the link layer.

Two connected link layers communicate with each other at the granularity of a flit, which is an acronym for flow control unit. In QPI, a flit is always 80 bits of information. Every flit contains 72 bits of message payload and 8 bits for the CRC checksum. The physical layer manages the transformation between flits and phits transparently. Figure 7 shows the subdivision of the 20 possible lanes into four quadrants (labeled Q\_0 to Q\_3 in Figure 7) of five lanes each. Flits are mapped onto the available physical lanes by the physical layer.

The link layer handles the flow of data and ensures that only as much data are sent to the link as the receiving agent can accept without overruns. It also ensures that data are transferred reliably across the link. Every flit received by the link layer is checked for errors, and groups of flits are acknowledged if they are free of errors. Otherwise, the receiving link layer requests retransmission of the flit with errors and all flits subsequent to it that may have been transmitted. Both short transient errors and burst errors that affect several flits can be corrected in this manner. Moreover, the order of transmission of the flits is maintained.

The link layer abstracts the physical link of QPI into a set of message classes, and each class operates independently. The message classes are similar to the types of mail handled by the post office. For example, you can request the post office to send a letter as ordinary first-class mail, or registered mail, or even as express mail. You may also have different types of things to send, a letter and a large package for example, and you may wish to use different delivery mechanisms for each item. Each of these classes of mail is handled independently of the other, very much like the message classes of the QPI link

layer. We briefly introduce the message classes here, but go into more detail about their operation and usage later in this article. These are the six message classes:

- Home (HOM)
- Data Response (DRS)
- Non-Data Response (NDR)
- Snoop (SNP)
- Non-Coherent Standard (NCS)
- Non-Coherent Bypass (NCB)

A collection of the six message classes is called a virtual network. QPI supports up to three independent virtual networks in a system. These are labeled VN0, VN1, and VNA. A basic one- or two-processor system can be implemented with just two networks, and typically VN0 and VNA are used. All three networks are usually used in multiple processor systems, ones that use the extra networks to manage traffic loading across the links; and also to avoid deadlocks and to work around link failures.

### The Routing Layer

The routing layer directs messages to their proper destinations. Every packet on a QPI link contains an identifier that states its intended destination. The routing layer logic contains several routing tables that indicate which physical link of a processor is the best route to a particular destination. These tables reflect the physical topology of the system. Whenever the link layer hands a message to the routing layer, the routing layer looks up the destination address in the tables and forwards the message accordingly. All messages directed at caching or home agents in a local component are sent to corresponding internal elements. Messages destined for agents in other sockets are sent down the appropriate QPI links identified in the tables.

The routing tables are set up by the firmware when the system is first powered up. Small systems usually run with these values unchanged. Multi-processor systems typically have more elaborate routing tables that contain information about alternative paths to reach the same destination. These can be used to help redirect traffic around a link that is heavily loaded. Fault-resilient, multi-processor systems can also use this information to work around failures in one or more links.

The routing layer can also help partition and reconfigure multi-processor systems into several smaller systems that logically operate independent of each other while sharing some of the same physical resources (i.e., the QPI infrastructure). Intel offers Xeon® and Itanium® processor servers that implement several of these features in their routing layers to provide high reliability.

*“QPI supports up to three independent virtual networks in a system.”*

*“Every packet on a QPI link contains an identifier that states its intended destination. The routing layer logic contains several routing tables that indicate which physical link of a processor is the best route to a particular destination.”*

*“The routing layer can also help partition and reconfigure multi-processor systems into several smaller systems that logically operate independent of each other while sharing some of the same physical resources.”*

*“QPI offers flexibility in the way cache coherence is managed.”*

*“QPI is designed to provide very high system performance over a wide range of system configurations and workloads.”*

*“High system throughput is achieved by pipelining a large number of transactions between the processors and the memory controllers and handling those transactions simultaneously and independently.”*

### The Protocol Layer

The protocol layer is the highest layer in the QPI hierarchy. A primary function of this layer is to manage the coherence of data in the entire system by coordinating the actions of all caching and home agents. The protocol layer also has another set of functions to deal with non-coherent traffic. QPI uses the well-known MESI protocol (Modified/Exclusive/Shared/Invalid) for cache coherence. It adds a new state known as Forward (F) to the protocol to allow fast transfers of Shared data. Hence the term MESIF better identifies the coherent protocol.

QPI offers flexibility in the way cache coherence is managed in a typical system. Proper cache coherence management is a responsibility distributed to all the home and cache agents within the system; each has a part to play, and operational choices can be made. Cache coherence snooping can be initiated by the caching agents that request data. This mechanism, called source snooping, is best suited to small systems that require the lowest latency to access the data in system memory. Larger systems can be designed to rely more on the home agents to issue snoops. This is termed the *home snooped coherence mechanism*. It can be further enhanced by adding a filter or directory in the home agent that helps reduce the cache coherence traffic across the links.

### Performance of the Intel QuickPath Interconnect

QPI is designed to provide very high system performance over a wide range of system configurations and workloads. It provides an excellent range of capabilities and permits component designers to select the features that are best suited for their target systems. High-performance, small-scale systems, such as workstations and computationally intensive desktop machines, tend to benefit from the low latency and high efficiency of the source snoop cache coherence mechanism. This variant of the snooping mechanism is designed to provide data to the processors with the lowest latency possible, as it requires the fewest number of hops across the links. A source snooping approach also takes advantage of the low latency of cache accesses to emphasize forwarding data from one processor to another, rather than getting the data from slower DRAM memory systems. This approach reduces latency by about 25% over comparably sized home snooped systems, producing a significant performance benefit.

Server systems benefit from large memory capacity and high memory bandwidth that can be readily shared across the multiple processors in the system. QPI provides efficient mechanisms to handle data traffic from multiple memory controllers. Very high system throughput is achieved by pipelining a large number of transactions between the processors and the memory controllers and handling those transactions simultaneously and independently. Such systems can benefit from the home snooped mechanism of QPI where system bandwidth can be further optimized with snoop filters or directories built into the memory controllers.

This behavior allows the home agent, or the memory controller, to keep track of the agents that have requested a particular cache line and only query them to cut down on traffic for cache-coherence resolution. Product designers can choose from a wide range of snoop filters or directory mechanisms to help reduce the traffic across the links. The QPI coherence protocol provides considerable flexibility in this area.

Larger system configurations with tens of processors can also be readily built in a hierarchical manner. Groups of two to eight processors and a node controller can be connected to other such nodes, where the node controllers are tied to each other over a set of links. Such a system configuration of two tiers of interconnect is shown in Figure 8. The second level of interconnect between node controllers may use QPI or it may not—the platform architect makes that decision.

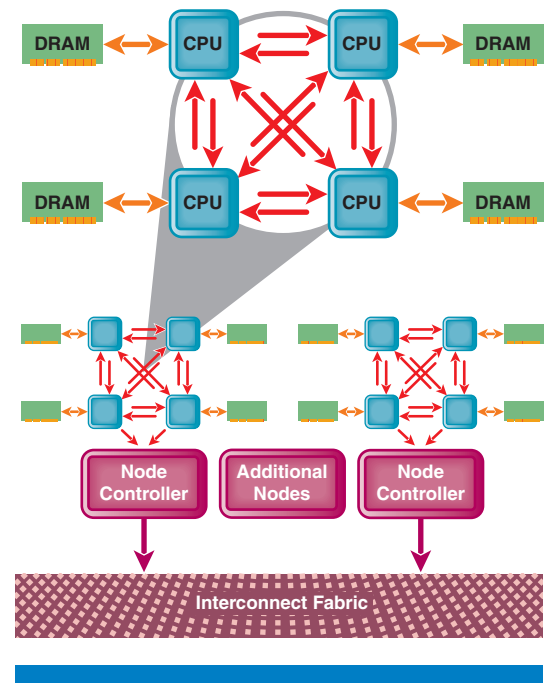
### Reliability of the Intel QuickPath Interconnect

QPI is designed to meet the demands of server systems where a premium is placed upon reliability, availability, and serviceability (RAS). The architecture offers several levels of error detection across the links and provides methods to correct those errors on the fly. However, if errors are seen repeatedly on one or more links, QPI has the capability to isolate the faulty lanes or links and work around the failed elements. QPI has mechanisms to try and recover from routine transient errors.

As mentioned earlier, QPI systems also may support memory mirroring where multiple memory controllers are paired to provide a more reliable memory storage capability. QPI handles all the traffic appropriately and ensures that data are reliably delivered to both of the memory controllers in the mirrored pair. In the event of the failure of one of the controllers, the system can continue operating by seamlessly drawing upon the data from the partner controller. QPI can indicate the occurrence of such a failure and permit replacement of the failed memory unit as the links provide the capability to support hot-plug of devices. In all cases, the goal is to keep the system up and running even in the face of several link failures.

### A Scalable I/O Subsystem to Match the Capability of the Processor/Memory Subsystem

The link-based architecture of QPI provides the infrastructure for the processor (and memory) to scale to meet the needs of different system requirements. To allow flexibility in the I/O subsystem, we developed a new component referred to as the I/O hub (or IOH). With the IOH, Intel has all the components required to address the increasing demand for system bandwidth while decreasing latency and power consumption. Ultimately, this system architecture effectively delivers computational capability within a tight power envelope. QPI-based platforms present an important advance in computing architecture.



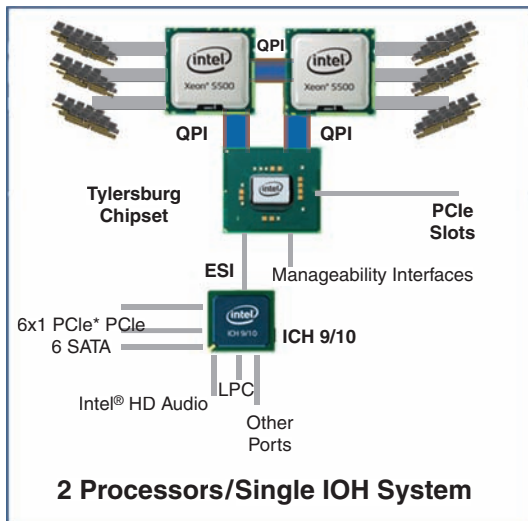
**Figure 8:** Hierarchical MP system with Intel® QuickPath Interconnect technology, using node controllers

*“The architecture offers several levels of error detection across the links and provides methods to correct those errors on the fly.”*

*“QPI systems also may support memory mirroring where multiple memory controllers are paired to provide a more reliable memory storage capability.”*

*“To allow flexibility in the I/O subsystem, we developed a new component referred to as the I/O hub (or IOH).”*

*“Traditional chipsets comprised a memory controller, I/O connectivity, and south-bridge functionality required to support baseline operating systems.”*



**Figure 9:** New platform paradigm showing the QPI links and the IO hub

*“The south bridge provides all the legacy I/O interconnects required for the ubiquitous peripherals that are found on PCs (such as USB, keyboard, Ethernet, SATA, and so on).”*

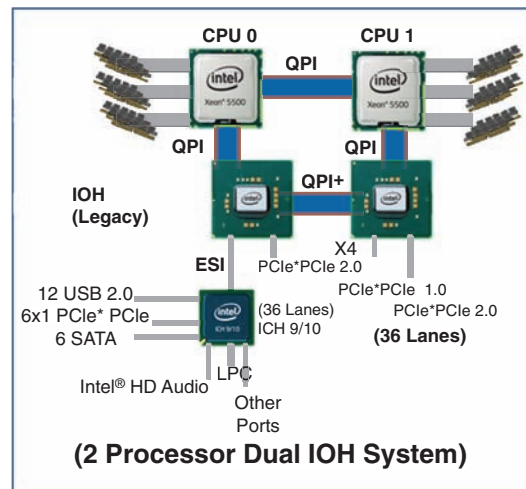
*“The platform supports two IOHs, perceived as a single, logical entity.”*

In traditional chipset architectures, the chipset defined the components outside of the CPU cores, which were all connected via a front-side bus interface. Traditional chipsets comprised a memory controller, I/O connectivity, and south-bridge functionality required to support baseline operating systems. In these new, repartitioned platforms, the memory controller is relocated to the same die as the CPU cores, the IOH provides the interface to high-bandwidth I/O, and the traditional south bridge component remains to support the standard I/O interfaces needed in computer systems today. Figure 9 illustrates the new QPI-based platform model.

In this platform, the I/O subsystem is built around the IOH and the south bridge. The IOH is responsible for translating PCIe protocols to the unordered QPI fabric and vice versa. The south bridge provides all the legacy I/O interconnects required for the ubiquitous peripherals that are found on PCs (such as USB, keyboard, Ethernet, SATA, and so on). The QPI to PCIe translation is a technical challenge, because the I/O interfaces must maintain backwards compatibility to keep the same software model, thereby relying on load/store ordered semantics. Furthermore, when the platform evolves from a single memory complex to multiple, distributed complexes, maintaining the ordering requirements of the I/O domain while promoting full performance adds a new level of complexity to the traditional chipset.

The IOH also provides new platform features such as I/O virtualization, security, and system manageability while continuing to provide prior-generation architecture features such as I/O acceleration.

Allowing I/O to scale up, the platform supports two IOHs, perceived as a single, logical entity. This feature enables smaller, dual-socket systems to support additional I/O expansion. An example of this configuration is represented in Figure 10. For larger systems with multiple caching agents, IOHs are added as separate entities as illustrated in Figure 3.



**Figure 10:** Dual-IOH platform architecture



## Dual IOH

Some single- and dual-socket server and workstation platforms require more PCIe lanes than provided by a single IOH. However, some processors are designed only for a single IOH. To enable two IOHs, we added special transaction flows between the IOHs to make them behave as a logical pair and appear as a single IOH to the CPU.

To accomplish this, we split the IOH responsibilities into *master* and *slave* functions for inbound requests to memory. The master IOH is the only IOH logically seen by the CPU. The slave IOH proxies all of its requests through the master IOH. The master is responsible for resolving any DRAM address conflicts that occur between the IOHs.

For outbound requests (CPU-initiated transactions targeting I/O), the master IOH does an additional level of address decoding to determine which IOH is the target of the request (master or slave).

## Speeds and Feeds

The IOH can support multiple topologies (refer to Figures 1, 2, 3, and 10 for some examples). The IOH can sustain an aggregate achievable I/O bandwidth of 34GB/s assuming 50% read and 50% write traffic into system memory.

The Nehalem IOH is the first Intel server chipset with PCIe 2.0 links. The IOH supports 36 PCIe 2.0 lanes, comprising two x16 links and one x4 link. The PCIe lanes can all run at either 2.5GT/s or 5GT/s data rate. These are differential links with an embedded clock and 8b/10b encoding. Each set of x16 link can form either of the following:

- One x16 link
- Two independent x8 links
- Four independent x4 links
- Three independent links of x8, x4, x4 widths

The x4 link can be configured as either one x4 link or two x2 links. In order to support this wide range of reconfiguration, the IOH implements ten PCIe controllers. In addition to the PCIe interfaces, the IOH has an Enterprise Server Interconnect (ESI) port, which is a x4 differential link, operating at 2.5GT/s with 8b/10b encoding; it connects to the south bridge (ICH10) to provide legacy I/O connection.

## I/O Virtualization

The IOH supports a rich set of features including hardware for optimizing I/O virtualization (VTd2), a manageability engine for system management, QuickData acceleration for CPU off-load of I/O, isochronous support, an integrated IOAPIC support to translate legacy interrupts from PCIe to Message Signaled Interrupts (MSI) for improved performance, and a rich set of RAS features.

*“The IOH responsibilities into master and slave functions for inbound requests to memory. The master IOH is the only IOH logically seen by the CPU. The slave IOH proxies all of its requests through the master IOH.”*

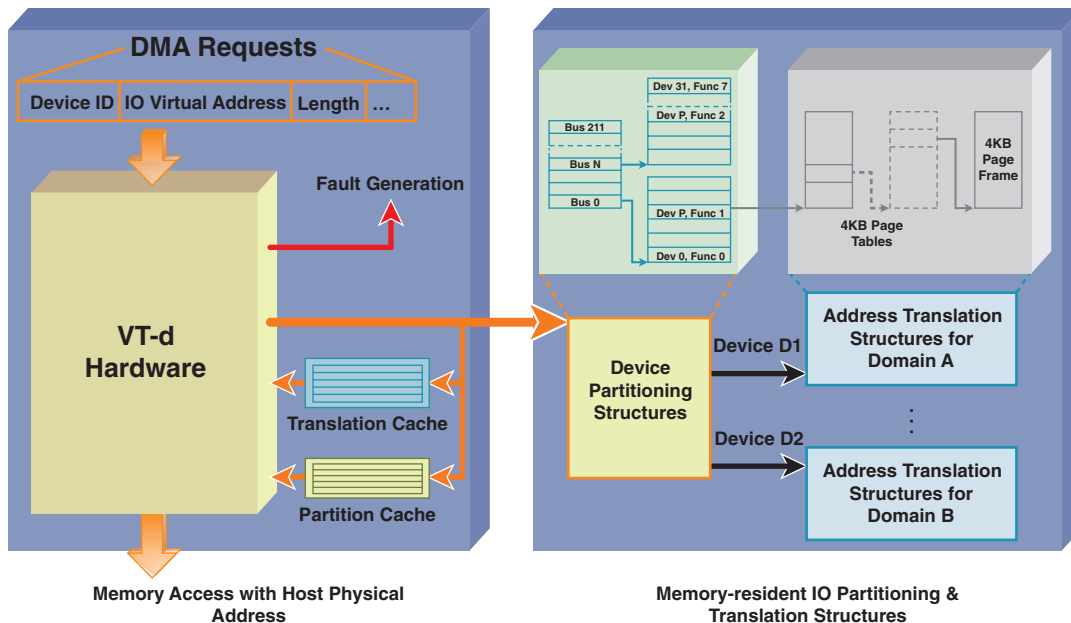
*“The IOH can sustain an aggregate achievable I/O bandwidth of 34GB/s.”*

*“The Nehalem IOH is the first Intel server chipset with PCIe 2.0 links.”*

*“In addition to the PCIe interfaces, the IOH has an Enterprise Server Interconnect (ESI) port, which is a x4 differential link, operating at 2.5GT/s with 8b/10b encoding; it connects to the south bridge (ICH10) to provide legacy I/O connection.”*

*“The address translation is based on the requestor’s bus, device, function (BDF), and the 4KB granular page address. Because the translation is based on BDF, the IOH enables the virtual machine monitor (VMM) to assign different functions to different virtual machines (VMs).”*

In a virtualized environment, all DMA memory accesses and MSIs, originating from each of the ESI/PCIe devices or internal sources (such as QuickData DMA engine, manageability engine, and root ports), require address translation to convert the Guest Physical Address (GPA) to a Host Physical Address (HPA). This is the purpose of the IOH VTd2 engine. The address translation is based on the requestor’s bus, device, function (BDF), and the 4KB granular page address. Because the translation is based on BDF, the IOH enables the virtual machine monitor (VMM) to assign different functions to different virtual machines (VMs). This policy eliminates the requirement for software emulation of I/O. The VTd2 translation function is illustrated in Figure 11. In addition to providing the translation function, the VTd2 engine also checks for access rights of the specific access and aborts any unauthorized access to provide for isolation and security between VMs.



**Figure 11:** VT-d DMA remapping

To keep the address translation latency to a minimum, the IOH maintains an IOTLB (implemented as a non-coherent cache) storing the BDF along with the page translation and access privilege for that BDF. To handle page misses in the IOTLB in an efficient manner, the IOH also maintains a context-entry cache that does the 2-level translation from the BDF to the address translation



structure, a L1/L2 cache for the first two levels of page translation, and an L3 cache for the third level of page translation. On a page miss, the VTd2 engine will walk through these structures to obtain the HPA. For isochronous accesses, there are three levels of page walk beyond the cache entry translation to ensure the QoS guarantee.

VTd entry invalidation can be done in three levels: domain specific, device-specific context cache, or IOTLB invalidation. Invalidation can be done either through writes to VTd2 configuration registers or by queuing up invalidation requests in main memory for better performance. The IOH supports address translation caching by PCIe devices conforming to the PCI-SIG defined Address Translation Service (ATS) and Access Control Service (ACS). For translated addresses, although it bypasses the IOTLB translation, for enhanced isolation and security the IOH continues to check the device cache to ensure that the requestor is eligible to bypass the IOTLB translation.

### Data Integrity and Other Improved RAS features

The IOH provides CRC protection and link-level recovery on error detection for each of its high-speed links (PCIe, ESI, and QPI). Poison support (which is the ability to explicitly mark packets with errors) is provided throughout the IOH for error containment. All the internal data paths are CRC- or ECC-protected, and the configuration registers are parity-protected for highly reliable operation. QPI has an elaborate set of error logs. PCIe supports the optional Advanced Error Reporting structures defined in the specification. Errors are reported to the system by root ports either as message-signaled interrupts or through error pins to external components. The IOH supports live error recovery on PCIe, in which each error type can be programmed to reset the PCIe link where the error occurred. If a PCIe device connected to the IOH does not handle poison data, that link can be programmed to reset instead of relying on the device to tolerate it. The IOH supports individual hot-plug of all PCIe links.

### Coherent Transactions into System Memory

Unlike bus-based systems, the IOH must handle a distributed memory subsystem on an unordered fabric. To support this architecture, the IOH fully participates in system coherency while maintaining the ordering requirements of PCIe.

IOH writes comprise a Request-for-Ownership followed by an eventual write back. The first step in the write flow is to prefetch ownership (RFO) of the cache line as a CPU core would. This prefetching takes time, and serializing subsequent, ordered transactions behind it would negatively affect performance—dramatically. To pipeline PCIe transactions on the unordered QPI fabric, the IOH maintains a Write Cache to temporarily buffer the cache lines to be written. Once the line to be written is prefetched and ordering requirements are met, the new data (partial or full-line) are stored into the

*“The IOH supports address translation caching by devices conforming to the PCI-SIG defined Address Translation Service (ATS) and Access Control Service (ACS).”*

*“The IOH provides CRC protection and link-level recovery on error detection for each of its high-speed links (PCIe, ESI, and QPI).”*

*“The IOH supports live error recovery on PCIe, in which each error type can be programmed to reset the PCIe link where the error occurred.”*

*“To pipeline PCIe transactions on the unordered QPI fabric, the IOH maintains a Write Cache to temporarily buffer the cache lines to be written.”*

*“The algorithm allows two writes to be combined into a single write, if the second write is observed in the Write Cache before the first write has been committed.”*

*“Every DRAM access from the processor results in a snoop to the IOH.”*

*“This flow combines the ownership phase and the write-back phase into a single transaction allowing the Home Agent (associated with each memory controller) that is tracking all pending requests to detect all conflict scenarios from CPU to IOH without snooping the IOH.”*

Write Cache. When the new data are written to the Write Cache, the IOH will write the data back to main memory. However it should be noted that the data are available to the CPUs prior to writing to memory through the snooping of the IOH.

The IOH Write Cache also enables the combining of inbound writes to memory. QPI and DRAM both use a fixed 64-byte payload (cache line), so combining writes into 64-byte chunks significantly improves efficiency on both interfaces. The algorithm allows two writes to be combined into a single write, if the second write is observed in the Write Cache before the first write has been committed.

Reads from I/O to memory are treated as non-posted on PCIe and do not have the strict ordering of posted transactions used by writes. This fact allows the IOH to issue the reads out of order (following prior posted transactions) and complete the reads in any order on PCIe. QPI provides two read transactions that the IOH uses: Read Current (RdCur) and Read Code (RdCode). RdCur is optimal because it does not perturb cache state in caching agents such as the CPU core. RdCode perturbs cache state by evicting the line out of the caching agents. The IOH provides a mode to select the read type to use depending on the abilities of the CPU.

#### **Snoop Filter (Snoop-less IOH)**

A source snooping coherency protocol like QPI or the traditional front-side bus broadcasts snoops to all caching agents to ensure coherency. By using this approach, every DRAM access from the processor results in a snoop to the IOH and responses to the processor. This pattern could degrade usable data bandwidth on QPI, and it could increase processor-to-memory latency. Highly scalable systems avoid broadcast snoop bandwidth by caching state directory data for each cache line—a costly solution. We created a more efficient solution for smaller-scale, one- and two-socket systems; we use a flow on the QPI called “invalidating write” to avoid the snoop of IOH on *all* coherent transactions. This flow combines the ownership phase and the write-back phase into a single transaction allowing the Home Agent (associated with each memory controller) that is tracking all pending requests to detect all conflict scenarios from CPU to IOH without snooping the IOH.

#### **Bandwidth Scaling**

The QPI protocol is, among other things, a pre-allocated protocol for DRAM requests. This means that the DRAM Home Agent must have resources guaranteed for all active transactions. This design approach requires that all sources of requests on QPI must be aware of the limitation of the target

through a variable referred to as *Max Requests*. The number of requests needed is a function of the bandwidth and latency to DRAM. The IOH provides a BIOS programmable register to scale the Max Requests value to meet the bandwidth needs of a given platform and CPU, thereby allowing bandwidth improvements in the platform with future CPUs, if additional DRAM Home Agent resources are added.

### IOH Microarchitecture

Roughly speaking, the IOH microarchitecture can be broken up into four main blocks: the QPI logic, PCIe/ESI logic, integrated functions, and the interconnect between them. Figure 12 represents a high-level block diagram of the IOH. The PCIe units (physical, link, and transaction layer queues) make up narrower building blocks that can be aggregated to form the wider links. For example, a x16 link CRC block is formed by stacking four independent x4 link CRC blocks so that they work cooperatively when the link is a x16 link. Similarly, the transaction layer queues combine to form larger queues when a higher link width is used. For example, a x8 link gets twice the space of a x4 link and a x16 link gets four times the space of a x4 link.

The Central Data Path (CDP) is responsible for processing all inbound and outbound transactions and interacting with the QPI protocol layer in a way that reconciles the differences in ordering and bypassing rules between QPI and PCIe. The CDP checks PCIe ordering rules for transactions in the ESI/PCIe domains, performs the VTd checks and splits memory transactions from PCIe into cache-line aligned transactions to main memory. It maintains the address map of the system and correctly routes the transactions. The internal datapaths have a raw bandwidth of 51GB/s and can support a sustained bandwidth of 34GB/s in both inbound and outbound directions.

The QPI protocol layer maintains a combined buffer for the coherent Write Cache (for DMA writes, cache lines are obtained with ownership), non-coherent DMA buffers (coherent snapshot for DMA reads as well as peer-to-peer writes to another IOH in dual IOH mode), as well as outbound access (Memory-mapped I/O, I/O Port and Configuration cycles from the CPU or peer IOH in a dual IOH configuration). For inbound accesses, the protocol layer resolves the internal conflicts between different transactions to the same cache line and ensures that only one request to the same address is outstanding from the IOH on the QPI fabric. When more than one CPU exists, the IOH will send a request to the destination QPI home agent and a snoop to any other caching agents in the system. The QPI link layer and physical layer (including the analog front end) perform the layer-specific functions.

*“Roughly speaking, the IOH microarchitecture can be broken up into four main blocks: the QPI logic, PCIe/ESI logic, integrated functions, and the interconnect between them.”*

*“The transaction layer queues combine to form larger queues when a higher link width is used. For example, a x8 link gets twice the space of a x4 link and a x16 link gets four times the space of a x4 link.”*

*“The internal datapaths have a raw bandwidth of 51GB/s and can support a sustained bandwidth of 34GB/s in both inbound and outbound directions.”*

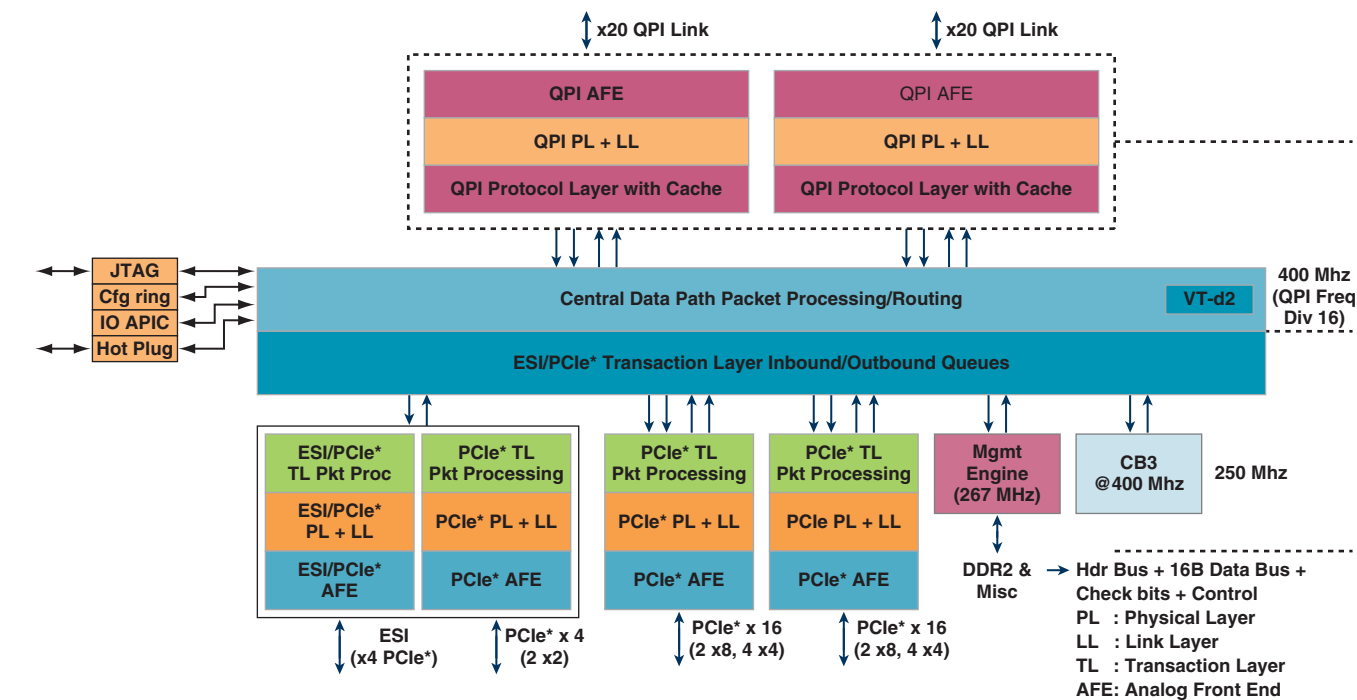


Figure 12: IOH block diagram

*“The QPI defines the physical, link, routing, and protocol layers. It provides the infrastructure to support the Nehalem and Westmere cores’ ability to consume gigabytes of data per second under normal operating conditions.”*

## Conclusion

In this article we provide a high-level overview of both the QPI and I/O subsystem of this new generation of platforms. It explains how the QPI enables multiple processor sockets to work together efficiently and communicate optimally with the I/O Hub system interfaces.

The QPI defines the physical, link, routing, and protocol layers. It provides the infrastructure to support the Nehalem and Westmere cores’ ability to consume gigabytes of data per second under normal operating conditions. In addition, the definition provides the scalability features of a platform by allowing processor sockets, memory capacity, and growth in the I/O subsystem.

In addition to the first QPI link-based system, the IOH is also the first Intel server chipset with PCIe 2.0\* links. It offers leadership I/O bandwidth with headroom for growth as well as the ability to provide high I/O fanout. It also offers leadership technology offering high-end user value with I/O virtualization, security, I/O acceleration, manageability, and isochrony.

## Authors' Biographies

**Gurbir Singh** is an Architect Senior Principal Engineer in the Intel Architecture Group. Gurbir led the architecture team defining the QPI specification. Gurbir has 32 years of experience in CPU and platform architecture. Gurbir was responsible for the architecture of many of the large processor caches and Intel FSB system interface definitions including the original P6, and the Pentium-4 quad-pumped extensions, prior to leading the Intel QPI definition. Gurbir holds a BS degree in Electrical and Electronics Engineering from the Indian Institute of Technology, Kharagpur, India and a MS in Electrical and Computer Engineering from Clemson University. His email is gurbir.singh at intel.com.

**Nilesh Bhagat** is a Senior Design manager in the Intel Architecture Group. Nilesh joined Intel in 1998, and he has 30 years industry experience primarily in Client and Server Chipset design. He was the Engineering manager for Tylersburg/Boxboro IOH Chipsets designed for the Nehalem/Westmere family of CPUs. Nilesh received a BS degree in Electrical Engineering from the University of Baroda and an MS degree in Computer and Electrical Engineering from Illinois Institute of Technology. His email is nilesh.bhagat at intel.com.

**Rob Blankenship** is an Architect Senior Engineer in the Intel Architecture Group. Rob joined Intel in 1997. He has been working in the area of coherency protocol for Intel chipsets since 1998. Rob was a key developer of the QPI specification. Rob holds numerous patents in the area of link coherency. Rob received his BS degree in Electrical Engineering from the University of Washington. His email is robert.blankenship at intel.com.

**Ken Creta** is an Architect Principal Engineer in the Intel Architecture Group. Ken joined Intel in 1990 with particular experience in high-end server chipset architectures and a broad knowledge of chip-to-chip interconnects. He is listed on numerous patents for chip interconnects including PCIe and Intel's QuickPath Interconnect. Ken was the Uncore lead architect for Intel's Sandybridge-EP microprocessor. Ken received his BSE degree in Computer Engineering from Arizona State University in 1991. His email is kenneth.c.creta at intel.com.

**Debendra Das Sharma** is an Architect Senior Principle Engineer in the Intel Architecture Group. Debendra is responsible for developing architecture, interconnect protocols, and server components. He is a leading contributor to the PCI-Express\* Generation 3 specification in the PCI-SIG. Debendra joined Intel in 2001 and led the development of the Twincastle multiprocessor Northbridge chipset, the Seaburg chipset in the Stoakley platform, the Tylersburg chipset in the Thurley platform, and the Boxboro chipset in the Boxboro platform. Prior to joining Intel, he was with Hewlett-Packard for seven years where he worked on the development of several server chipsets, including Superdome. Debendra earned a PhD degree in Computer Engineering from the University of Massachusetts, Amherst in 1994 and a Bachelor in Technology (Hons.) degree in Computer Science and Engineering from the Indian Institute of Technology, Kharagpur in 1989. He holds 36 patents in the areas of chipsets, interconnect, coherence protocol, high availability, and microarchitecture. His email is debendra.das.sharma at intel.com.

**David L Hill** is an Architect Senior Principal Engineer in the Intel Architecture Group. Dave joined Intel in 1993, and he has 30 years industry experience primarily in high-performance caches, interconnect controllers, and coherent memory systems. He was the Uncore lead architect of concepts and implementation for Nehalem and Westmere Xeon-5500/5600 and client products, and the Westmere product lead architect. Dave received his BS degree in Electrical Engineering from the University of Minnesota. His email is david.l.hill at intel.com.

**David Johnson** is a Special Circuits Design Lead in the Intel Architecture Group. David joined the Server Chipset Division of Intel in 1996. He has architected and designed many high-speed I/O phys in his 14-year career at Intel. David architected five source synchronous and common clock I/Os as well as a CAM cache for the 460GX chipset. He was the design lead for all I/Os on the 80870 SIOH server chipset, the lead for Hublink and DDR on the Placer HEDT chipset, and he worked on the first 8Gbps Intel PHY while in the Circuits Research Lab. On the Tylersburg and Boxboro Chipset program, David was the architect and technology lead for the QPI, PCIe2, and DDR3 interfaces. David is currently the Architect and Technology Lead for the 14nm ASDG ModPhy and Analog Library developments. His email is david.r.johnson at intel.com.

**Robert A Maddox** is a Staff Platform Applications Engineer in the Intel Architecture Group. Robert received an MS degree in Electrical Engineering from Georgia Tech. He joined Intel in 1998 and worked on the pre-silicon validation aspects of the IO hubs. He is currently a Staff Technical Marketing Engineer in the Server Platforms Group focusing on QPI, where he works with both internal teams and external companies on the application of this new platform interconnect. His email is bob.maddox at intel.com.

**Robert Safranek** is an Architect Principal Engineer in the Intel Architecture Group. Robert was a primary developer and author of the QPI specification, and he also led development of the processor QPI implementation architecture for Intel Xeon 5500/5600 products. Robert holds a BS degree in Electronic Engineering from the University of Nebraska and an MS degree in Computer Engineering from Portland State University. His email is robert.j.safranek at intel.com.

## Copyright

Copyright © 2011 Intel Corporation. All rights reserved.

Intel, the Intel logo, and Intel Atom are trademarks of Intel Corporation in the U.S. and other countries.

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

Requires an Intel® HT Technology enabled system, check with your PC manufacturer. Performance will vary depending on the specific hardware and software used. Not available on Intel® Core™ i5-750. For more information including details on which processors support HT Technology, visit <http://www.intel.com/info/hyperthreading>

Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM). Functionality, performance or other benefits will vary depending on hardware and software configurations. Software applications may not be compatible with all operating systems. Consult your PC manufacturer. For more information, visit <http://www.intel.com/go/virtualization>

Requires a system with Intel® Turbo Boost Technology capability. Consult your PC manufacturer. Performance varies depending on hardware, software and system configuration. For more information, visit <http://www.intel.com/technology/turboboost>



# ARCHITECTED FOR PERFORMANCE—VIRTUALIZATION SUPPORT ON NEHALEM AND WESTMERE PROCESSORS

## Contributors

**Bill Alexander**  
Intel Corporation

**Andy Anderson**  
Intel Corporation

**Barry Huntley**  
Intel Corporation

**Gil Neiger**  
Intel Corporation

**Dion Rodgers**  
Intel Corporation

**Larry Smith**  
Intel Corporation

## Index Words

Intel Vt-x  
VMX Transitions  
Guest-Segment Cache  
VMCS Caching  
Virtual Process Identifiers

## Abstract

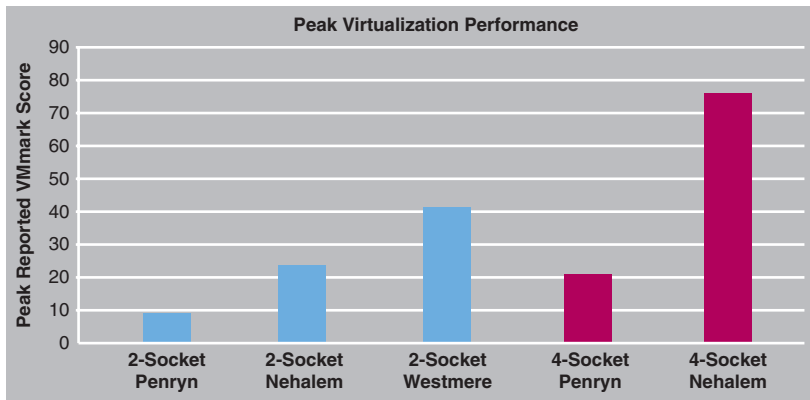
As use of Intel® Virtualization Technology for IA-32 and Intel 64 Architectures (Intel® VT-x) has grown, the demand for improved performance in virtualized systems has increased. The *Intel® microarchitecture code name Nehalem* and *Intel® microarchitecture code name Westmere*, have introduced extensions to Intel VT-x and its underlying hardware support to meet these increasing performance demands. This article discusses the architectural and microarchitectural improvements provided by these processors and how they translate into improved performance for software that uses this technology.

## Introduction

Intel® virtualization technology for IA-32 and Intel 64 architectures (Intel VT-x) comprises extensions to the instruction-set architecture that provide processor-level support for virtual machines [1]. Intel VT-x was first implemented in some generations of Intel Pentium® 4 processors. Because it simplifies significantly the task of virtualizing IA-32 and Intel 64 processors, VT-x has been adopted for use by all major vendors of virtualization software for those processors.

The Intel processors, codenames Nehalem and Westmere, include changes to improve the performance of software that uses Intel VT-x. Figure 1 illustrates the improvements in virtualization performance achieved by these platforms as measured by VMware’s VMmark\* benchmark. VMmark is a consolidation benchmark that measures aggregate system throughput for multiple tiles, or sets of web, mail, database, file, Java\*, and standby server virtual machines (VMs) executing concurrently. Tiles of VMs are added to the system for as long as the addition of these tiles increases throughput, with the peak throughput reported as the benchmark score. The bars in Figure 1 represent the peak-reported benchmark score by processor generation.

The illustrated performance improvements are attributable to a variety of factors, including increases in core counts and general architectural optimizations, in addition to virtualization-specific optimizations made in the design of Nehalem and Westmere.



**Figure 1:** The peak-reported VMmark score for each processor generation illustrates the significant improvements made by Nehalem and Westmere (Source: <http://www.vmware.com/products/vmmark/v1/results.html>)

As shown in Figure 1, 2-socket Nehalem and Westmere platforms deliver 2.8x and 4.5x VMmark performance improvements, respectively, over a corresponding Penryn 2-socket platform. The 4-socket Nehalem platform delivers a 3.8x improvement over the corresponding 4-socket Penryn platform.

In this article we present an overview of Intel VT-x, and we explain the extensions to the instruction set as well as the differences in the microarchitectures between the Intel Penryn family of processors and the Nehalem and Westmere processor generations. We discuss how these differences delivered the enhanced performance shown in Figure 1.

## Overview of Intel Virtualization Technology

Intel VT-x defines processor-level support for two classes of software: virtual-machine monitors (VMMs, sometimes known as hypervisors), which have full control of the processor; and guest software, which operates in a VM under the control of a VMM. Intel VT-x provides processor-level support for virtualization with a range of technologies that deliver VMX operation. There are two kinds of VMX operation: VMX root operation and VMX non-root operation. In general, a VMM runs in VMX root operation, and guest software runs in VMX non-root operation.

Transitions between VMX root operation and VMX non-root operation are called VMX transitions. There are two kinds of VMX transitions: transitions into VMX non-root operation (into a VM), which are known as *VM entries*; and transitions from VMX non-root operation (out of a VM) to VMX root operation (to the VMM), which are known as *VM exits*.

*“There are two kinds of VMX operation: VMX root operation and VMX non-root operation. In general, a VMM runs in VMX root operation, and guest software runs in VMX non-root operation.”*

*“A VMM accesses a VMCS with two instructions, VMREAD and VMWRITE. These instructions create a layer of abstraction between VMM software and the actual format of the VMCS region.”*

VMX non-root operation and VMX transitions are controlled by a data structure called a virtual-machine control structure (VMCS). Each VMCS is associated with a region of memory (the VMCS region). Software makes a VMCS *active* by executing the VMPTRLD instruction with a pointer to the associated VMCS region. Software uses the VMCLEAR instruction with that pointer when it has finished using a VMCS.

VMM software does not access the VMCS region directly by using ordinary memory reads and writes. Instead, a VMM accesses a VMCS with two instructions, *VMREAD* and *VMWRITE*. These instructions create a layer of abstraction between VMM software and the actual format of the VMCS region. This layer of abstraction also enables processors to cache active VMCSs in on-chip resources, separate from, and not necessarily coherent with, the VMCS region in memory.

One of the uses of a VMCS is to manage the state of guest software. Every VM exit saves the values of numerous processor registers into the VMCS; these values represent the state of guest software. Correspondingly, each VM entry loads those registers from the VMCS with the state of guest software.

## Contributions to Performance: Microarchitecture

Intel VT-x was designed so that microarchitectural enhancements could be made to processor implementations to improve the performance of VMX operation. Some performance enhancements were implemented by previous generations of Intel processors; however, the Nehalem and Westmere families of processors go much further, achieving significant performance improvements over previous generations of processors.

### Microarchitectural Enhancement: VMREAD and VMWRITE Instructions

As noted previously, software accesses a VMCS by using the VMREAD and VMWRITE instructions. Previous processors implemented the VMREAD and VMWRITE instructions primarily with microcode, along with a small amount of dedicated hardware. Because microcode was responsible for detecting various failure conditions, the performance of the instructions was limited with regard to both latency and throughput. Since these instructions are executed frequently by VMM software, Intel decided to add dedicated hardware to accelerate execution of VMREAD and VMWRITE on Nehalem. As a result, the extensive microcode was no longer required.

This change resulted in significant reductions in latency: the instructions are considerably faster than those of the previous processor generation, as summarized in Figure 2. The figure shows the latencies of both instructions for all processor generations that support Intel VT-x.

*“Intel decided to add dedicated hardware to accelerate execution of VMREAD and VMWRITE on Nehalem. As a result, the extensive microcode was no longer required.”*

Instruction	Prescott	Cedar Mill	Yonah	Merom	Penryn	Nehalem	Westmere
VMREAD	174	91	47	54	33	7	7
VMWRITE	170	118	40	48	31	7	7

**Figure 2:** VMREAD/VMWRITE timings as measured in clock cycles—lower is better  
(Source: Intel Corporation)

In addition, doing fault checking in hardware (instead of microcode) improves throughput, because it allows the instructions to be pipelined more efficiently. This is important, as the instructions are typically executed in batches.

The following items summarize the additional enhancements made to the implementation of the VMREAD and VMWRITE instructions:

- Expanded capabilities of an internal programmable logic array (PLA) that defines the size, formatting, and memory-offset details of the VMCS region.
- Dedicated hardware in the integer-execution unit to quickly format data.
- Dedicated branch hardware in the processor front-end to pre-compute any failure conditions (based on machine mode and other factors) so that, in the normal case, no execution bandwidth is spent to check for failure cases.
- Special hardware in the memory execution unit to provide a variable-sized store operation for physical addresses. This hardware allows microcode to access the VMCS region more efficiently.

The implementation of VMWRITE was also modified to support acceleration of VM entries. We discuss this change next.

#### Microarchitectural Enhancement: Guest-Segment Cache (Nehalem)

The VMCS includes fields containing the guest's values of the processor's segment registers: CS, SS, DS, ES, FS, GS, TR, and LDTR. The treatment of these registers by microcode is important because the registers control the processor's basic protection mechanisms; (2) processor generations often add microarchitectural enhancements to the segment-register implementation; and (3) the consistency checking of guest segment registers adds latency to VM entries.

The guest state of each segment register is represented by four fields in the VMCS: selector, base address, segment limit, and access rights. VMM software can access each of these fields independently by using the VMREAD and VMWRITE instructions. Intel VT-x specifies the format of these fields, called here the VMCS-segment format. The VMCS-segment format need not correspond to the processor's internal representations of the segment registers.

Thus, guest segment-register state comprises 32 fields in the VMCS (4 fields for each of the 8 registers). Every VM exit must save the segment registers' values into these 32 fields, and every VM entry must load those registers from those

*“Doing fault checking in hardware (instead of microcode) improves throughput, because it allows the instructions to be pipelined more efficiently.”*

*“The implementation of VMWRITE was also modified to support acceleration of VM entries.”*

*“Thus, guest segment-register state comprises 32 fields in the VMCS (4 fields for each of the 8 registers). Every VM exit must save the segment registers' values into these 32 fields, and every VM entry must load those registers from those 32 fields—as well as check their consistency.”*

*“The Nehalem processor accelerates VMX transitions by maintaining guest segment-register state in a dedicated on-chip, guest-segment cache.”*

*“Nehalem avoids this problem by having each VM exit save each segment register both in the guest-segment cache (in the wide-segment format) and in the VMCS region (in the VMCS-segment format).”*

*“VM entry loads a segment register from the guest-segment cache (by using the wide-segment format) if none of the VMCS fields for that register is dirty; otherwise, it loads the register from the VMCS region (by using the VMCS-segment format).”*

32 fields—as well as check their consistency. In particular, these consistency checks could add significant latency to a naïve VM-entry implementation in microcode.

The Nehalem processor accelerates VMX transitions by maintaining guest segment-register state in a dedicated on-chip, guest-segment cache. Nehalem optimizes the guest-segment cache by representing the four elements (selector, base address, segment limit, and access rights) of each segment register as a single quantity in what is called the wide-segment format; VMX transitions can thus save or load each segment register in a single operation.

The wide-segment format of the guest-segment cache contains the segment-register data from the VMCS but in a format that is different than that of the VMCS segment. If the guest segment-register state were maintained in the VMCS region only in the wide-segment format, VMREAD and VMWRITE would have to translate between the two formats, adding latency and potentially compromising the improvements to their performance, as described previously.

Nehalem avoids this problem by having each VM exit save each segment register *both* in the guest-segment cache (in the wide-segment format) and in the VMCS region (in the VMCS-segment format). The VMREAD and VMWRITE instructions can use the values in the VMCS region (by using the VMCS-segment format); while a subsequent VM entry could load the segment registers from the guest-segment cache (by exploiting the wide-segment format).

The VMWRITE instruction updates only the VMCS region and not the guest-segment cache. If software uses VMWRITE to modify a VMCS field for a segment register, that register’s value in the guest-segment cache is no longer valid. For this reason, a VM entry must not load a segment register from the guest-segment cache, if such a VMWRITE has occurred.

The fact that software must use the VMWRITE instruction (and not ordinary memory writes) to modify the VMCS allows the processor to track which fields a VMM has modified. The Nehalem processor does this by assigning each field in the VMCS a *dirty bit*. Every VM exit clears these dirty bits, while each execution of VMWRITE sets the dirty bit assigned to the field being written.

A subsequent VM entry uses these dirty bits to determine, for each segment register, whether the corresponding value in the guest-segment cache is valid. Each VM entry loads a segment register from the guest-segment cache (by using the wide-segment format) if none of the VMCS fields for that register is dirty; otherwise, it loads the register from the VMCS region (by using the VMCS-segment format).

The dirty bits allow an optimization beyond the use of the more efficient wide-segment format. Recall that the processor performs consistency checks to prevent a VM entry from loading inconsistent guest state. The checks

performed on the segment registers are especially time-consuming. The state saved by VM exit is always consistent (because it was in the segment registers). Thus, the segment-register state in the VMCS can be inconsistent only if VMM software has modified it by using the VMWRITE instruction. This implies that the consistency checks need not be performed for a segment register if none of the VMCS fields for that register is dirty. Because it is expected that a VMM will not modify guest segment registers when handling most VM exits, this optimization has the potential to reduce significantly the latency of most VM entries.

As noted earlier, VMM software should not attempt to modify a VMCS by writing directly to its VMCS region. Such writes for guest segment-register fields will not update the corresponding dirty bits. As a result, a subsequent VM entry may use the guest-segment cache, effectively ignoring the improper VMCS modification. This processor behavior is part of the Intel VT-x specification.

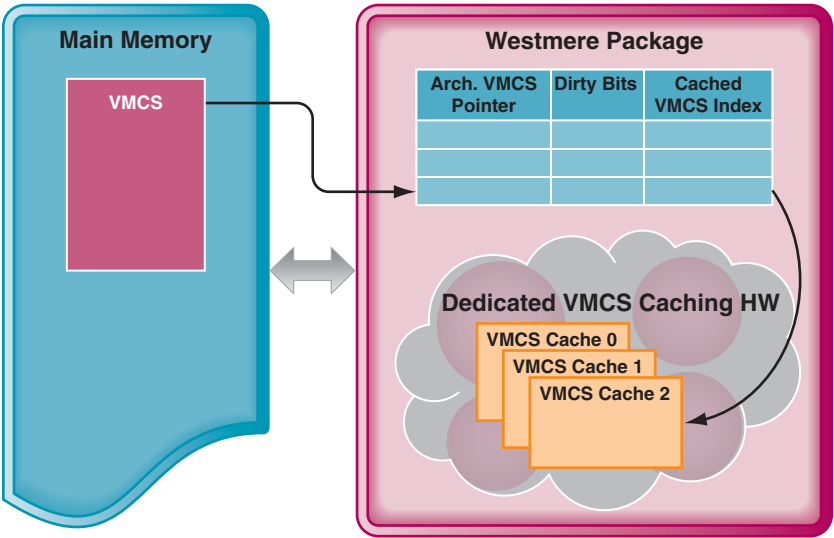
Nehalem’s guest-segment cache supports optimizations of VM entries relative to guest segment-register state. Westmere generalizes those optimizations with full VMCS caching.

**Microarchitectural Enhancement: Full VMCS Caching (Westmere)**

In the previous section we explained how Nehalem reduces VM-entry latency by maintaining guest segment-register state on-chip in a guest-segment cache that cannot be modified by software. Westmere generalizes this approach by caching the entire VMCS (see Figure 3), resulting in a corresponding extension of the performance benefits.

*“The consistency checks need not be performed for a segment register if none of the VMCS fields for that register is dirty.”*

*“Nehalem’s guest-segment cache supports optimizations of VM entries relative to guest segment-register state. Westmere generalizes those optimizations with full VMCS caching.”*



**Figure 3:** VMCS caching shadows the architectural VMCS from main memory to on-package dedicated hardware for faster, more controlled access (Source: Intel Corporation)



*“Intel VT-x requires a VMCS to be coherent with its VMCS region only after the VMCLEAR instruction is executed with a pointer to the VMCS region.”*

*“VM entry can skip any consistency checks pertaining entirely to VMCS fields for which the corresponding dirty bits are all clear.”*

The following are the other key differences between the Nehalem and Westmere approaches to the caching of VMCS state:

- Nehalem uses on-chip registers for the guest-segment cache; Westmere uses resources on the CPU package for a VMCS cache. These on-package resources are not accessible to software except through the VMREAD and VMWRITE instructions.
- On Nehalem, VM exits save guest segment-register state both to the guest-segment cache and to the VMCS region; on Westmere, VMX transitions use only the VMCS cache and do not use the VMCS region at all.
- Nehalem uses the guest-segment cache only for the current VMCS; Westmere may use the VMCS cache for other active VMCSs as well.

As noted earlier, software makes a VMCS active by executing the VMPTRLD instruction with a pointer to the associated VMCS region. Westmere implements the VMPTRLD instruction by copying the entire VMCS region from system memory to the VMCS cache (assuming that the VMCS is not already cached). All operations that use the VMCS (e.g., VMX transitions, and the VMREAD and VMWRITE instructions) access only the cached VMCS. While a VMCS is cached, its contents are no longer coherent with the associated VMCS region. Intel VT-x requires a VMCS to be coherent with its VMCS region only after the VMCLEAR instruction is executed with a pointer to the VMCS region. For this reason, Westmere implements the VMCLEAR instruction by copying the entire cached VMCS to the associated VMCS region (assuming that the VMCS was being cached).

As mentioned earlier, Nehalem’s guest-segment cache is not by itself sufficient to optimize VM-entry consistency checking. The same is true of Westmere’s VMCS cache. Westmere optimizes VM entries by generalizing Nehalem’s use of dirty bits.

Westmere supports a set of dirty bits for each cached VMCS. Each dirty bit corresponds to a set of VMCS fields. The processor maintains the dirty bits as follows:

- An execution of VMPTRLD that loads a new VMCS into the VMCS cache sets all the dirty bits for that VMCS.
- VM entries clear the dirty bits for the current VMCS as part of consistency checking (see below).
- An execution of VMWRITE sets the dirty bit for the current VMCS that corresponds to the VMCS field that was written.

Westmere uses these dirty bits to generalize Nehalem’s VM-entry optimizations. Specifically, a VM entry can skip any consistency checks pertaining entirely to VMCS fields for which the corresponding dirty bits are all clear. Because software can modify a cached VMCS only with VMWRITE, a VMCS field whose corresponding dirty bit is clear has not been modified since the last VM exit. (Unlike on Nehalem, Westmere does not use the



dirty bits to decide whether to load guest state from the VMCS region. On Westmere, each VM entry loads guest state only from the VMCS cache and never from the VMCS region.)

While it might seem ideal for each VMCS field to have its own dirty bit, this was not practical for Westmere. Instead, dirty bits were assigned to sets of VMCS fields based on an analysis of the usage patterns of existing VMMs. The microcode implementation of VM entries was then organized to exploit this assignment of dirty bits. Specifically, VM entries are implemented to group the consistency checks associated with each dirty bit. If a dirty bit is clear (indicating that the corresponding set of VMCS fields has not been modified), the microcode responsible for checking the consistency of the associated fields can be skipped. If, however, the dirty bit is set, the associated consistency checks are completed and, if the checks are successful, the dirty bit is cleared. Together, the assignment of dirty bits and the organization of VM-entry consistency checks reduce the number of dirty bits and provide much of the benefit of an ideal implementation.

Providing a VMCS cache for the current VMCS offers a substantial performance improvement. If the VMCS cache were available only to the current VMCS, every execution of VMPTRLD would load a new VMCS into the cache. (The previous VMCS, which is still active, would be copied to its VMCS region in system memory.) As noted earlier, this would set all the dirty bits for the new VMCS—even if that VMCS had been used earlier and were still active. As a result, the first VM entry after each execution of VMPTRLD would perform all consistency checks.

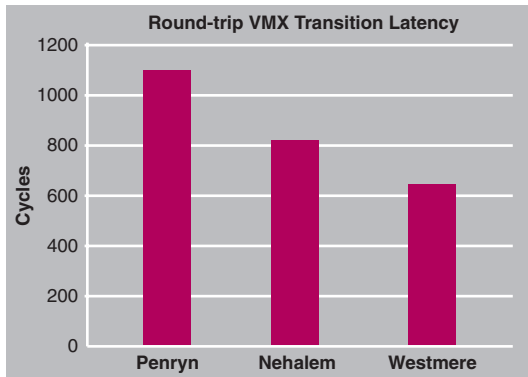
Some VMM usages require relatively frequent switches between VMs (by using VMPTRLD). As suggested in the previous paragraph, the ability to cache a single VMCS would not provide all the potential performance benefits of VMCS caching. (In the extreme case in which software alternates between two VMs—with a VMPTRLD between every pair of VM entries—every VM entry will be forced to perform all consistency checks; in this case the benefits of VMCS caching would be lost entirely.)

To support usages that switch between multiple VMs, Westmere's VMCS cache was built to accommodate three VMCSs concurrently. A VMM can thus switch between three VMs, and the processor will track the dirty status of fields in each of the corresponding VMCSs.

The Westmere VMCS cache allows significant reductions to the latency of VM entries. An intelligent assignment of dirty bits allows these reductions to be retained even when VMM software modifies the VMCS following a VM exit. Caching multiple VMCSs ensures that the reductions apply even with VMM software switches between multiple guests that use different VMCSs.

*“While it might seem ideal for each VMCS field to have its own dirty bit, this was not practical for Westmere. Instead, dirty bits were assigned to sets of VMCS fields based on an analysis of the usage patterns of existing VMMs.”*

*“To support usages that switch between multiple VMs, Westmere's VMCS cache was built to accommodate three VMCSs concurrently.”*



**Figure 4:** Comparison of Round-trip Transition Latencies on different processor generations  
(Source: Intel Corporation)

*“Some performance benefits, however, can be realized only with architectural extensions that require software changes.”*

*“Extended page tables (EPT) add a layer of address translation that a VMM can use to protect the physical address space.”*

*“Across a wide-range of workloads, virtualization of the paging hardware is typically the greatest source of overhead.”*

### Reductions to VMX-Transition Latencies: Summary

We have now described the microarchitectural enhancements to the Nehalem and Westmere processors that improve the latency of VMX transitions. These benefits are summarized in cycles in Figure 4. The best-case, round-trip latency (VM exit plus VM entry) on Nehalem is only about 75% (820 clock cycles) of that seen on the earlier Intel Penryn family of processors (1100 cycles). VMCS caching on Westmere reduces that latency to about 650 clocks—which translates to 60% of the latency on the Penryn family of processors.

### Contributions to Performance: Architecture

The architecture of Intel VT-x allows implementations to achieve performance benefits that allow existing software to run faster. Some performance benefits, however, can be realized only with architectural extensions that require software changes.

In this section we discuss four kinds of architectural extensions to Intel VT-x that improve performance:

- Extensions that reduce the frequency of VM exits.
- Extensions that allow direct execution of guest software in operating modes that might otherwise require software emulation.
- Extensions that support more efficient use of processor resources.
- Extensions that support more efficient operation of VMM software.

#### Architectural Extension: Extended Page Tables

Extended page tables (EPT) add a layer of address translation that a VMM can use to protect the physical address space. We now explain how the microarchitectural implementation of EPT solves potential performance challenges.

##### Motivation for EPT

To identify potential architectural extensions to Intel VT-x, we profiled the operation of some commercial, open-source and internally developed VMMs that use Intel VT-x. We found that, across a wide-range of workloads, virtualization of the paging hardware is typically the greatest source of overhead.

A VMM must control the physical address space to protect itself and to provide isolation between VMs. Prior to Nehalem, a VMM could do this only by using the existing paging hardware. Therefore, a VMM had to virtualize guest use of the paging hardware. A VMM could accomplish this by analyzing the paging structures created by guest software (*guest-paging structures*) and generating a corresponding set of paging structures in memory, often called *shadow-paging structures*. The shadow-paging structures may differ from the guest-paging structures, because the VMM can use different physical page

frames than those expected by guest software, and because the VMM sets page permissions differently: for example, a VMM may mark “not present” a page corresponding to a memory-mapped I/O device that it is emulating.

Various algorithms exist for managing shadow-paging structures. A VMM may map the guest-paging structures with read-only translations so that the VMM can intercept all guest paging-structure modifications, updating the shadow-paging structures with every such modification. Alternatively, it can update the shadow-paging structures in a lazy manner, emulating the behavior of a hardware TLB. This approach eliminates VM exits on writes to the guest-paging structures, but it induces a large number of page faults that must be intercepted by the VMM. The VMM must also intercept all instructions that change paging controls and control TLB contents (such as MOV CR3). Either approach induces frequent VM exits.

#### Architecture of EPT

EPT was architected to allow VMMs to avoid the frequent VM exits required by shadow-paging-structure algorithms. When EPT is in use, addresses that would otherwise be treated as physical addresses (and used to access memory) are instead treated as guest-physical addresses.

Guest-physical addresses are translated in a manner similar to the translation of linear addresses. The bits of a linear address are partitioned to select entries from a hierarchy of paging structures (for example, bits 47:39 select an entry from a PML4 table; bits 38:30 select an entry from a PDP table, etc.). Correspondingly, the bits of a guest-physical address are partitioned to select entries from a hierarchy of EPT paging structures; these entries contain physical addresses that are used to access memory.

To support EPT, the VMCS includes a new field called the EPT pointer (EPTP). The EPTP contains the physical address of the root of the EPT paging structures, the EPT PML4 table. EPT-based translation of a guest-physical address proceeds in much the same way as that of the ordinary translation of a linear address, beginning with the EPT PML4 table and followed by accesses to the EPT page-directory-pointer tables, EPT page directories, and EPT page tables.

A VMM that uses EPT need not intercept guest-paging-structure modifications, modifications to paging controls in registers CR3 or CR4, or executions of the TLB-management instruction INVLPG. Elimination of these VM exits and their handling (shadow-paging-structure maintenance) significantly reduces virtualization overhead. There may be savings in the VMM's memory footprint as well, since a VMM might otherwise maintain multiple sets of shadow-paging structures for each VM—while only a single set of EPT paging structures is required.

*“It can update the shadow-paging structures in a lazy manner, emulating the behavior of a hardware TLB.*

*This approach eliminates VM exits on writes to the guest-paging structures, but it induces a large number of page faults that must be intercepted by the VMM.”*

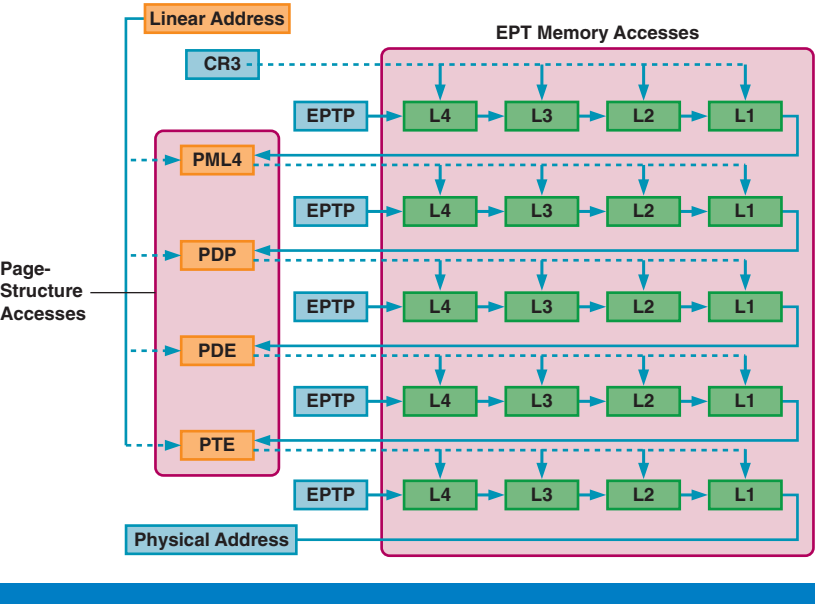
*“When EPT is in use, addresses that would otherwise be treated as physical addresses (and used to access memory) are instead treated as guest-physical addresses.”*

*“EPT-based translation of a guest-physical address proceeds in much the same way as that of the ordinary translation of a linear address.”*

*“While EPT eliminates the overhead of maintaining shadow-paging structures, the additional translation stage increases the latency of address translation during guest execution.”*

**EPT: Microarchitecture**

While EPT eliminates the overhead of maintaining shadow-paging structures, the additional translation stage increases the latency of address translation during guest execution. Figure 5 compares the work normally required to create a TLB entry (in IA-32e mode) with that required when EPT is active.



**Figure 5:** Page walks with and without EPT  
(Source: Intel Corporation)

Without EPT, creating a TLB entry requires accessing four paging-structure entries (the four yellow boxes marked PML4, PDP, PDE, and PTE in Figure 5). With EPT, each of these four accesses is preceded by accesses to four EPT paging-structure entries to identify the physical address of an ordinary paging-structure entry (the four green boxes in the row above each yellow box). After identifying the ultimate guest-physical address in the PTE, four additional EPT paging-structure entries are accessed to determine the ultimate physical address (the four green boxes in the last row).

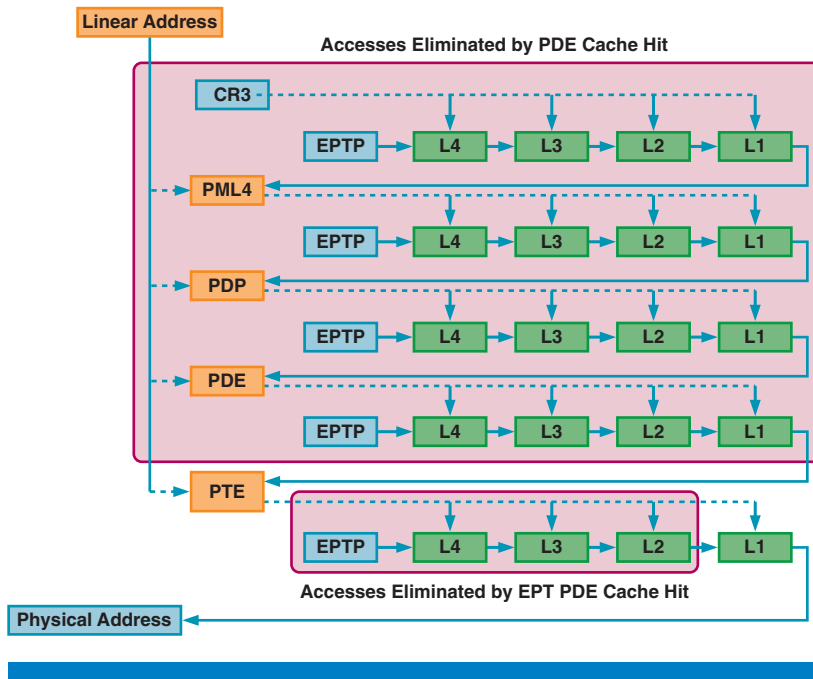
As can be seen, the number of memory accesses required to create a TLB entry may increase from 4 to 24. Given this increase, a naïve implementation of EPT would result in a loss of performance for many workloads.

Even in the absence of EPT, Nehalem provides paging-structure caches to reduce the cost of a TLB fill. For example, Nehalem uses a cache of page-directory entries (PDE cache). Each valid entry in the processor’s PDE cache associates a 2MB region of the linear address space with the physical address of the page table responsible for that region. Only a single memory access (to PTE) is required to create a TLB entry for a linear address for which there is a valid entry in the PDE cache. If EPT is in use, four additional memory accesses (for a total of five) are required to create a TLB entry.

*“A naïve implementation of EPT would result in a loss of performance for many workloads.”*

Nehalem further reduces the overhead of EPT with EPT paging-structure caches. Each valid entry in the processor's EPT PDE cache associates a 2MB region of the guest-physical-address space with the physical address of the EPT page table responsible for that region. If there is a valid PDE-cache entry for a linear address, and there is a valid EPT PDE-cache entry for the resulting guest-physical address, only two memory accesses (one to a PTE, one to an EPT PTE) are required to create a TLB entry. The overhead of EPT is reduced to a single additional memory reference for every TLB fill.

Figure 6 illustrates the benefits of both the ordinary PDE cache and the EPT PDE cache. Contrast this with Figure 5. Because an entry in the ordinary PDE cache contains the physical address of a PTE, it eliminates the need for 19 memory accesses (the dimmed accesses in the upper part of the figure). Because an entry in the EPT PDE cache contains the physical address of an EPT PTE, it eliminates the need for 3 memory accesses (the dimmed accesses in the lower part of the figure).



**Figure 6:** Benefits of paging-structure caches  
(Source: Intel Corporation)

In addition to the previously described PDE and EPT PDE caches, Nehalem also has a PDPTE cache and an EPT PDPTE cache. Elements in the PDPTE and EPT PDPTE caches correspond to 1-GBYTE regions of the linear and guest-physical address spaces, respectively.

The sizes of Nehalem's paging-structure caches are such that they contain valid entries for virtually all such references (this is because references to linear addresses exhibit high degrees of locality). While streams of references to guest-physical addresses exhibit less locality, the EPT paging-structure caches

*“If there is a valid PDE-cache entry for a linear address, and there is a valid EPT PDE-cache entry for the resulting guest-physical address, only two memory accesses (one to a PTE, one to an EPT PTE) are required to create a TLB entry.”*

*“The sizes of Nehalem's paging-structure caches are such that they contain valid entries for virtually all such references”*

*“In practice, system BIOS and operating-system loaders operate in real-address mode, while operating systems and their applications operate in protected mode with paging.”*

*“Virtual-8086 mode imposes restrictions on segment-register state that real-address mode does not (for example, segment limits are fixed at 64 KBytes), and certain instructions are not supported in virtual-8086 mode.”*

still significantly reduce the number of memory accesses required to translate a guest-physical address.

### **Architectural Extension: Virtualizing Real Address Mode and Unpaged Protected Mode**

The Intel 64 and IA-32 architectures support several modes of operation defined by the processor's segmentation and paging hardware. Each logical processor comes out of RESET in real-address mode; this mode does not support protection based on either segmentation or paging. Software can subsequently transition the logical processor to protected mode, in which segment-based protections are available. Within protected mode, software can enable *paging*, with page-based protections. In practice, system BIOS and operating-system loaders operate in real-address mode, while operating systems and their applications operate in protected mode with paging.

These modes are managed by two bits in control register CR0: protection enable (PE) and paging (PG). The processor is in real-address mode when  $CR0.PE = 0$ . The processor is in protected mode when  $CR0.PE = 1$ . Paging is enabled in  $CR0.PG = 1$ .

Processors supporting Intel VT-x prior to Westmere required that  $CR0.PE$  and  $CR0.PG$  both be 1 in VMX non-root operation. On these processors, guest software can operate only in paged protected mode; it cannot operate in either real-address mode or in unpaged protected mode.

VMMs can approximate a virtualization of real-address mode by using virtual-8086 mode. However, virtual-8086 mode imposes restrictions on segment-register state that real-address mode does not (for example, segment limits are fixed at 64 KBytes), and certain instructions are not supported in virtual-8086 mode.

Complete virtualization of real-address mode requires time-consuming instruction emulation. Virtualization of unpaged protected mode could be accomplished by using a variety of techniques, each of which requires either significant software complexity or a large memory footprint for additional page tables. Because of these limitations, some VMMs had significant limits on their ability to virtualize modules that use real-address mode or unpaged protected mode (for example, OS loaders).

Westmere alleviates these problems with a new feature called unrestricted guest. When this feature is enabled, a logical processor relaxes the restrictions on  $CR0.PE$  and  $CR0.PG$ , allowing a VM entry to either real-address mode or unpaged protected mode.<sup>1</sup> Enabling the unrestricted-guest feature also relaxes certain VM-entry restrictions, allowing segment registers to contain all the states that are possible during transitions between the various modes discussed earlier.

<sup>1</sup> Because there are no paging protections, if  $CR0.PE$  or  $CR0.PG$  is 0, the unrestricted-guest feature can be used only when EPT is in use.



The unrestricted-guest feature allows VMMs to virtualize software operating in real-address and unpaged protected modes and to achieve essentially the same performance as software operating in paged protected mode.

### Architectural Extension: Virtual Processor Identifiers (VPIDs)

The term linear-address space refers to the translation of linear (virtual) addresses to physical addresses as controlled by paging (and, optionally, EPT). Operating-system software typically changes from one linear-address space to another when it changes applications. When a VMM is involved, the linear-address changes when VMX transitions between guest and host (VMM) software.

Processors accelerate linear-address translation by caching translations in TLBs. Before the introduction of the Nehalem processor, Intel 64 and IA-32 processors cached translations only for the current linear-address space. For that reason, changes to linear addresses invalidate all translations in the TLBs.

TLB invalidations ensure correct software operation, but they reduce the benefits of TLBs. In particular, the TLB invalidations caused by VMX transitions may adversely affect the performance of software operating in a VM. This is especially true when EPT is in use, as EPT increases the cost of creating a TLB entry (discussed previously).

Nehalem addresses this performance limitation by allowing its TLBs to cache translations for multiple linear-address spaces concurrently. It achieves this through a new processor feature called the virtual-processor identifier (VPID).

A logical processor maintains a current VPID, which it uses as follows:

- Translations inserted into the TLBs are associated with the current VPID.
- Only translations associated with the current VPID are used to accelerate paging.
- Operations that flush or invalidate TLB entries need do so only for translations associated with the current VPID.

Before the introduction of VPIDs on Nehalem, every VMX transition invalidated all TLB entries. If VPIDs are enabled, VMX transitions simply change the current VPID without invalidating any TLB entries.

(In practice, there may be a limit to the number of VPIDs for which the TLBs may cache entries concurrently. Because of this, a VMX transition may invalidate TLB entries if it is establishing a VPID for which the TLBs are not currently caching.)

A VMM can assign a different VPID to each VM, thus avoiding the TLB invalidations that occurred on every VMX transition on processors prior to Nehalem. In this way, VPIDs reduce or eliminate the performance penalty that used to be incurred by guest software on earlier generations of processors.

*“The unrestricted-guest feature allows VMMs to virtualize software operating in real-address and unpaged protected modes and to achieve essentially the same performance as software operating in paged protected mode.”*

*“Before the introduction of VPIDs on Nehalem, every VMX transition invalidated all TLB entries.”*



*“Enabling VPID and EPT increased the vConsolidate performance of Nehalem by 25%.”*

*“The Nehalem and Westmere processors support architectural extensions to Intel VT-x that improve a VMM’s ability to perform such scheduling.”*

*“When the VMX-preemption timer is enabled, a VM entry loads the timer from a VMX-preemption timer value in the VMCS. As long as the processor is in VMX non-root operation, the timer counts; when the timer value reaches 0, a VM exit occurs.”*

The performance benefits of EPT and VPID support are highly workload dependent. We used the vConsolidate<sup>2</sup> benchmark to assess these benefits for consolidation workloads. Enabling VPID and EPT increased the vConsolidate performance of Nehalem by 25%.

### **Architectural Extensions: Support for VMM Scheduling**

The performance of a virtualized platform depends on the ability of the VMM to schedule VMs effectively. The Nehalem and Westmere processors support architectural extensions to Intel VT-x that improve a VMM’s ability to perform such scheduling. We now look at two such extensions.

#### **VMX-Preemption Timer**

For a VMM to time-multiplex a logical processor between multiple VMs, it must have a mechanism to limit guest execution time. That is, the VMM must be able to allow guest software to run, knowing that some hardware mechanism will cause a VM exit in an amount of time under the control of the VMM.

The requisite hardware mechanism could be one of the various platform timer sources. Because there may be disparity between the guest software running in the different VMs, the use of a timer source local to a logical processor is preferred over the use of an external platform source shared by logical processors.

There is a local-APIC timer for each logical processor. If, however, a VMM uses the local-APIC timer to limit guest execution, it cannot allow guest software to program that timer directly. If guest software itself wants to use the local-APIC timer, the VMM may require software to support the sharing of the timer between guest and host and to disambiguate the proper recipient of each timer interrupt.

To spare VMMs the complexity of timer sharing (for guests using the local-APIC timer), Nehalem introduced the VMX-preemption timer. This is a new timer that is controlled by the current VMCS. Because each logical processor has its own VMCS, a VMM can configure the timer independently on each logical processor.

When the VMX-preemption timer is enabled, a VM entry loads the timer from a VMX-preemption timer value in the VMCS. As long as the processor is in VMX non-root operation, the timer counts; when the timer value reaches 0, a VM exit occurs.

In addition, software can specify that every VM exit (not only VM exits due to timer expiry) save the current timer value (which was decremented during guest execution) into the VMCS; the next VM entry loads the decremented

<sup>2</sup> vConsolidate is a virtualization benchmark in which database, Java\*, web, mail, and idle servers are added, in sets, to a platform until maximum throughput is achieved.

value into the timer. Otherwise, the timer value in the VMCS is left unchanged; every VM entry loads the same value into the timer. This flexibility allows software a variety of scheduling algorithms.

### PAUSE-Loop Exiting

VMMs supporting VMs with multiple virtual processors face additional challenges. A VMM could choose to run all the virtual processors of a guest concurrently. This approach, called gang scheduling, can be inefficient (for example, if not all the virtual processors of a VM have work to do) and result in performance loss.

A VMM may instead choose, at any given time, to run some virtual processors of a VM while not running others. This approach, while more flexible than gang scheduling, may lead to problems of lock-holder preemption. Suppose, for example, that the VMM has scheduled one virtual processor (VP1) but not another (VP2). The operation of the VM may be such that VP2 is holding a software lock, while VP1 is trying to acquire that lock. Because the VMM has not scheduled VP2, it cannot release the lock; VP1 may spend its entire scheduled time quantum waiting for the lock. Lock-holder preemption may thus result in inefficiencies and performance loss when gang scheduling is not used.

Server processors based on the Nehalem core introduce PAUSE-loop exiting, which can eliminate the inefficiencies of lock-holder preemption. The feature is based on the fact that software waiting for a lock typically enters a loop that includes an execution of the PAUSE instruction in each iteration. In the absence of lock-holder preemption, such loops are typically short-lived. A VMM can thus detect lock-holder preemption by identifying loops that use the PAUSE instruction (“PAUSE loops”) that persist for too long.

A VMM controls PAUSE-loop exiting with two new fields in the VMCS: PLE\_Gap and PLE\_Window. When the feature is enabled, the logical processor notes the time (as measured by the timestamp counter) of every execution of PAUSE with CPL = 0. If the time between two successive executions does not exceed PLE\_Gap, they are considered to be part of the same PAUSE loop. If the time of an execution in a PAUSE loop exceeds that of the first execution by more than PLE\_Window, a VM exit occurs with an exit reason indicating “PAUSE.”

A VMM can use the VM exits generated by PAUSE-loop exiting to identify occurrences of lock-holder preemption and reschedule virtual processors according to an algorithm of its choosing. A VMM can configure the fields PLE\_Gap and PLE\_Window based on its own scheduling requirements, properties of the guest software being scheduled, or a combination of the two.

To assess the benefit of PAUSE-loop exiting, a Xen system was over-committed with 12 Web-bench VMs and 4 SPECjbb VMs. Enabling PAUSE-loop exiting increased Webbench transactions per second by up to 40%.

*“Server processors based on the Nehalem core introduce PAUSE-loop exiting, which can eliminate the inefficiencies of lock-holder preemption.”*

*“Some features of Intel VT-x serve to separate the functionality of VMM and guest software, allowing each to operate independently and reducing the frequency of transitions between the two.”*

*“If a VMM can monitor certain aspects of guest-software execution, it may be able to enhance the security or robustness of guest software.”*

*“A VMM can respond one of these VM exits either by blocking the exiting instruction (if it is deemed malicious) or by changing the addresses protected by EPT (if the instruction is deemed legitimate).”*

## Architectural Support for New Usages

We previously described extensions to Intel VT-x that were added to the Nehalem and Westmere processors to improve performance. These processors also support extensions that enable new VMM functionality.

Some features of Intel VT-x serve to separate the functionality of VMM and guest software, allowing each to operate independently and reducing the frequency of transitions between the two. EPT is an example. Without EPT, VMM and guest software both seek to use the same paging hardware to protect memory, and the VMM requires frequent VM exits to support this sharing of hardware. With EPT, VMM and guest software can each maintain its own paging structures independently.

Despite this, some VMM developers have identified new usages that require closer interactions between VMM and guest software. If a VMM can monitor certain aspects of guest-software execution, it may be able to enhance the security or robustness of guest software. In this section we describe two new features of Intel VT-x, introduced in Nehalem, that support such new usages.

### VMM-Based Security: Descriptor-Table Exiting

The global descriptor table (GDT), local descriptor table (LDT), interrupt-descriptor table (IDT), and task-state segment (TSS) are data structures that are central to the operation of an x86-based operating-system kernel. A VMM can enhance the security of guest operating systems (those running in the VMM’s VMs) by protecting the integrity of these data structures. For example, it can use EPT to ensure that the pages containing these data structures are read-only. The processor locates these data structures by using addresses in the GDTR, LDTR, IDTR, and TR registers. Software can write to these registers by using the LGDT, LLDT, LIDT, and LTR instructions.

If malicious software used these instructions to change the location of the data structures, the EPT-based protections we described in the previous paragraph would be ineffective (the VMM would be protecting the wrong pages).

To prevent malicious software from compromising this security usage, Nehalem introduced a new feature called descriptor-table exiting. When this feature is enabled, executions of LGDT, LLDT, LIDT, and LTR cause VM exits. A VMM can respond to one of these VM exits either by blocking the exiting instruction (if it is deemed malicious) or by changing the addresses protected by EPT (if the instruction is deemed legitimate).

### VMM-Based Debugging: Monitor Trap Flag

VMM developers may want to provide support to debug guest software. A VMM providing such debugging support might use the debugging features provided by the processor (breakpoints, single stepping, etc.) Most of these debugging features are controlled by processor resources that can be controlled by Intel VT-x (debug registers and model-specific registers).

One debugging feature that is not controlled by Intel VT-x is the ability to do single-step execution. This feature is enabled by the setting of the trap flag (TF) in the RFLAGS register: the processor generates a debug exception after any instruction that commences with  $RFLAGS.TF = 1$ . Unlike the debug registers and model-specific registers, the processor provides no protections for RFLAGS.TF. It can be modified by software operating at any privilege level, and Intel VT-x does not include a feature to prevent modification of RFLAGS.TF by guest software. A VMM that uses single-stepping to debug guest software could be confounded if guest software executes an instruction that clears RFLAGS.TF.

To support the debugging of guest software, Nehalem introduced a new feature called the monitor trap flag. If this flag is set, a VM exit occurs after the first instruction executed in VMX non-root operation. This functionality is independent of the value of RFLAGS.TF.

## Conclusion

Intel VT-x, as first implemented in some generations of Intel Pentium® 4 processors, comprises extensions to the instruction-set architecture that provide processor-level support for VMs. Because it simplifies significantly the task of virtualizing IA-32 and Intel 64 processors, VT-x has been adopted for use by all major vendors of virtualization software for those processors.

As its use has grown—in both the number of systems on which it is being used and the ways in which it is being used—the demand for improved performance in virtualized systems has increased. For this reason, the Nehalem and Westmere processors include changes to the microarchitecture and Intel VT-x architecture that improve the performance of software that uses Intel VT-x. In this article we describe those changes and how they have significant positive effects on performance of virtualized systems that use the Intel VT-x architecture.

## Acknowledgements

We thank the following for their contributions to the development of Intel VT-x on the Nehalem and Westmere processors: Steve Bennett, Rob Chappell, Travis Furrer, David Koufaty, Sean Mirkes, Iredamola Olopade, Madhavan Parthasarathy, Camron Rust, Rajesh Sankaran, Andrea Tan, Richard Uhlig, and Rupin Vakharwala.

## References

- [1] “Intel Virtualization Technology.” *IEEE Computer Magazine*, May 2005, pp. 48–56.

## Authors’ Biographies

**Bill Alexander** is a Senior Processor Design Engineer in the Intel Architecture Group and has in the past worked on microcode and architecture that implement Intel VT-x. He received a B.S. degree in Electrical and Computer Engineering from Carnegie Mellon University and an M.S. degree in Computer Engineering from the University of Illinois at Urbana-Champaign.

**Andy Anderson** is a Research Scientist in Intel Labs working on the architecture of Intel VT-x and on performance analysis of virtualized systems. He received B.S. and M.S. degrees in Electrical and Computer Engineering from Brigham Young University.

**Barry Huntley** is a Senior Processor Design Engineer in the Intel Architecture Group working on microcode and architecture that implement Intel VT-x. He received a B.S. degree in Electrical Engineering with a minor in Computer Science from the University of Illinois at Urbana-Champaign.

**Gil Neiger** is a Senior Principal Engineer in Intel Labs and leads development of the architecture of Intel VT-x. He received a Ph.D. in Computer Science from Cornell University.

**Dion Rodgers** is a Senior Principal Engineer in the Intel Architecture Group and is responsible for bringing multiple advanced technology initiatives such as Intel VT-x to the product line of Intel 64 and IA-32 processors. He received an M.S. degree in Computer Engineering from Clemson University.

**Larry Smith** is a Senior Staff Processor Design Engineer in the Intel Architecture Group working on microcode and architecture that implement Intel VT-x. He received a B.S. degree in Industrial Management with a minor in Finance from the Lawrence Institute of Technology.

## Copyright

Copyright © 2011 Intel Corporation. All rights reserved.

Intel, the Intel logo, and Intel Atom are trademarks of Intel Corporation in the U.S. and other countries.

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

Requires an Intel® HT Technology enabled system, check with your PC manufacturer. Performance will vary depending on the specific hardware and software used. Not available on Intel® Core™ i5-750. For more information including details on which processors support HT Technology, visit <http://www.intel.com/info/hyperthreading>

Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM). Functionality, performance or other benefits will vary depending on hardware and software configurations. Software applications may not be compatible with all operating systems. Consult your PC manufacturer. For more information, visit <http://www.intel.com/go/virtualization>

Requires a system with Intel® Turbo Boost Technology capability. Consult your PC manufacturer. Performance varies depending on hardware, software and system configuration. For more information, visit <http://www.intel.com/technology/turboboost>

# CIRCUIT AND PROCESS INNOVATIONS TO ENABLE HIGH-PERFORMANCE, AND POWER AND AREA EFFICIENCY ON THE NEHALEM AND WESTMERE FAMILY OF INTEL PROCESSORS

## Contributors

**Kumar Anshumali**  
Intel Corporation

**Terry Chappell**  
Intel Corporation

**Wilfred Gomes**  
Intel Corporation

**Jeff Miller**  
Intel Corporation

**Nasser Kurd**  
Intel Corporation

**Rajesh Kumar**  
Intel Corporation

## Index Words

Clocking  
Voltage Scaling  
Small Signal Arrays  
Leakage

Achieving high performance, energy efficiency, and scalability across a large power envelope were the high-level goals of the designers of the *Intel® microarchitecture code name Nehalem and Intel® microarchitecture code name Westmere processors*. This article describes the key innovations in circuit technology that were implemented along with the co-optimization of process and circuits to achieve our goals.

### Introduction

A clocking- and power-plane-partitioning scheme with low-latency, synchronous communication enabled us to use modular building blocks; thus, helping with scalability. By using numerous circuit and process technology changes and by converting the high-power domino logic family to static CMOS logic, we reduced chip operating voltage; thus, achieving energy efficiency. Innovations in data and instruction caches and multi-ported Register File (RF) design helped us realize the highest amount of performance with the lowest cost. In this article, we describe the related process and circuit innovations in the building of the memory controller and the cache coherent link for high-performance, scalable systems.

### Guiding Principles of the Nehalem Processor: Energy and Area Efficiency for High Performance

The large dynamic performance range of the Intel® family of processors, codename Nehalem, was reached by innovations in process technology, circuit technology, and micro-architecture. We achieved a low-latency, balanced pipeline by using energy-efficient static circuits, instead of domino circuits, for datapath and control blocks. These features were coupled with innovations in RF and cache design. We also used multi-ported RFs that allowed us to design a wide, super-scalar machine that supported multi-threading with minimal area growth. The design and methodology were simplified by using a high-density unified Standard Cell Library that could be used for control, datapath, and RFs. The cores were coupled with a clocking and power architecture that enabled system-level power optimization. Higher performance cores require more bandwidth. This was accomplished by integrating the on-die memory controller and the input output (IO) circuits for Double Data Rate (DDR)



memory. These techniques were extended to the Quick Path Interconnect (QPI), which allows for a large, scalable system.

## Low-Power Design: Voltage Scaling and Clocking

Our goal was to achieve low-voltage operation and a large dynamic range for performance to support multiple laptop and desktop Personal Computer (PC) segments. The low-voltage operations were achieved by designing the Nehalem and Westmere processors to leverage independent supply domains for core (the technology used inside each core) and Uncore (I/O and last-level caches) sections to optimize overall performance and die area. The Uncore targets maximum logic density at a fixed voltage by the use of sleep circuits (local supply gating) to achieve power savings. Conversely, cores target performance-per-watt energy efficiency through Vmin (low voltage) circuit technologies to achieve operation at supply voltages below 750mV.

### Nehalem Clocking

Greater integration of core and cache on a single die to improve performance delivered per watt is leading to increased clock challenges. The die is becoming larger, making it prohibitive to design a single, low-jitter clock domain. In addition, to efficiently manage power consumption, the die is divided into different voltage and frequency domains, each with a dedicated local Phase Lock Loop (PLL). Multiple PLLs lead to multiple clock domains. Transferring data across different clock domains requires that skew and frequency mismatch be accounted for, and it typically requires the use of First-In-First-Out (FIFO) buffers to prevent data corruption. The data in FIFO buffers are written by one clock domain and held until they can be reliably read by a second clock domain. This adds latency in the data transfer. A major component of the FIFO depth budget is used to absorb random and deterministic clock jitter and skew between the interfacing clocks. Thus, it is more critical than ever to architect low-skew, low-jitter clock solutions [1].

Figure 1 illustrates the Nehalem and Westmere processor's high-level clocking architecture. An external 133MHz clock is multiplied by scale factors of 1, 2 and 4X by a filter PLL and routed to the dedicated local PLLs. Higher-reference clock frequencies improve performance with fast-frequency transitions enabled by reduced PLL re-lock times. Further, higher sampling frequencies minimize FIFO latencies by reducing long-term jitter between the different PLL domains. Each PLL implements duty cycle correctors to mitigate Fmax degradations due to duty cycle variation and aging [2].

Changing activity level in the processor core creates  $di/dt$ , resulting in supply voltage droops. In traditional designs, the processor core frequency remains fixed irrespective of the transient voltage levels in the core. Hence, the circuit

*“The Uncore targets maximum logic density at a fixed voltage by the use of sleep circuits (local supply gating) to achieve power savings.”*

*“The die is becoming larger, making it prohibitive to design a single, low-jitter clock domain.”*

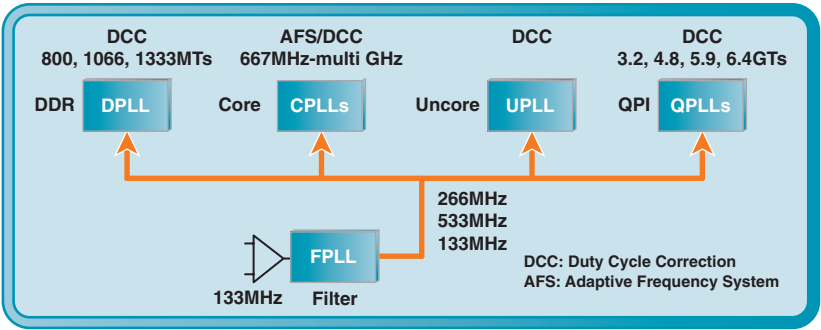


Figure 1: High-level clock architecture

*“Often, significant performance upside is left unclaimed on the table because of the voltage droop.”*

must be guaranteed to be functional at the lowest transient voltage value at the specified target frequency. To reduce the frequency impact from the droop, we used large, on-die decoupling capacitors and expensive low-impedance package capacitors in addition to speeding up the critical paths to meet the target frequency at the lowest droop voltage. Large, on-die decoupling capacitors are increasingly becoming less feasible for every successive generation because of exponentially increasing leakage current. Also, speeding up critical paths to meet frequency at lower voltage levels results in larger devices, and hence it contributes to higher leakage and dynamic power. Often, significant performance upside is left unclaimed on the table because of the voltage droop. The solution is to adapt the core frequency in response to the core supply droops to obtain this performance upside. In the Nehalem and Westmere design, we implemented an Adaptive Frequency System (AFS). The adaptive PLL modulates the clock frequency based on the core supply first voltage droops, as shown in Figure 2.

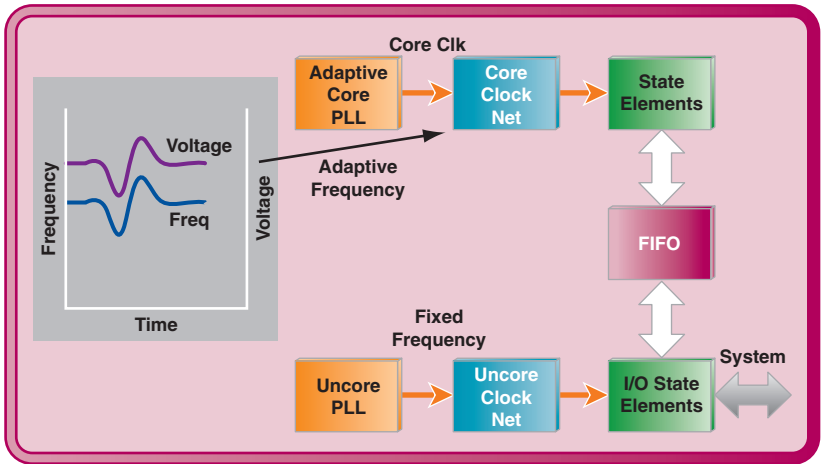
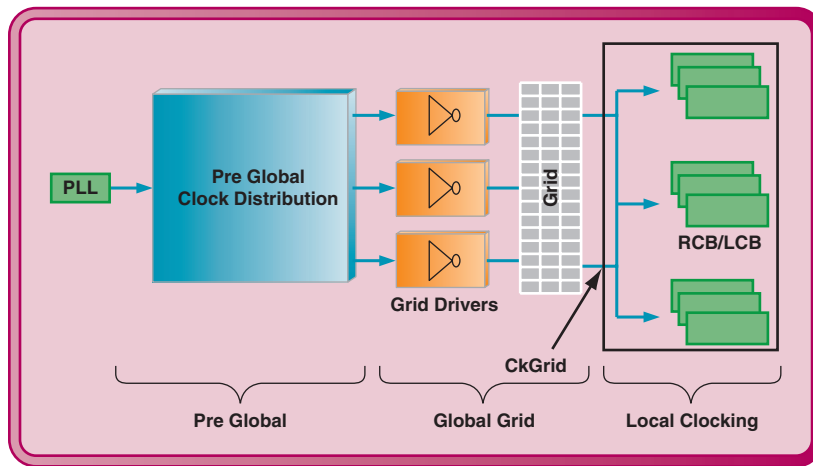


Figure 2: Adaptive Clocking System (AFS)

For the core to remain deterministic and synchronous, a FIFO buffer absorbs transient frequency changes but maintains a constant frequency data flow to the outside world [3].



**Figure 3:** Nehalem core clock distribution at three levels of hierarchy

Figure 3 illustrates the core clock distribution hierarchy that is constructed from three levels: pre-global clock spines that deliver matched clocks to the global grid drivers, global grid wires that deliver the clock to the entire core, and local clocking. Local clocking consists of regional and local clock buffers (RCB/LCB) with fine-grained clock gating. The global clock topology consists of one horizontal spine, several vertical spines, and M8 grid wires. Clock recombination is done from the center to the full height and width of the spine. The last-stage drivers of all vertical spines are shorted together via M8 grid wires. This hybrid routing style is carefully chosen to reduce overall skew and design complexity without using unnecessary energy [4].

## Voltage Scaling—Minimum to Maximum

The large voltage range of the Nehalem processor was achieved by circuit, process, and methodology innovations in RF and cache designs that were built for low-voltage operations. The design tradeoffs are listed in this section.

### Core V<sub>min</sub> and Energy-Efficient Computing

Nehalem and Westmere cores support Speedstep\* scalable performance through paired voltage and frequency operating points known as P-states. P-states step in 133MHz increments from low-frequency mode (LFM) P<sub>n</sub>, to high-frequency mode (HFM) P<sub>1</sub>, to turbo “highest” frequency modes (TFM) P<sub>0</sub>. Maximum efficiency is attained by simultaneously lowering voltage in

*“Nehalem and Westmere cores support Speedstep\* scalable performance through paired voltage and frequency operating points known as P-states.”*

*“The focus of Vmin circuit design is to identify circuits that impede low-voltage operation and then to improve them either by enhancing their design and/or dynamically repairing them during test or use.”*

*“Vmin circuit design is guided by statistical evaluations of individual circuits to meet aggregate yield loss goals based on total cell counts within a die.”*

conjunction with each step in order to lower frequency. A Vmin floor exists below which circuits will no longer function. At this Vmin floor, frequency can only be lowered at a fixed voltage to throttle performance. Consequently, dynamic power reduction only scales linearly relative to performance at the Vmin floor. Lowering the Vmin floor results in a cubic reduction in power with a linear reduction in performance, resulting in a large dynamic performance and power range.

### Vmin Circuit Design for Variation

The focus of Vmin circuit design is to identify circuits that impede low-voltage operation and then to improve them either by enhancing their design and/or dynamically repairing them during test or use. Fundamentally, Vmin operation below 750mV requires that circuits be chosen, designed, and tested for tolerance to transistor parametric variations that occur during manufacture and ones that may continue to shift throughout a processor’s lifetime. Primary device-variation parameters include gate threshold ( $V_t$ ) and gate length ( $L_e$ ). The magnitude of device variation is further dependent upon the physical size of a transistor; smaller devices, as measured by gate length and width, exhibit wider variation. Variations are measured as deviations from nominal parameter targets and are further classified as systematic and random. Systematic variation affects multiple devices collectively that are in close proximity to one another, whereas random variation uniquely affects an individual device. Device variation encountered in manufacturing is specifically referred to as built-in variation. Device variation also continues throughout the lifetime of an individual processor die as measured by changes in device parameters over time. This variation over time is referred to as aging. Aging variation also has systematic and random components driven by usage conditions that vary across the die, such as voltage, temperature, and device operating states.

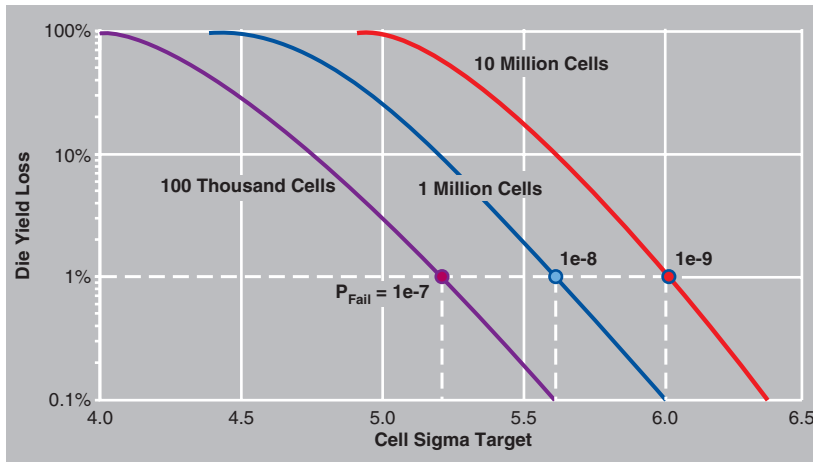
Vmin circuit design is guided by statistical evaluations of individual circuits to meet aggregate yield loss goals based on total cell counts within a die. Cells that are used in greater numbers must be designed for higher tolerance to variation. From the aggregate yield target, an acceptable cell failure probability ( $P_{Fail}$ ) is calculated as shown in [a].

$$P_{Fail} = 1 - (1 - YieldLoss)^{(1 \div cell.count)} \quad [a]$$

A cell variation design target, measured in sigma (standard deviation) is determined from a standard normal distribution as shown in [b].

$$P_{Fail} = f(sigma) = \int_{-\infty}^{sigma} \frac{1}{\sqrt{2\pi}} \exp(-x^2/2) \quad [b]$$

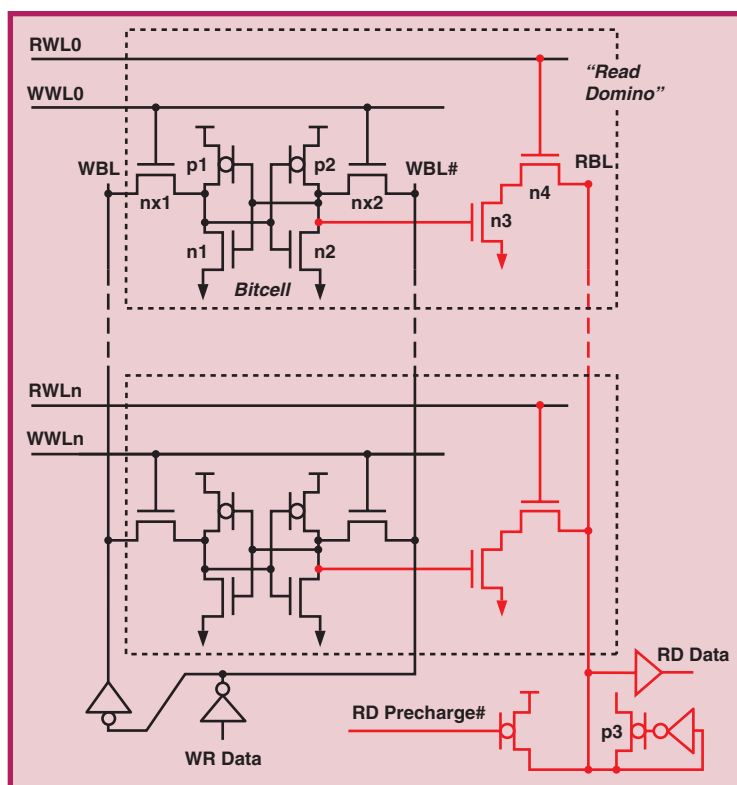
Figure 4 graphically depicts die-level yield loss versus cell sigma design target for various cell counts. The sigma target for a cell corresponds to a cell's failure probability  $P_{\text{FAIL}}$ . Consider a die-level yield loss goal of 1%. If any individual cell fails, then the die will be discarded as a yield loss. Thus, as the total cell count is increased, the cell failure probability must decrease in order to maintain the same aggregate 1% yield loss. The challenge of maintaining high die yields with increasing integration, therefore, becomes apparent, and thus motivates us to refine statistical design and yield recovery methods.



**Figure 4:** Die-level yield loss versus sigma design target

### Register Files (RF) Small Signal Array (SSA) and Active Vmin

The Nehalem and Westmere processor cores incorporate more than 90 unique RFs and small signal arrays (SSAs). RF and SSA storage elements may impede low-voltage operation due to their high cell counts and inherent sensitivity to device variation with multiple failure modes that impose competing constraints. Typical RF write and read circuitry are shown in Figure 5, with the read path highlighted in red. The cross-coupled inverters within the bitcell and the domino keeper are jam (or contention) topologies in which a stronger device must overcome a weaker device to drive a change of state. The weaker devices, namely p1, p2 within the bitcell and p3 within the domino are intended to retain state between operations. The devices nx1, nx2, n3, and n4 operate to change state by overpowering the weak devices.

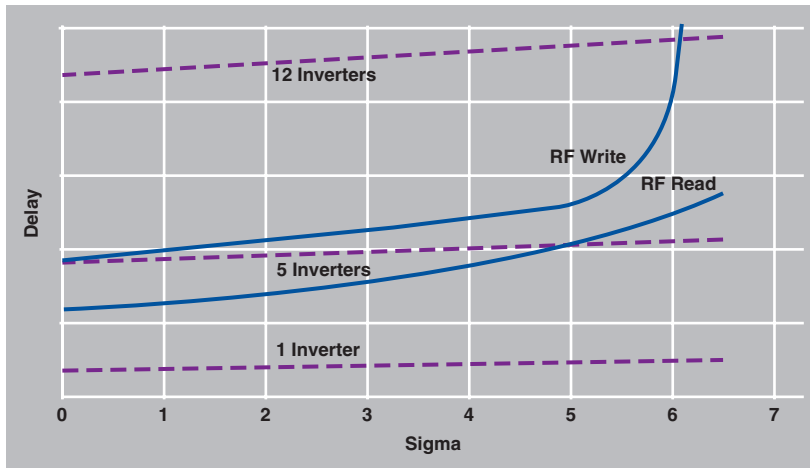


**Figure 5:** Register File Read Write circuits

A shift in NFET to PFET strengths results in a nonlinear degradation of jam performance as measured by a delay of writes and reads.

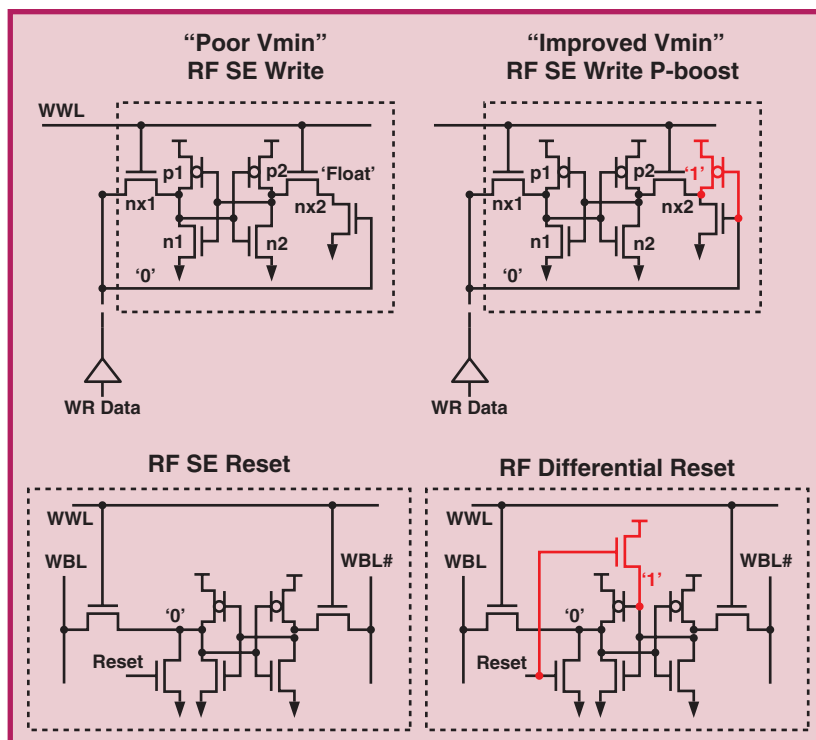
Figure 6 shows an RF case study at  $V_{min}$  as a sweep of delay versus  $\sigma$ . The RF write nominally matches the delay of 5 inverter stages at 0 $\sigma$  (or no variation). At 6 $\sigma$ , however, RF write delay expands, becoming nearly equivalent to 12 inverter stages under the same total variation. At higher operating voltages this effect is not observed. In the case of RF writes, a restriction on time borrowing can be imposed at  $V_{min}$  to accommodate extended write delay.

Alternative non-jam topologies, which insert additional devices to interrupt the feedback path of storage elements, are more tolerant of device variation, but are not favored due to their area and power costs. Thus, underlying objectives of area and power savings outweigh the desire to replace all  $V_{min}$ -sensitive circuits.



**Figure 6:** RF delay versus variation at  $V_{min}$

The  $V_{min}$  of common RF jam topologies can also be improved by adding devices to convert them from single ended (SE) to differential configurations. Differential jam topologies are more tolerant of individual device variations. Some examples are depicted in Figure 7 with added devices depicted in red. One example, an “RF SE write,” can be improved with one additional PFET to become a differential “SE Write with P-Boost.” A second example, an “RF SE reset,” can become a “Differential Reset” with the addition of one NFET.

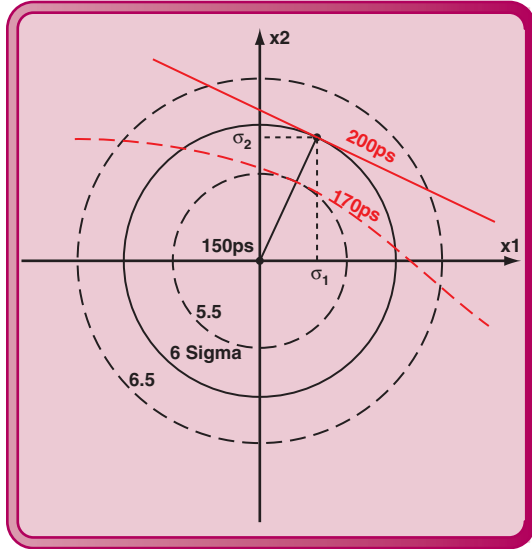


**Figure 7:** Alternative Register File topologies to improve  $V_{min}$



### Most Probable Point Algorithm

The Most Probable Point (MPP) simulation search algorithm is the preferred method to evaluate a circuit's robustness under high levels of variation—on the order of  $5-7\sigma$ . MPP iteratively runs circuit simulations while varying selected device parameters to seek the most likely combination of variation assignments that cause failure of a specified measurement criterion. Delay is a typical measurement criterion [5].



**Figure 8:** MPP search space for circuit robustness

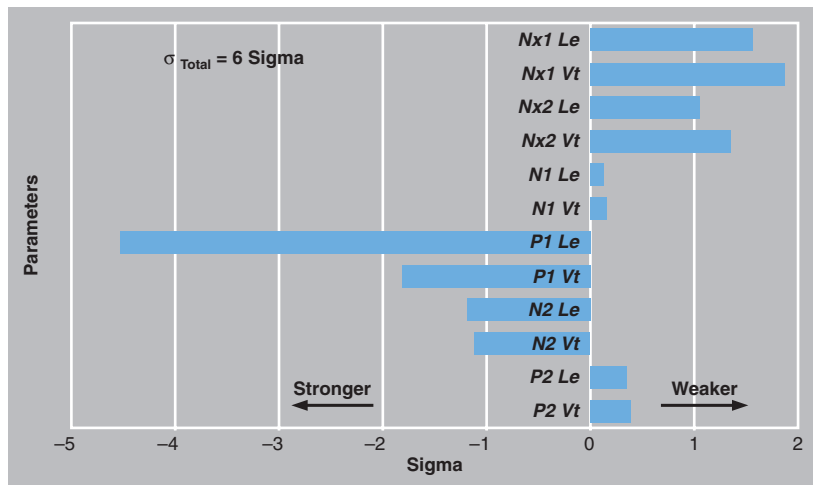
Monte Carlo simulation is an alternative to MPP that can also predict the response of circuits under variation, but for reasonable run times, it only provides results out to a total of  $3-4\sigma$  [5]. The difference between Monte Carlo and MPP is that Monte Carlo runs a random selection of device parameter variations to predict a response to total variation, whereas MPP evaluates intermediate simulation results, identifies the most sensitive device parameters, and progressively walks next simulations toward the precise lowest total variation that just reaches the measurement fail criterion.

Figure 8 depicts the outcome of an MPP search involving two device parameters in probability space. The variations of two parameters are represented by axes  $x_1$  and  $x_2$ , measured in standard deviation units of sigma. The red line is the loci of points that achieve a target delay of 200ps. In probability space, the set of points of constant total variation is a circle. Concentric circles represent progressive contours of constant total sigma. The circle of 6sigma is the minimum total sigma that just reaches the loci of 200ps delay at a single point  $(\sigma_1, \sigma_2)$ . This singular point of contact is referred to as the most probable point of failure.

When MPP delay analysis is extended to a circuit including  $n$  device parameters, the target response becomes an  $n$ -dimensional surface (not necessarily flat). Likewise, a contour of constant variation becomes an  $n$ -dimensional sphere. MPP searches probability space to find the sphere of minimum radius  $\sigma_{Total}$  that just reaches the target response surface at a singular point  $(\sigma_1, \sigma_2, \dots, \sigma_n)$  as shown in [c].

$$\sigma_{Total} = \sqrt{\sigma_1^2 + \sigma_2^2 + \dots + \sigma_n^2} \quad [c]$$

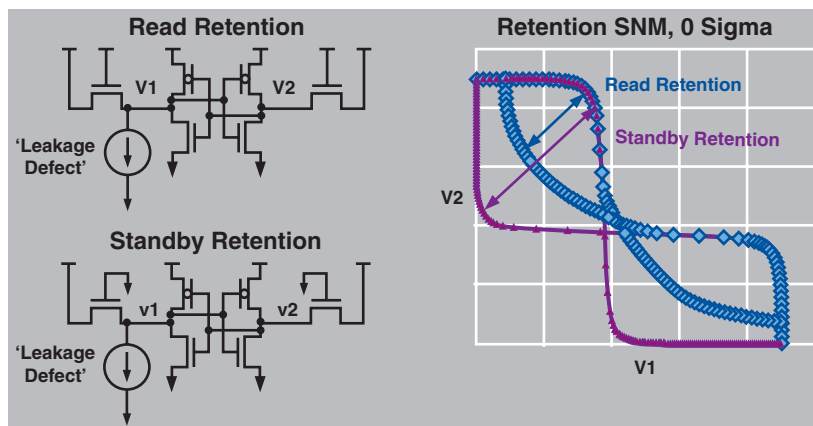
A collection of MPP variation assignments, commonly referred to as a Pareto, indicates relative sensitivity to each of the  $n$  parameters. MPP-based Pareto analysis can guide a design engineer to improve a circuit's robustness through device sizing or topology selection. As depicted in Figure 9 for an RF write at  $V_{min}$ , the magnitude of sigma assigned to device  $p_1$  is  $> 4$  and therefore would be a candidate for resizing.



**Figure 9:** RF Writes—MPP Pareto at Vmin

### SSA Retention Disturb Vmin

In addition to jam writes, SSA reads include a competing constraint for read Vmin that does not exist in RF. Reads are performed through the pass transistor access port onto bitlines that have been precharged to VCC. During the read access the charge transfer from bitlines collapses internal storage nodes of the bitcell, thereby reducing the static noise margin, as depicted in Figure 10. RF bitcells avoid this retention disturb by reading through an isolated domino read port.



**Figure 10:** Read and Standby retention

A key observation of aging variation is that jam Vmin tends to improve with aging, while retention disturb Vmin tends to degrade with aging.

### SSA Bitcell Selections for Vmin

Core and Uncore bitcells are selected for their Vmin yield after redundant repair. Larger SRAM cell sizes support lower Vmin. Core 270KB mid-level

cache arrays use the largest 0.256um<sup>2</sup> XLV bitcell for yields up to 700mV. The Uncore last-level cache chooses two bitcells, a 0.199um<sup>2</sup> ULV bitcell for the 240-720KB Tag arrays and the smallest 0.171um<sup>2</sup> LV bitcell (the SEM photo shown in Figure 12) to maximize density in the 4-12MB data arrays. For a modest redundant repair, the ULV bitcell achieves a Vmin of 800mV, whereas the LV bitcell Vmin is 100mV higher as shown in yield versus VccMin in Figure 11 [5].

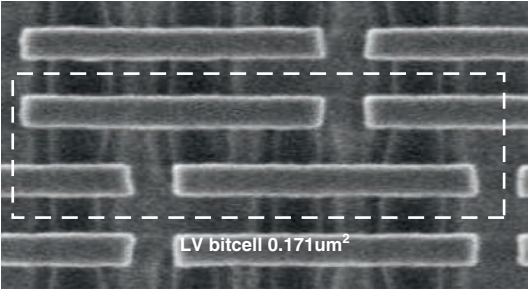


Figure 12: Bitcell for large, last-level caches

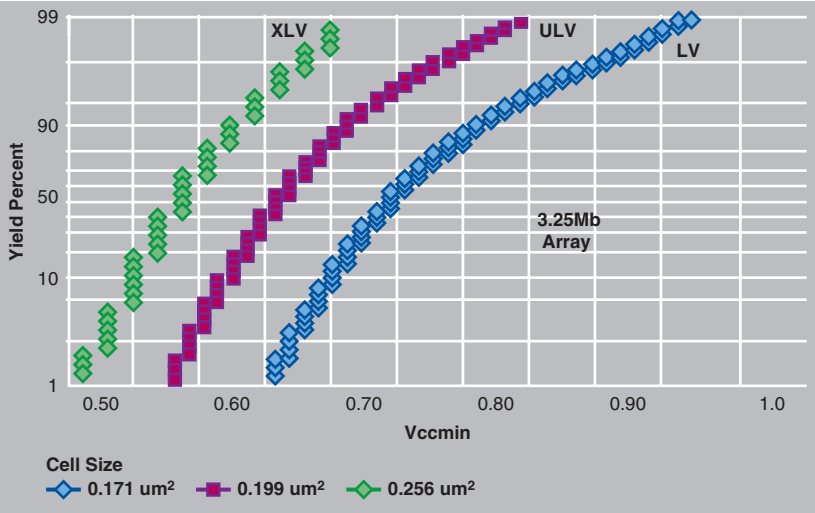


Figure 11: Cache bitcell selection for Vmin

An LLC data array applies double error correct, triple error detect (DECTED) encoding, in addition to redundant repair, to recover defective bitcells and thereby bring LLC-Data Vmin in line with Tag Vmin. The impact of redundant repair and DECTED ECC on cache yield and Vmin are illustrated in Figure 13.

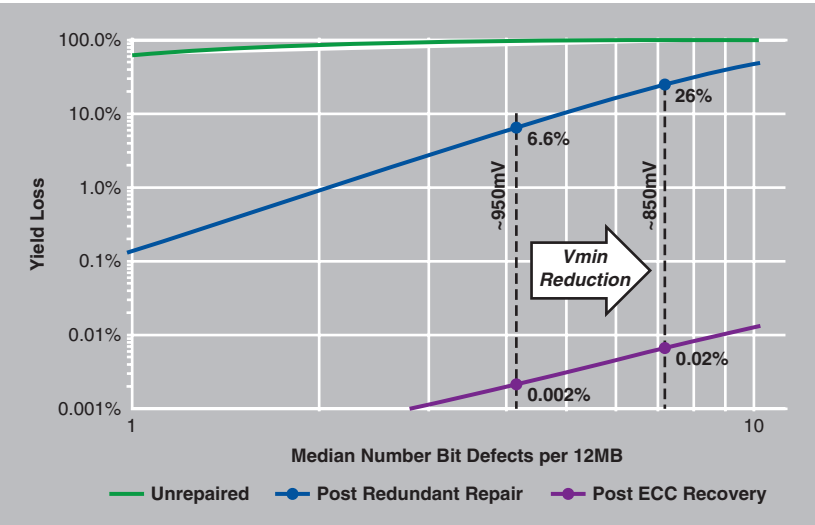
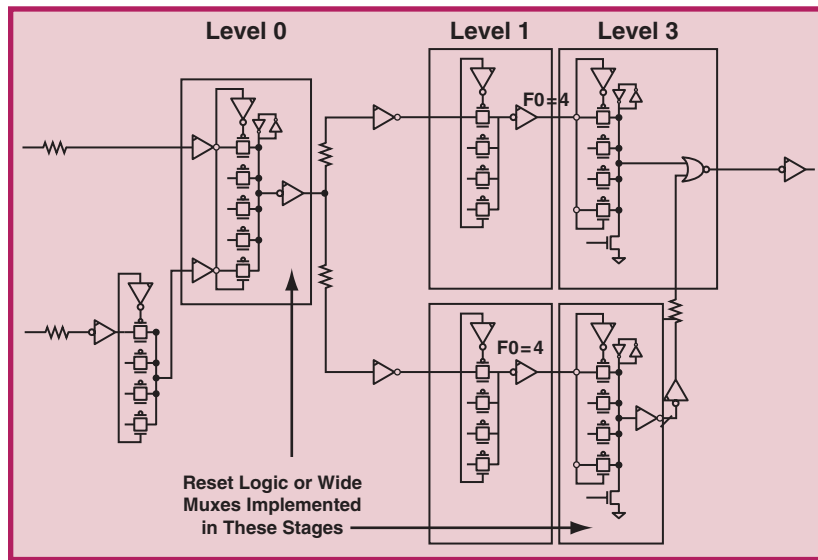


Figure 13: Yield loss improvement with Error correction

## Domino to Static and Efficient Pipeline Depth

We made use of several circuit techniques in the Nehalem design that gave us the benefit of domino in a static design. Significant among these was our use of wide fan-in pass-gate and skewed circuits, and embedding logic with reset conditions to approximate low-latency domino circuits. With the benefit of static logic, we balanced the pipelines to eliminate all hard edges that are the source of most post-silicon speed paths and miscorrelations. An example is shown in Figure 14 of the data-cache alignment path that has been traditionally built with domino or sense logic. The figure shows wide fanin logic and reset logic embedded in muxes that allow us to implement wide datapaths.



**Figure 14:** Data cache alignment path demonstrating wide fanin static logic

## Nehalem Density Improvement

The Nehalem design reduced chip area by about 20% on the same technology node compared to similar products through innovations in memory arrays, layout innovations, standard cell library extensions, and optimization of logical to physical mapping.

### Memory Array Innovations

For all highly-ported RF blocks, the traditional memory array layout that embeds read ports with each storage element (Figure 15) was reorganized.

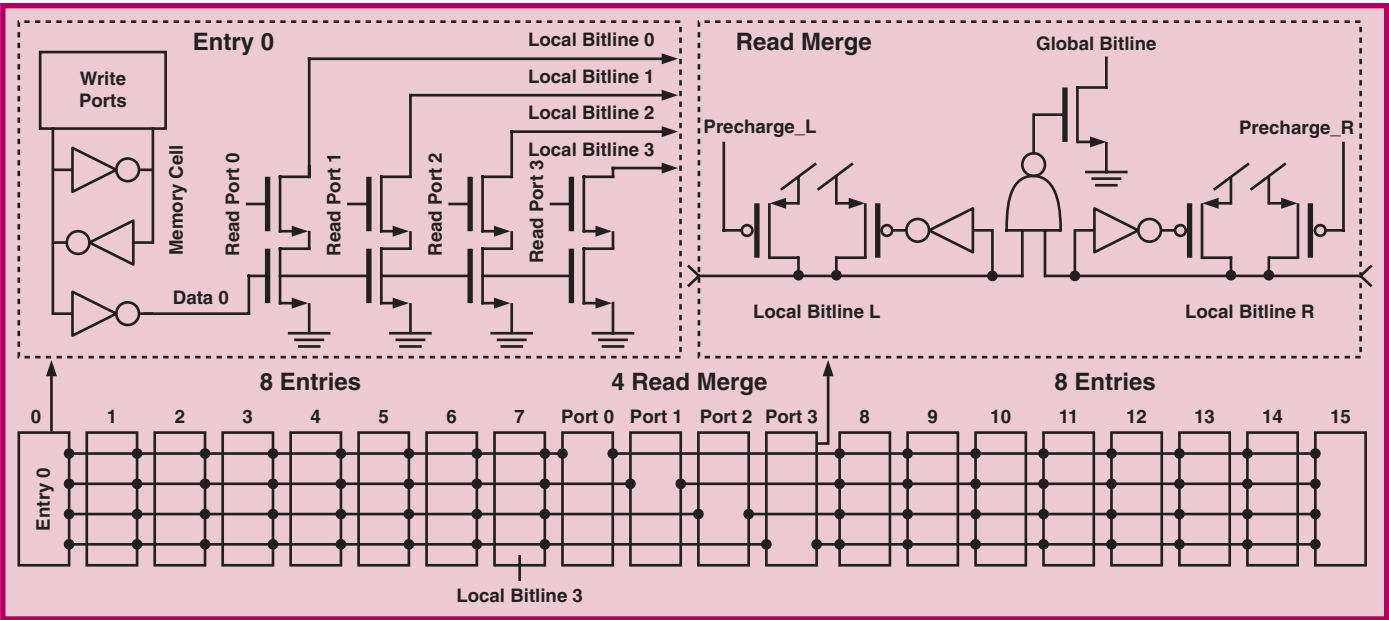


Figure 15: Traditional 16-entry bundle with 4 read ports

In the new design, the read ports are clustered into a separate cell (Figure 16).

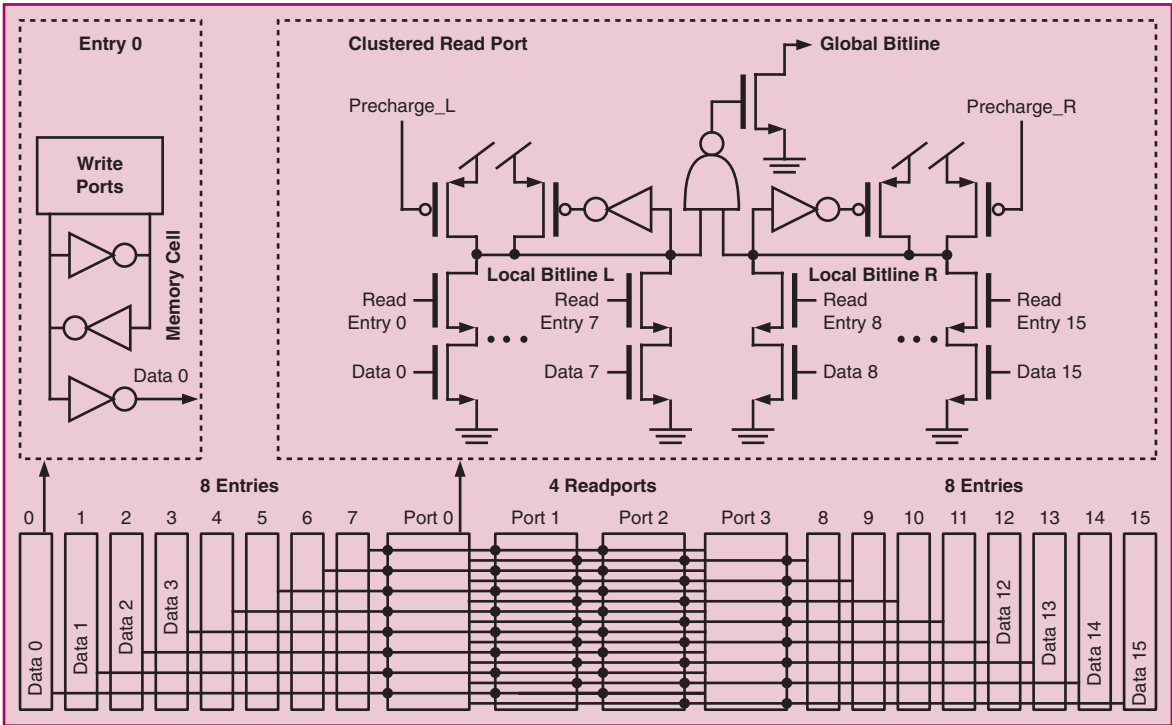


Figure 16: New 16-entry bundle with 4 clustered read ports

At iso-delay this new organization reduced the local read-bitline length by 60%, the read pulldown size by 50%, and the layout area by 10% for lower ported (4 ports) cases to 30% for highly ported cases (more than 12 ports). As a consequence, coupling noise onto the read bitline is significantly reduced, read wordline driver size drops by 30%, cross-talk between read and write wordlines is eliminated, and coupling noise from adjacent read wordlines is cut in half. Overall, this innovation results in a smaller, lower-power and a more robust design.

### **Layout Innovations**

The Intel 45nm process added a local interconnect layer. This was successfully exploited to reduce the area of many wire-limited layouts. The local interconnect was used to reduce the memory cell width by a track. Highly ported RFs could use a local interconnect to free up design interconnect, and dense structures such as Read Only Memory (ROM) were kept to the smallest device pitch. In addition, the standard cell library obtained pin porosity for improved routing.

### **Standard Cell Library Extension**

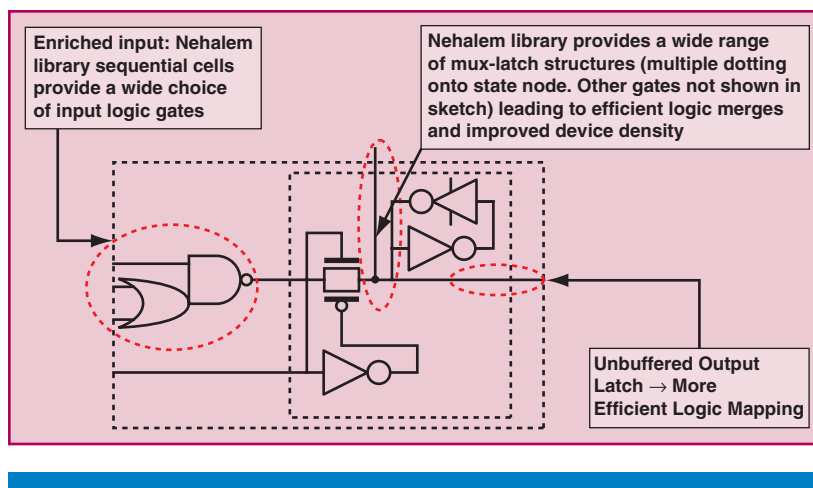
The standard cell library and the metal pitches on Nehalem were designed to unify the bit and lego pitch to enable a single standard library across all design domains.

The standard cell library was additionally augmented to include bit-pitch and entry-pitch matched combinatorial CMOS cells tailored to arrays in the critical cut of the die. This resulted in up to a 50% area reduction in the periphery of these arrays. The layout innovations described earlier permitted the development of a much larger set of effective cells. Among these was a 4-track wide latch that was slightly taller but much narrower than the 7-track cell it replaced. This design enabled tiling 4 latches into one row of the 18-track bit-pitch of a 64-bit array. Another example consisted of merged NAND-inverter cells whose height was matched to the entry-pitch and whose I/O ports were matched to the wordline and address line positions.

Substantial logical content innovations were added to the Nehalem library to maximize the layout efficiency of standard cell designs:

- By removing the embedded output driver within multi-stage logic gates or sequential cells (flip-flops and latches), the output logic stage was opened up for effective use in datapath design thereby converting it into more complex logic for more optimal driver location placement.
- The primary intent of the library was to provide designers a more enriched library that would enable better logic mapping and improve the efficiency of logic gates.
- Nehalem identified that a richer set of multiplexor (mux) designs and logic embedded mux and mux latch families would lead to substantially improved logic mapping in logic/timing critical design blocks.

- Nehalem library innovations were made tool friendly and were easily automated to help design convergence flows and tools. An example in Figure 17 shows where allowed routable distances on outputs of unbuffered-output sequentials were selected based on evaluations of compliance with noise checks and tool-flows that use specific cell designs.



**Figure 17:** Embedded input-logic, unbuffered-output CMOS latch on Nehalem and Westmere microprocessors

#### HighHigh and Medium Speed and Leakage Cells; and Low Leakage Cells

The Nehalem library provided a wide menu of speed and leakage choices to designers so that critical paths could be mapped to the high-speed cells (at higher leakage cost), and cells in paths that were not critical would map to the low-leakage (and slower) cells. These cells were constructed from the three types of transistors available on Intel's industry-leading, high-K metal-gate, 45nm technology: high-performance, nominal, and low-power transistors.

#### Library Design Efficiency

A key productivity goal of library design was reached in close cooperation with Intel's Process Technology development teams by implementing layout design rules that allowed the library to be laid out once for a given transistor type and the cells could be remapped to meet both low power and high performance design targets.

With an eye toward post-first-silicon speed fixes, the library layout was such that flip flops were designed with multiple amounts of internal clock delays (which translates to varying amounts of "time-borrowing" or "transparency") between slave and master; yet, the layout footprints were compatible to container sizes, pins, ports, and power hookups. The consequence of this to post-silicon was that during speed-path fixes, often these flops could just be swapped in or out without perturbing design-block floorplans. This choice had a good time-to-market impact on post-silicon frequency-push work.



### Optimization of Logical to Physical Mapping

For arrays that access logically sequential entries for either read or write (or both) operations, entries were organized into enough banks to prevent access to more than one entry in a bank.

The combination of the four techniques outlined earlier resulted in up to a 50% area reduction in the largest multi-ported memory arrays. When combined with the 65nm to 45nm process shrinkage, a 75% area reduction could be achieved for the same architectural algorithm.

## Process Technology Innovations to Enable the Nehalem Processor

Nehalem co-optimized process and design to obtain the optimal transistor and the metal stack for higher performance and to also obtain area and power savings. We list the process tradeoffs and the optimizations in this section to achieve these goals.

### Co-Optimized Design of Process Technology Transistors and Nehalem Microprocessors

The 45nm generation marked the introduction of Intel's breakthrough HighK + Metal gate transistor [6]. The transistors were adapted to deliver the high performance for the Nehalem microprocessor.

#### Early Focus on $V_{min}$ (Lowest Operating Voltage)

As mentioned earlier one of the main goals of the Nehalem program was  $V_{min}$  (lowest operating voltage). The goal of the designers of Intel's 45nm technology transistors was to find the highest performance transistors across the entire voltage range of operation. In addition, the transistors were designed to be able to achieve the aggressive low-voltage targets of the Nehalem microprocessor. These two factors led to the best optimization of transistor leakage, density, and performance on Intel's 45nm highK, metal-gate technology.

#### Rejection of Body-biasing Technology

The design team also evaluated whether body-biasing was a useful technology in the 45nm generation of Nehalem processors—a key evaluation. We rejected body-biasing, because we could not control the transistor channel through the substrate. Instead, we focused on (a) making power-gating transistors work to control standby leakage and (b) getting the lowest operating voltage to improve the power/performance tradeoff of the microprocessor.

#### Transistor Choices Optimized to Design Needs

The Nehalem processor has a natural technology separation between the core and the Uncore (all the non-core circuitry). The logic gates of the latter are, in general, under less performance pressure. The effect of this on process technology was that the core was primarily designed with 'fast' (high-leakage) and 'medium' (medium leakage) transistors, while the Uncore was designed with 'medium' (medium leakage) and 'low power' (ultra-low leakage) transistors.

**“Thirty Percent Leakage Guideline”**

In general, optimizing device characteristics with product frequency yields a “30% leakage guideline,” which says that any design block that is not clamped at turbo voltage or frequency-scaling down below  $V_{min}$  should be optimized so that approximately 30% of leakage is in the voltage-frequency scaling domain. This guideline was followed to set primary leakage targets for transistors on Intel’s 45nm process technology so that the product leakage would be in the 30% range.

**Device Leakage Separation Optimized**

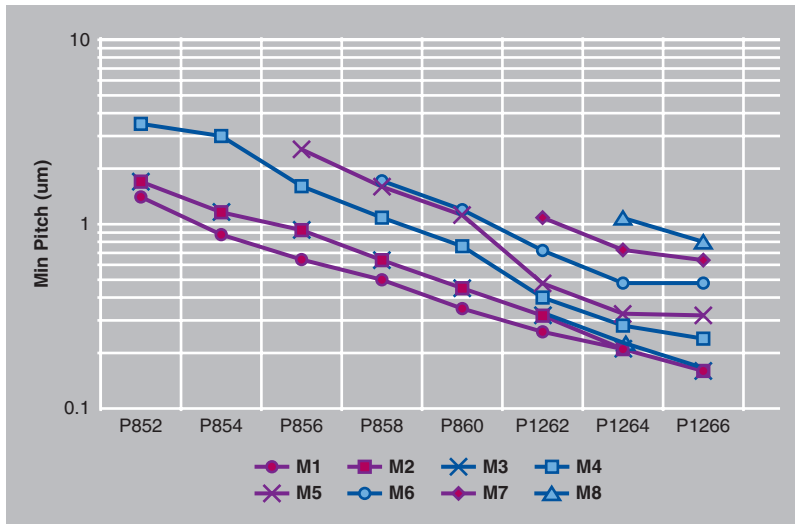
The relative leakage between the various transistor types used on the Nehalem processor was optimized to ensure that each transistor was used at the optimal percentage appropriate for that transistor type. In addition, the insertion rate of each transistor type in the design had to be able to be converged by design tools and power optimization flows.

**Optimal Number of Process Skews and Lines**

The number of process skews and lines (center targeting of entire distribution) in the 45nm generation was optimized for Nehalem based on the segments that parts would feed into and on respective power and performance requirements; die configurations; and maximizing yield.

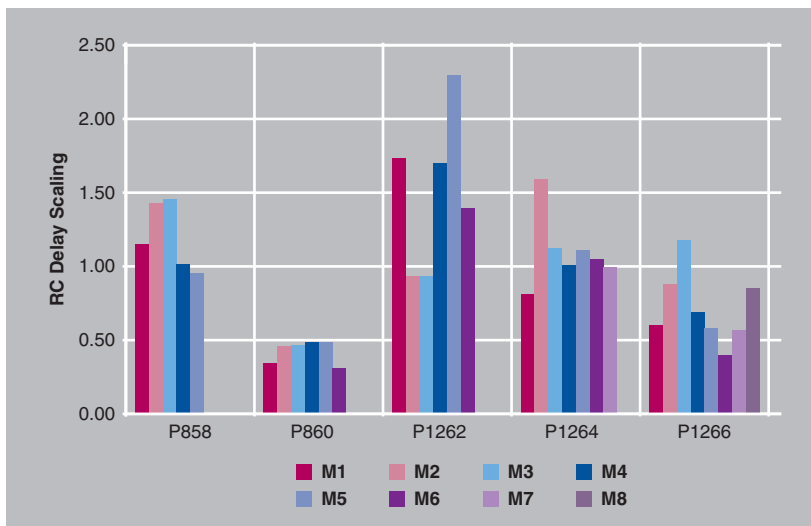
**Interconnect Strategy: High-density Lower Metal Layers and New Lower-layer Metal and Performance and Density Optimized Upper-metal Layers**

In the Nehalem process, the RFs and the standard cell libraries benefited from a new interconnect layer that increased the porosity of the pins. Traditionally the lower-layer metals were scaled to obtain a high-density, gate-contacted pitch. The metal layers were designed with a uniform pitch standard cell library. The choice of the high-density, lower layer, coupled with the added gate interconnect layer, allowed us to create a single datapath library at a fixed height for RFs, datapath, and control logic. Additionally, to obtain high bandwidth with performance, the upper-layer metals were designed to obtain both performance and density as shown in Figure 18. By choosing 45nm Nehalem metal pitches we were able to scale the Nehalem processor to the next-generation Westmere processor in the 32nm processor; this was achieved with minimal effort, and it resulted in enhanced performance. A new low-resistance, power-delivery layer was added to enable power gates and separate voltage domains that allowed for the low-power design of the Nehalem processor.



**Figure 18:** Metal1(M1) to Metal8(M8) pitch scaling on 45nm technology (P1266)

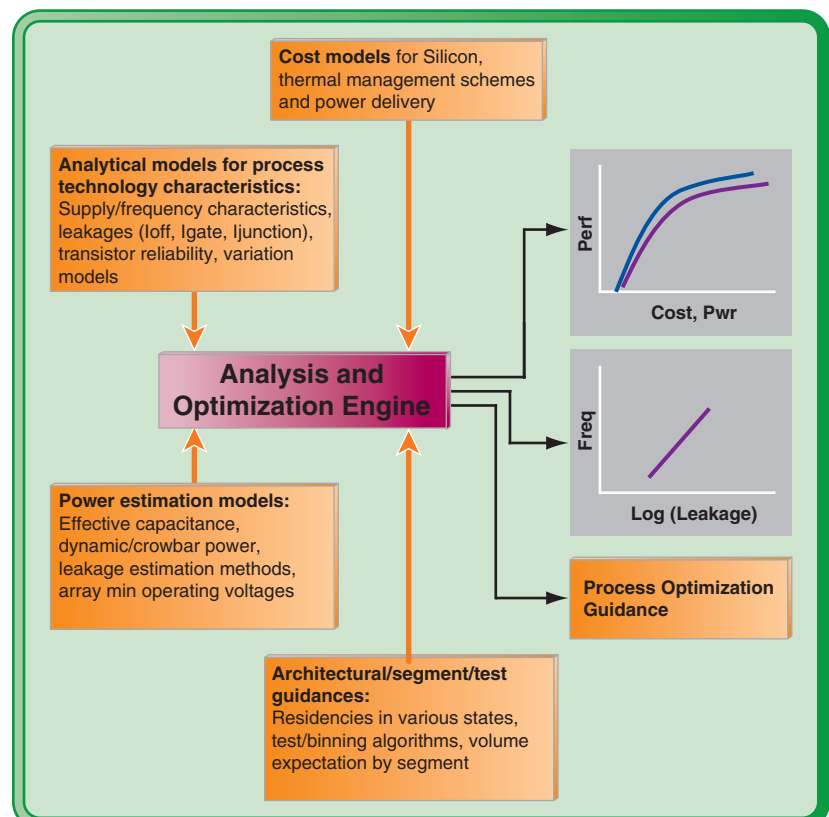
Figures 18 and 19 show the 45nm (P1266) Metal1 (M1) to Metal 8 (M8) layers that were created to obtain a low-latency design with reduced interconnect delays compared to past processor designs. The added benefit of this scheme, compared to earlier-generation products, was that it helped us increase process scaling while maintaining delays.



**Figure 19:** Delay scaling of Metal1 (M1) to Metal8 (M8) on 45nm (P1266) technology

## Unified Framework for Entire Platform Power/Performance Modeling and Optimization

The Nehalem program introduced an innovative new evaluation testbench called *Blizzard* that combines low-level transistor modeling with product-level predictions of effective capacitance, frequency, standby leakage, and speed/lkg distribution; platform-level expectations of thermal environment and power-delivery loadlines; and architectural expectations of residency in various states to make the best predictions of binsplits, thermal design power (TDP), and average power. Thus, Blizzard was very useful in evaluating whether the entire product distribution would land enough volume in various segments at the expected power-performance points. A system-level sketch of this analysis and optimization framework is illustrated in Figure 20.

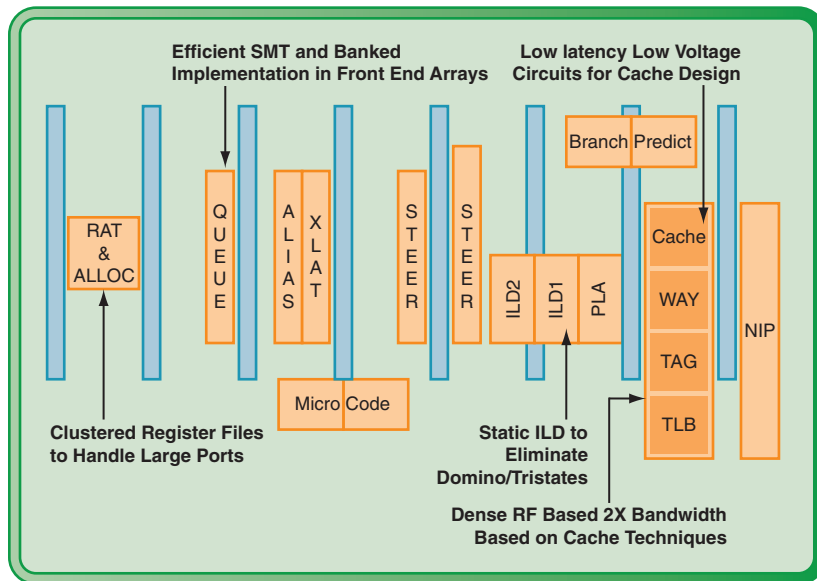


**Figure 20:** Top-level design power and performance optimization tool

## Example of the Outlined Optimizations on the Nehalem Frontend and Register Allocation Table

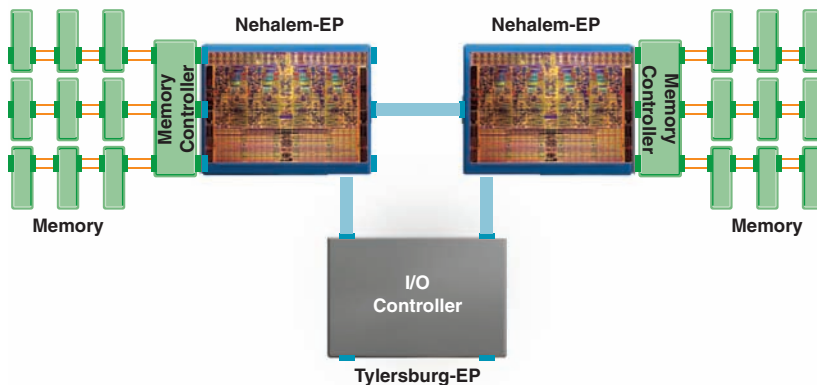
We employed the optimizations discussed in this article in critical areas of the Nehalem pipeline. An example of the front-end and the Register Allocation Table (RAT) in the out-of-order section is shown in Figure 21; the example demonstrates the applicability of some of these techniques. The instruction cache benefits from the increased bandwidth by the use of the RF. The cache-like

design techniques for these structures resulted in a high-density design. The instruction length decoder was built with a new algorithm that uses static circuits but maintains single-cycle throughput. Simultaneous multi-threading was successfully implemented, with minimal cost when compared to baseline costs, by using highly-efficient, static-banked algorithms that significantly reduce ports. The RAT was built with clustered designs to support highly-ported structures.

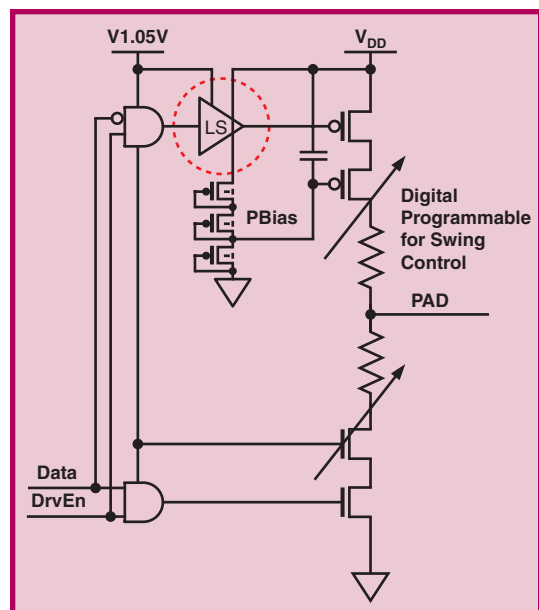


**Figure 21:** Front end and RAT pipeline illustrating Nehalem circuit innovations

## Memory Controller Integration and High-Voltage Tolerance



**Figure 22:** QPI—Nehalem dual socket with DDR and quick path interconnect (QPI)



**Figure 23:** High-voltage tolerant DDR3 Output Driver

The Nehalem family of processors, implemented on 45nm technology, integrates the memory controller that supports several DDR3 channels operating at up to 1333MT/s and delivering a bandwidth of up to 32GB/s per socket and the Quick path Interconnect (QPI) as shown in Figure 22 to build scalable systems. The Westmere processor, implemented on a 32nm technology node, extends Nehalem's DDR3 memory support to include low-voltage DDR3-LV. This requires the memory controller to run at either 1.5v DDR3 voltage or 1.35v DDR3-LV voltage without performance or power degradation [2]. Furthermore, both the Nehalem and Westmere processes support only natively "thin gate" transistors with a maximum allowed gate voltage below the DDR voltage. The combination of variable supply voltage, high frequency, low power, and transistor  $V_{MAX} < V_{DD}$  created challenges in the design of the the output driver. The push-pull voltage mode (driver shown in Figure 23) addresses these constraints [2].

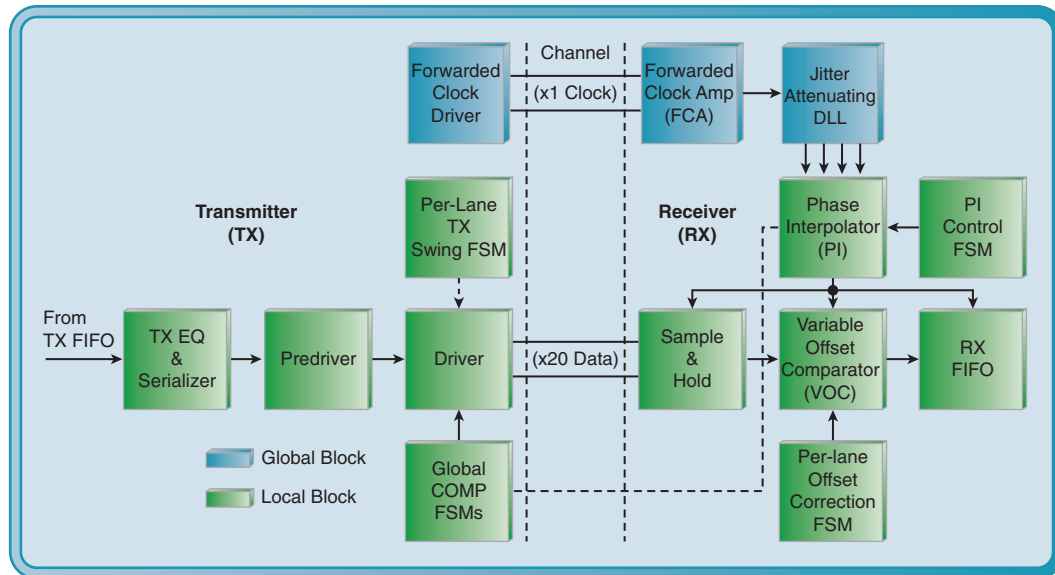
For high-voltage protection, the output is fully cascoded with a PMOS biased at  $V_{DD}/3$  and an NMOS biased by using the existing logic supply. To reduce power and jitter, PMOS-bias (Pbias) is generated with a high-impedance voltage divider heavily decoupled to the  $V_{DD}$  rail. The pull-down NMOS cascade is implemented by using normal logic with no gate oxide reliability concerns. However, the pull-up path requires a level shifter with an output that swings from Vdd to (Vdd-logic voltage). To ensure low power and adequate settling time, a pulsed-level shifter is used. When switching, the pull-down is pulsed to quickly transition the level shifter with fast slopes. However, for low standby power, the main pull-down is disabled while the low level is maintained by two series NMOS diodes and the PN diode clamp [2].

## QPI Variations Tolerant Link

With higher core count, the demand for I/O bandwidth increases. To meet these demands, we implemented a dual Intel QPI link for communication between processors and chipsets. The QPI link is a coherent, point-to-point, fully-differential forwarded-clock, high-speed link that delivers up to 25.6GB/s bandwidth at 6.4GT/s. The design of high-speed links, with much higher analog content, in deep-submicron technologies, poses several challenges due to lower voltage headroom, degraded transistor properties such as  $R_{out}$ , and higher device variations. The Nehalem generation of processors includes circuit architectures and adaptive techniques to overcome these just-mentioned challenges, while delivering energy-efficient performance across a wide array of platforms.

Figure 24 shows a high-level diagram of the QPI transmit and receiver Analog Front End (AFE) architecture.

The Transmitter (Tx) is a scalable current mode driver with programmable voltage swings [2]. At the receiver (Rx,) a jitter attenuating DLL, coupled with two independent phase interpolators (PIs) optimized per lane, clock two



**Figure 24:** Quick Path Interconnect (QPI) Transmit and Receiver Analog Front End Architecture

parallel variable-offset comparators (VOC) [1]. Closed-loop state machines compensate for voltage/temperature and random/systematic variations. The compensation network is divided into global and local loops. The global machines compensate the Tx and Rx termination resistances, the Tx swings for systematic variation and tracks the voltage and temperature (VT) drifts. The global network also produces variation-independent biases for the Rx circuits. The local compensation machines, on the other hand, add a per-lane offset to correct for lane-to-lane random variations [2].

## Conclusions

The Intel Nehalem and Westmere processors were designed to be highly energy and area efficient, high-performing processors. We achieved these goals by simultaneously optimizing the process, circuits, and the design.

Power efficiency was achieved by designing circuits with a large dynamic voltage range. The cores and the Uncore were developed with a modular clocking and power architecture that supports the performance and power range.

The large performance improvement was achieved with new circuit techniques for RFs, caches, and the standard library that allowed us to build a higher performance machine with minimal area growth.

The high-performance core can be used in different Personal Computer segments, and the performance and platform scalability is supported by the integrated DDR and QPI circuits.



## References

- [1] Rajesh Kumar, Glenn Hinton. “A family of 45nm IA Processors,” ISSCC Technical Digest, February 2009.
- [2] N. Kurd, et al. “Westmere: A Family of 32nm IA Processors,” ISSCC Technical Digest, February 2010.
- [3] N. Kurd et al. “Next Generation Intel® Core™ Micro-Architecture (Nehalem) Clocking,” IEEE Journal of Solid State Circuits, vol. 44, issue 4, pp. 1121–1129, April 2009.
- [4] G. Shamanna et al. “A Scalable, Sub-1W, Sub-10ps Clock Skew, Global Clock Distribution Architecture for Intel® Core™ i7/i5/i3 Microprocessors Clocking,” Symposium VLSI Circuits Tech Technical Digest, June 2010.
- [5] P. Newman; J Miller. “Sigma Calculator Method and Yield Metric Roll-up for Full-Chip VccMin Assessment,” Intel Design and Test Technology Conference. 2007.
- [6] Mark Bohr, Kaizad Mistry et al. “Intel Demonstrates High-k+Metal Gate Transistor Breakthrough on 45nm Microprocessors.” Available at [http://download.intel.com/pressroom/kits/45nm/Press45nm107\\_FINAL.pdf](http://download.intel.com/pressroom/kits/45nm/Press45nm107_FINAL.pdf). January 2007.

## Authors’ Biographies

**Nasser A. Kurd** received an MSEE degree from the University of Texas in San Antonio (UTSA) in 1995. He joined Intel Corporation in 1996 where he is currently a Senior Principal Engineer in the Microprocessor Development Circuit Technology Group. His group is leading next-generation microprocessor clocking technologies. He has been involved in clocking, analog design, analog process scaling, and I/O for several microprocessor generations. Between 1994 and 1996, he was with AMD in Austin, Texas in the 29K microprocessor development group. Nasser has served on several conference committees, authored and coauthored several internal and external publications, and holds 29 granted patents. His e-mail is Nasser.a.kurd at intel.com.

**Kumar (Anshu) Anshumali** obtained a Master’s degree in Electrical Engineering from NTU in 1993 and is an alum of IT-BHU (India) and the University of Illinois, Champaign-Urbana. He is a Senior Principal Engineer and has been with Intel Corporation since 1992. His primary areas of focus have been optimizing process technology for lead Intel microprocessors, low-power and advanced circuit design, CMOS libraries for lead processors, and unified power-performance optimization frameworks for Intel microprocessors. He holds five issued patents, has written nine papers (some internal), and he received an Intel Achievement Award in 2003. His e-mail is kumar.anshumali at intel.com.

**Terry I Chappell** completed his PhD degree in Electrical Engineering at the University of California at Berkeley in 1978. From 1978 to 1995 he worked at the IBM Thomas J. Watson Research Center on high-speed CMOS SRAMs and circuits. Currently he is a Senior Principal Engineer at Intel Corporation in the Microprocessor Development Circuit Technology Group working on co-optimization of circuit and process technologies. He is a developer of several novel CMOS circuit topologies including self-resetting (pulsed) CMOS circuits. Terry is an author on several internal and external publications and holds 36 patents. His e-mail is [terry.chappell@intel.com](mailto:terry.chappell@intel.com).

**Wilfred Gomes** obtained a Master's degree in Electrical Engineering from the University of Hawaii in 1995 and a Bachelor of Technology degree in Electronic Engineering from the National Institute of Technology, Calicut in 1992. From 1994 to 1997 he worked at Cadence Design on RTL and logic synthesis. He joined Intel in 1997 and is currently a Principal Engineer. He has worked on the synthesis of domino control logic and the development of the Register File synthesis tools. He has also been involved in co-optimizing process, design, and micro-architecture to achieve the frequency, area, and power goals of several generations of Intel microprocessors. His e-mail is [wilfred.gomes@intel.com](mailto:wilfred.gomes@intel.com).

**Jeffrey L. Miller** graduated in 1986 from the University of Illinois at Urbana-Champaign. He is a Principal Engineer and has worked for Intel Corporation since 1990, specializing in Cache and Register File architecture and circuits in server, desktop, and mobile microprocessors spanning ten process technology generations. His e-mail is [Jeffrey.l.miller@intel.com](mailto:Jeffrey.l.miller@intel.com).

**Rajesh Kumar** is an Intel Fellow and Director of Circuit and Low Power Technologies. In this capacity he directed the development of circuit and power technologies as well as interface to process technology for the Nehalem & Westmere family of processors. He is currently directing similar work on Intel's 22nm TOCK and manages the circuit technology team in the Microprocessor Development Group. Rajesh received a Master's degree from the California Institute of Technology in 1992 and a Bachelor's degree from the Indian Institute of Technology in 1991, both in Electrical Engineering. His e-mail is [rajesh.kumar@intel.com](mailto:rajesh.kumar@intel.com)

## Copyright

Copyright © 2011 Intel Corporation. All rights reserved.

Intel, the Intel logo, and Intel Atom are trademarks of Intel Corporation in the U.S. and other countries.

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

Requires an Intel® HT Technology enabled system, check with your PC manufacturer. Performance will vary depending on the specific hardware and software used. Not available on Intel® Core™ i5-750. For more information including details on which processors support HT Technology, visit <http://www.intel.com/info/hyperthreading>

Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM). Functionality, performance or other benefits will vary depending on hardware and software configurations. Software applications may not be compatible with all operating systems. Consult your PC manufacturer. For more information, visit <http://www.intel.com/go/virtualization>

Requires a system with Intel® Turbo Boost Technology capability. Consult your PC manufacturer. Performance varies depending on hardware, software and system configuration. For more information, visit <http://www.intel.com/technology/turboboost>

# THE ROAD TO PRODUCTION—DEBUGGING AND TESTING THE NEHALEM FAMILY OF PROCESSORS

## Contributors

**Derek Feltham**  
Intel Corporation

**Colin Looi**  
Intel Corporation

**Keshavan Tiruvallur**  
Intel Corporation

**Hermann Gartler**  
Intel Corporation

**Chuck Fleckenstein**  
Intel Corporation

**Lily Looi**  
Intel Corporation

**Michael St. Clair**  
Intel Corporation

**Bryan Spry**  
Intel Corporation

**Tim Callahan**  
Intel Corporation

**Rob Mauri**  
Intel Corporation

## Index Words

Design For Test  
JTAG Interface  
System Validation  
Mirror Port  
Survivability

*“The first time an operating system boots on a microprocessor is a significant milestone.”*

## Abstract

This article describes innovations in the Design for Validation (DFV), Design for Test (DFT), and High Volume Manufacturing (HVM) features of the *Intel® microarchitecture code name Nehalem family of products*. These features are critical to debugging the complex architecture of this product family, and they are key to bringing the product family to market on schedule. The challenges we faced included dealing with new high-speed interfaces, a high level of integration, multiple cores, multi-level on-die caching, extensive architectural changes over previous architecture generations, and the integration of PCI Express (PCIe) Gen2 and graphics in client versions of the product. Test and debugging features described in this article include the test access mechanisms, mirror port technology, manufacturing test features, in-system observability hooks for debug, and survivability features used in the platform. Key debugging experiences are highlighted while showing how these features enabled a quick and predictable path through component and system debugging stages through to production.

## Introduction

Debugging is always an exciting phase in the development of any new microprocessor. This is when the design team first gets to see its design fabricated in silicon and takes on the new challenge of translating that design into a bug-free, robust working product. In the early part of this post-silicon challenge, the first time an operating system boots on a microprocessor is a significant milestone and is cause for a jubilant pause. This milestone signifies that a minimum level of functionality has been achieved, and it sets the pace for the rest of the post-silicon debug activities.

The first time the Nehalem family of processors booted was equally exciting. In fact, it was even more exciting than previous technology generations because this family of processors included a significant platform transition from frontside bus (FSB) to the Intel® QuickPath Interconnect (QPI) technology, an integrated 3-channel DDR memory controller, multiple cores, multi-level caching, on-die power-management capabilities, and core performance enhancements. Later versions of the product family also introduced a second platform transition from QPI to PCI Express (PCIe) Gen2 interface, and introduced integrated graphics for the client products.

The transition to QPI was especially challenging for post-silicon debug activities, because QPI replaced the FSB that had been used for many previous technology generations for debugging and test access. Because of this transition, the FSB information was not available to the Nehalem family of products. Consequently, all component debugging, in-house system-level validation, customer/OEM platform debug enabling, and high-volume manufacturing (HVM) test delivery capabilities had to be re-engineered. We faced another challenge because of the highly integrated nature of this family of products. The integrated memory controller, and in later versions, the integrated PCIe bus and graphics, served to further obscure our ability to observe the inner workings of the microprocessor. To add further to the post-silicon challenge, the highly segmented nature of our industry required more than twenty variations (SKUs) of this product. Even though these SKUs share a common design database with many re-used elements, from a post-silicon perspective, each variant represented a completely new product that required full debug, validation, and manufacturing qualification.

We answered these challenges with features such as a mirror port to provide visibility to the high-speed interfaces [1], on-die observability features to provide insight into internal transactions, and determinism and pass-through modes for HVM. The Nehalem family platform transition forced a fundamental reliance on in-silicon debug hooks and related tools. In this article, we describe these capabilities.

We first provide an overview of Design for Test (DFT) and Design for Validation (DFV) features implemented in the Nehalem family of processors. We then take a more detailed walk-through of our post-silicon debug experiences, describing the contents, features, mirror ports, our power debug experiences, PCIe debug experiences, and survivability features. We conclude with a description of our test strategy and HVM enabling experience for this family of products.

## Overview of Design for Validation and Design for Test Features

All complex silicon products include features specifically targeted at enabling debug, validation, or manufacturing test. These features are not typically enabled during normal product operation, but they are instead accessed through special modes. These features are collectively referred to as Design for Validation (DFV) or Design for Test (DFT), or more simply as “DFx” features.

The high level of integration in the Nehalem processor family introduced several problems for HVM. Having four cores on the same die, an integrated memory controller, and ultimately for client products, integrated PCIe and graphics, presented new challenges to HVM.

HVM control is gained through the test access port (TAP) that traditionally had to interface with only one or two CPU cores. With four cores, we had to come up with a new working model: we introduced multiple TAP controllers,

*“The transition to QPI was especially challenging for post-silicon debug activities, because QPI replaced the FSB that had been used for many previous technology generations for debugging and test access.”*

*“The Nehalem family platform transition forced a fundamental reliance on in-silicon debug hooks and related tools.”*

*“Control is gained through the test access port (TAP) that traditionally had to interface with only one or two CPU cores.”*

*“All previously developed test content for the Nehalem could be reused for Lynnfield and Arrandale client products.”*

*“On the Nehalem processor the external observation points multiplied fivefold (three independent channels of DDR with two QPI links, and later PCIe and DMI). These were considerably more difficult to trace.”*

*“The mirror port provided a dedicated top-side port to “mirror” the link-layer traffic of the QPI bus for capture by a custom ASIC.”*

one for each core, and a master TAP to chain them together. While this sufficed for low-speed control, it did not for high-speed test data; these were loaded through the QPI and DDR interfaces by the use of an HVM mode.

We were further challenged in HVM with the even more highly integrated client products, codenames Lynnfield and Arrandale. The integration resulted in an embedded QPI bus and only a PCIe link. However, through the engineering of a deterministic pass-through mode, over several new clock domains, the HVM data could be loaded from the PCIe and passed to the internal QPI-like interface. This meant that all previously developed test content for the Nehalem could be reused for Lynnfield and Arrandale client products. We describe this pass-through mode and other unique HVM enhancements later in this article.

Traditional debug methods within Intel have relied heavily on observation of the FSB, which was introduced with the Intel Pentium® Pro processor and subsequently improved in only minor ways. The FSB provided a central view of activity in a processor by providing visibility for all memory and IO transactions; thus, it was a quick and efficient means of analyzing issues. Further, it was a single point of reference for critical replay technologies used for root cause analysis of processor bugs.

In contrast, on the Nehalem processor the external observation points multiplied fivefold (three independent channels of DDR with two QPI links, and later PCIe and DMI). These were considerably more difficult to trace. Even where all the interfaces could be captured together, they provided only a fraction of the information that an FSB could provide. Four cores and eight threads dynamically shared data invisible to the external interfaces. The coherency protocols were either managed within the die or distributed over multiple QPI links. In the case of the memory controller, the information available externally was captured in a foreign physical address whose correlation to system addresses varied according to DIMM configuration and BIOS setup.

To solve these problems, we took several approaches. First, we had to solve the problem of observing the QPI links on both Intel and customer boards with minimal perturbation to the platform. We used a novel invention known as a mirror port, which we describe in detail later in this article. The mirror port provided a dedicated top-side port to “mirror” the link-layer traffic of the QPI bus for capture by a custom ASIC.

Next, we had to enhance a traditional approach that created a “snapshot” of the state of the processor by freezing execution following a hang or context-based trigger, and then extracting the current state. In addition to advances in observation, we paid special attention to the ability to selectively disable (de-feature) key parts of the architecture at both a coarse- and fine-grain level for problem isolation and workarounds. All of these methodologies are detailed in the “CPU Debug Methods in a System” section of this article.

In addition, because of the integration of chipset features, we had to expand the toolkit available to work around or “survive” bugs found late in the debugging process (discussed later in detail).

## System Validation and Debug

We now describe the post-silicon content development, CPU debugging, system debugging with the mirror port, electrical validation, and survivability. We provide examples of DfX utilization to validate and debug the Nehalem family of processors. We illustrate several methods we used throughout the validation of this family of CPUs.

### System Validation Test Content

One important challenge for system debug was ensuring the right content would be available to stress systems for extensive post-silicon validation. Validation content planning, development, intent verification, execution, and debug were the focus of the content teams. We used multiple test environments to validate the functionality of the CPU. A new Random Instruction Test (RIT) environment was used for coverage of core architecture as well as for portions of the full-chip design. The RIT environment was used by both pre- and post-silicon validators to generate content for coverage of both instruction set architecture (ISA)[2] and micro-architecture features. Thus, teams could share features, lessons learned, and coverage tools and metrics. Tool expertise, content, and coverage feedback were developed and analyzed in the pre-silicon environment to be later leveraged in the post-silicon environment where cycle limitations were not a factor.

Post-silicon test environments also included a low-level focus test environment that consisted of algorithms focused on functions such as cache coherency, memory ordering, self-modifying code, power management, etc. The test environment also consisted of a memory consistency directed random environment [4, 5], a dedicated system validation operating system environment, which covered full-chip concurrency, cache coherency, isochrony, memory controller validation, and reliability, accessibility and serviceability (RAS), and specialized IO cards. The test environment also included data space validation.

Compatibility or commercial validation is the validation of a typical production system (including silicon, boards, chassis, ecosystem components, BIOS, OS, and application software subsystems) viewed as a real-world platform or application. The commercial validation content included Intel developed tests consisting of focused and stress scenarios, commercial off the shelf (COTS) software, as well as updated versions of software to support new CPU features.

For Nehalem, the system validation teams requested additional DFV hooks on silicon to provide internal visibility so as to understand what the tests and environments were covering—from an architectural as well as micro-architectural perspective. DFV hooks are micro-architecture features that aid the validation teams by exposing additional information for validation

*“One important challenge for system debug was ensuring the right content would be available to stress systems for extensive post-silicon validation.”*

*“A new Random Instruction Test (RIT) environment was used for coverage of core architecture as well as for portions of the full-chip design.”*

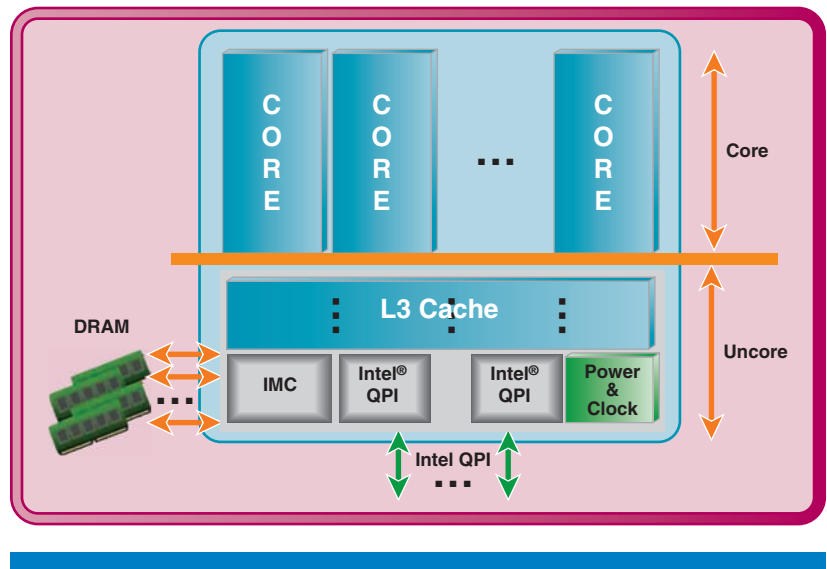
*“The commercial validation content included Intel developed tests consisting of focused and stress scenarios, commercial off the shelf (COTS) software, as well as updated versions of software to support new CPU features.”*



*“Microcode path coverage to these DFV features to provide post-silicon engineers visibility into a vast coverage space that is typically visible only at the pre-silicon stage.”*

suite analysis and internal events, allowing creation of key micro-architecture validation conditions, as well as to aid in debugging of failures. The new DFV hooks facilitated non-intrusive setup and capture of coverage information from specific areas of the microarchitecture. Given design and space constraints in silicon, post-silicon teams looked to leverage existing logic to add hooks for visibility. The visibility hooks consisted of new “validation” events added to the existing performance monitoring logic, architecture features, as well as debug features. We added microcode path coverage to these DFV features to provide post-silicon engineers visibility into a vast coverage space that is typically visible only at the pre-silicon stage. Opening up this important coverage space to post-silicon provides the capability to look for content holes in the coverage of areas such as Intel® Virtualization Technology, and the ability to re-steer content to target specific areas. Further, it provides greater confidence when making production release decisions.

An example of DFx use for both debug as well as intent verification feature is an internal inter-core triggering mechanism. A DFx hook was needed to permit triggering of the interconnect between the processor cores and the Uncore shown in Figure 1. This debug feature helped verify specific internal transactions and what responses were occurring and to what extent they were occurring. In addition, this capability also became a very useful survivability feature. The ability to observe inter-core transaction information allowed us to configure specific transaction types to be counted and to revise their content, if they were not generating the specific type or amount of traffic they expected. Mechanisms such as these also permitted the validation and architecture teams to see whether non-targeted content was generating specific types of transactions.



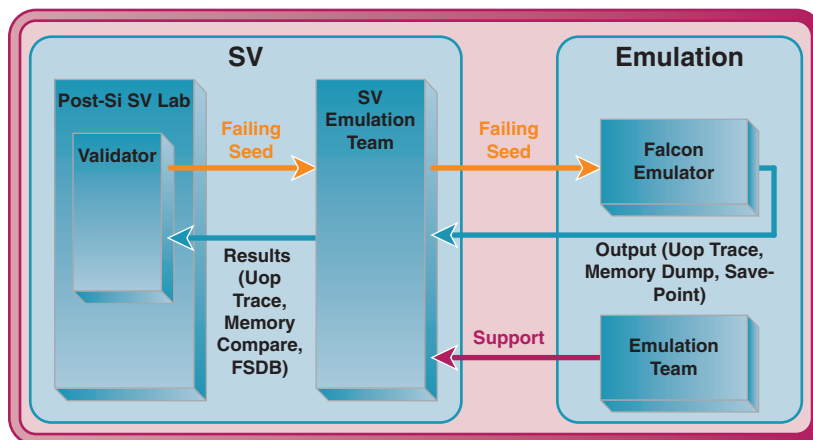
**Figure 1:** Nehalem modular design [3]



Given multiple post-silicon validation environments, as well as the requirement to monitor internal events that are not supported by utilities such as the VTune™ analyzer [6], the post-silicon team created a DFV hook based on microcode known as TAGEC — Tool AGnostic Event Coverage. This new DFV feature provided capabilities to set up and collect internal micro-architecture event counter information from the test environments with minimal alteration to the test environments of interest. This DFV helped us understand whether the intent of the test was being met or not.

Hardware emulation [7] is by no means new to hardware validation teams. It has been used for some time by various groups inside and outside [8] of Intel to speed execution of tests and also as part of particular debug flows. The emulation model [9] was used for content failure debug due to its high speed and visibility. Several system validation (SV) RIT post-silicon failures were reproduced by taking the failing tests and running them on the emulation models. Running these failing tests on the normal RTL simulation model would not have been practical given the length of time it took to rerun a test. Other classes of RIT failures were debugged by using the emulation model. These failures included RIT tool errata, as well as a number of architecture simulator errata.

*“Several system validation post-silicon failures were reproduced by taking the failing tests and running them on the emulation models.”*



**Figure 2:** High-level block diagram of Emulation Reproduction Flow

Figure 2 demonstrates the flow from the post-silicon system validation failure to emulation resulting in valuable debug information being provided to the validation engineer and silicon debug teams. The post-silicon validation team identifies the failing test, then works with the emulation team to run it on the emulation model. The emulation model can be stopped at any time to capture a save point and to restore state in a more traditional and slower RTL simulator. The emulation model can produce a micro-operation trace, memory dump, and Fast Signal Database (FSDB) file to aid debug of the failure in the pre-silicon model that ultimately originated in the post-silicon environment. Having this capability of taking failures back to pre-silicon is an extremely

*“Projects have expanded the use of emulation as a debugging tool in order to resolve silicon issues more quickly.”*

*“The Nehalem team developed a method of combining the ability to freeze writes to arrays with a freeze of the clock.”*

*“Pre-silicon tools could be transparently leveraged for decoding and for visualizing tracking structures and arrays.”*

powerful debugging tool: it allows us to observe nodes that would otherwise be unreachable. Validators and design engineers can use this debug information to determine whether the root cause of the error is related to hardware or software. Subsequent projects have expanded the use of emulation as a debugging tool in order to resolve silicon issues more quickly.

### **CPU Debug Methods in a System**

To deal with the increased level of integration in the Nehalem family, we developed several different techniques or borrowed significantly from prior architectures.

#### **Snapshot Debugging**

The basic method for debugging internal logic failures, used in several processor generations, involved freezing arrays and subsequently reading them; it was eventually dubbed “snapshot debugging.” The key to this method is to identify a triggering event, as close as possible to the failure, and to stop all updates to internal state when the event occurs. This is followed by extraction of the state for analysis.

The observation mechanisms for snapshot debugging relied almost exclusively on existing HVM DFT features; thus, they occupied very little additional silicon area. Taken in combination, control registers, array test, and partial scan and “scanout,” based on the internal signature mechanism, the snapshot debug methodology provided accessibility to almost every state element of the design. The Nehalem team developed a method of combining the ability to freeze writes to arrays with a freeze of the clock distributed to the Scan logic when a triggering event occurred. Then, in a carefully managed fashion, they extracted each individual piece of state data with the values that were present at the trigger. Key to this flow was the latch-B Scan design for Nehalem. This design incorporated two parallel latches in a latch design that allowed Scan data to be captured without disturbing the logic. By using the secondary latch, we were able to preserve the contents of the Scan cells at the triggering event without having to actually shift through the chain—normally a destructive operation—until all of the other DFT operations were complete.

Once the state was acquired from a snapshot dump, it was processed into a signal database that was matched in both hierarchy and signal name to the signal database acquired from full-chip simulations. This was a single cycle of data but a single cycle that was immediately familiar to pre-silicon design and validation. With the captured data in this format, pre-silicon tools could be transparently leveraged for decoding and for visualizing tracking structures and arrays.

However, this method does require significant knowledge of the internal microarchitecture; therefore, it meant that pre-silicon design and validation were heavily involved with post-silicon debug throughout post-silicon execution.

#### **On-die Triggering**

Because the snapshot debug method depended on triggers in close proximity to the initial failure, several internal triggering blocks were added to provide

localized trigger points for basic events (machine checks, IP and microcode matches). With the higher integration, we could not observe externally core-to-core and core-to-Uncore transactions. Therefore, the largest on-die triggering investment was for transactions between the core and the Uncore, which provided a triggering point analogous to the historical FSB triggers. Finally, to coordinate triggers in multiple clock domains, we implemented a distributed triggering network to synchronously stop execution throughout the processor.

While the on-die triggering was important, one of the key triggering innovations developed was the ability to trigger the snapshot from microcode by updating the microcode with a patch. Microcode patching is traditionally used for fixing microcode or hardware bugs in silicon without requiring a stepping of the part. In the past, microcode patching had been used on a very limited basis to capture data, based on theories about the bug; or to pulse a pin observable to an external logic analyzer. On Nehalem, a library of patches, dedicated to triggering, was developed that initiated a stop, based on unexpected exceptions or page faults. Because the patches could be modified based on the specific theory of the failure, they proved to be remarkably useful and flexible, and they provided contextual triggers that would have been impossible with any sort of dedicated hardware block.

#### Architectural Event Tracing

While the snapshot debug method proved successful in finding the root cause of many hardware failures, the single slice of time it provided did not apply well to all types of bugs and certainly was not always the most efficient way to analyze all problems. This was especially true of BIOS, software, and other platform failures. The Nehalem family of products included an external 8-pin parallel bus that worked as both a triggering network and alternatively as a slow-speed debug port where microcode could write packets for events captured by the JTAG hardware. Architectural Event Tracing (AET), a debugging method, grew out of this ability of the microcode to send packets on events and certain flows.

The concept behind AET is simple enough: instrument the microcode to send information about key microcode flows and transactions as it executes them. Some examples include exposing every machine state register (MSR) access, results of I/O operations, or page fault information including the addresses of the faults. Linear instruction pointers (LIP) and timestamps were included with each packet to create a timeline of execution.

What AET allowed us to do was to profile the basic execution of the processor during BIOS, test, or application execution in order to isolate the failure. It could rarely be used to determine the root cause of a bug in the hardware, but it accelerated isolation of failures because it gave the “10,000 foot” view of the failure. The debug team used that view to discover the high-level sequence of events that preceded the crash, or to determine where the part, that was

*“With the higher integration, we could not observe externally core-to-core and core-to-Uncore transactions.”*

*“While the on-die triggering was important, one of the key triggering innovations developed was the ability to trigger the snapshot from microcode by updating the microcode with a patch.”*

*“Architectural Event Tracing (AET), a debugging method, grew out of this ability of the microcode to send packets on events and certain flows.”*

*“Defeating is the ability to selectively disable specific features throughout the processor; it can greatly help with bug isolation.”*

*“The ability to observe major busses has historically been a significant part of debugging processors and processor platforms.”*

generally operating fine, suddenly ran off the rails—something that was often visible by examining a long section of the timeline or by measuring latencies based on the timestamps. In addition, because AET was based on execution of the microcode, the information exposed by AET could be extended and replaced with information targeted at specific silicon failures.

### Defeating

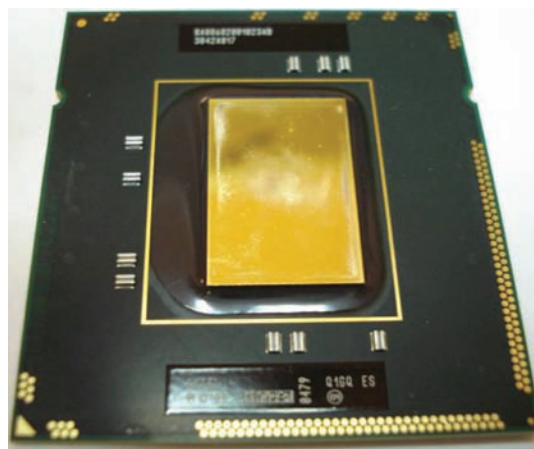
*Defeating* is the ability to selectively disable specific features throughout the processor; it can greatly help with bug isolation. Theories and hunches about the root cause of a bug can be honed by disabling features thought to contribute to a bug. Workarounds for bugs can also be quickly implemented with defeatures. Therefore, we drew up an extensive list of defeature bits in the Nehalem design.

Our choice of defeating capabilities was based on three basic principles. First, the defeature capability had to be simple. Creating a defeature that was as complex, or that was more complex than the feature itself was generally counter-productive. Second, where possible the capabilities had to include both coarse- and fine-grained options (for example, disabling branch prediction entirely or disabling only a certain type of branch prediction). Third, wherever defeature capabilities were added, the design team validated the feature with the defeature capability set during the pre-silicon design phase to ensure that the defeatures did not cause their own unique failures.

As with prior projects, we found that the defeating capabilities not only were successful in working around bugs by disabling badly performing hardware, but that they were also a key lever for bug isolation. Based on working theories of the bug, the debug team would define several experiments involving specific defeature settings that were believed to affect the bug in some measurable way. Those results often helped to prove or disprove theories about the bug and helped to focus the debug process on the likely area of the failure.

### Mirror Port for System Debug

The ability to observe major busses has historically been a significant part of debugging processors and processor platforms. The introduction of the QPI bus presented an enormous challenge to these methods and equipment. The industry-leading speed dictated that traditional resistor probing methods would degrade signals too much and thus were inadequate. Socket interposer solutions were determined to perturb electrical performance on the QPI bus as well as other busses. Solutions based on observing the signals while repeating introduced onerous latencies that would have affected system behavior. Therefore, the most expedient solution to this observability problem was to mirror all the traffic going across the QPI bus through a set of pins. Pins on the topside of the package were chosen for their accessibility and absence of impact on the critical field of pins on the bottom side of the package. This solution had the added benefit of allowing probing to be available on all platforms, because the access point was on each individual package and not hidden on the bottom side (see Figure 3).



**Figure 3:** Topside of package showing mirror port pins on the bottom and right edges.

The Nehalem product family developed a mirroring mechanism at the link layer of the QPI bus rather than at the physical layer. At this functional layer, the mirroring logic did not have to deal with the encodings for high-speed transmission in the physical layer. Additionally, mirroring at this layer was the most feasible, given routing, die-area, and ROI considerations. Because the purpose of the mirror port is to transmit data rather than electrical levels, the link layer had all the necessary information, and there was no loss of information with this strategy.

As a dedicated transmit-only interface for QPI traffic, the mirror port required a sizeable investment in silicon area and thus, the mirror port presented us with significant floor-planning challenges. However, the mirror port was found to be profoundly useful, not just for the Nehalem family of products, but also for its derivatives that used an embedded QPI bus, such as the Lynnfield and Arrandale derivatives. We solved the floor-planning problems by architecting the mirror port protocol to use the minimum number of pins while sharing layout with the QPI bus. Specifically, the protocol did not require receiving signals during initialization, which saved a significant overhead in clocking circuits. The reduced area, resulting from the link layer mirroring architecture, allowed the mirror port block to be placed in whitespace anywhere on the perimeter.

The main architectural challenges with a mirror port were initialization of the mirroring link ensuring it is alive before the main QPI links were up, while not having any impact on the initialization of the links being mirrored or on the power-up sequence of the processor. Additionally, because the mirror port is transmit only, initialization of a high-speed serial link without any handshake protocols presented additional challenges.

We met both of these challenges by using a carefully tuned timeout-based training. High-speed links, such as PCIe, generally initialize by going through multiple steps and by using a handshake to ensure the far side of the link is synchronized. The mirror port dispensed with the handshake, as the handshake requires additional circuitry that the floor plan did not have space for. However, what allowed development of this bold and risky initialization strategy was three-fold. First, the engineering teams were very familiar with QPI initialization because since 2003, we had been developing QPI-based silicon at Intel. Second, Intel would control both ends of the mirror port link. Third, the electrical distance of the mirror port link was short and thus not subject to electrical interference or degradation. This solid understanding of QPI initialization allowed reliable initialization without affecting the QPI and the processor initialization flow.

The mirror port bus consists of mirrored versions of the QPI Rx and Tx links that are selectable when there is more than one pair. The mirrored data are identical to packets observed by QPI bus agents, with the exception of power states where filler data were substituted. The mirror port architecture ensured maximum reuse of logic and circuits from QPI blocks. Mirrored data are driven through pins on the topside of the package. The pin location ensured minimal impact to the platform and minimized platform cost, since no additional package pins are needed on the bottom side nor is routing of signals

*“The Nehalem product family developed a mirroring mechanism at the link layer of the QPI bus rather than at the physical layer.”*

*“Because the purpose of the mirror port is to transmit data rather than electrical levels, the link layer had all the necessary information.”*

*“Because the mirror port is transmit only, initialization of a high-speed serial link without any handshake protocols presented additional challenges.”*

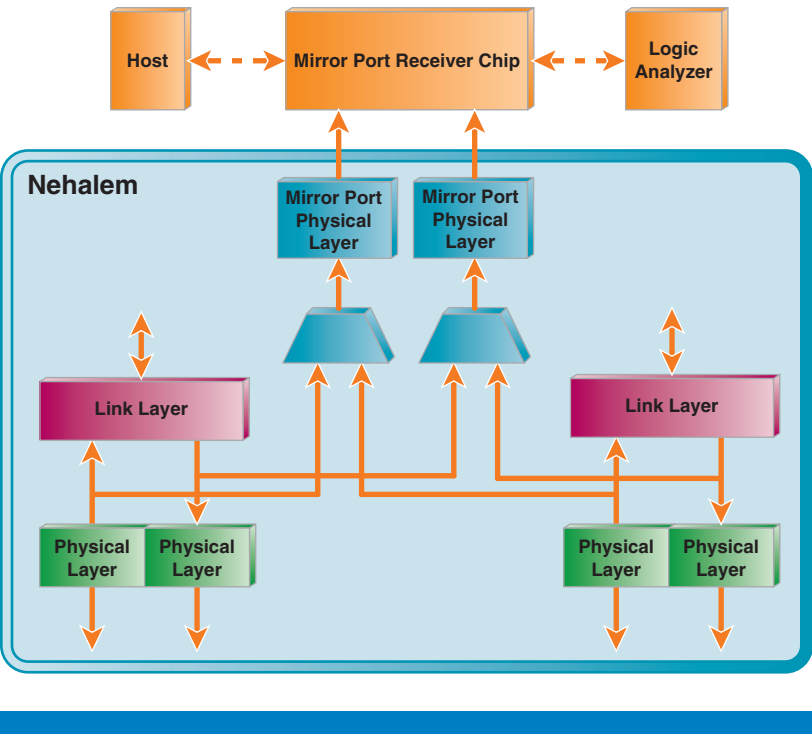
*“The mirror port bus consists of mirrored versions of the QPI Rx and Tx links that are selectable when there is more than one pair.”*

*“The mirror port data were received by an internally developed custom observability chip.”*

*“To ensure large or more complex problems could also be solved, the chip also provided an interface to a logic analyzer.”*

needed on the motherboard. Further, when the Nehalem family of products introduced integrated client products with an embedded QPI, the topside pins were the only way to observe the embedded QPI traffic. This ability allowed us to use the same set of debugging tools and software previously developed for the initial Nehalem processor. The resulting savings was significant in both engineering time and cost. Further, additional savings were realized, because engineers did not have to learn a new set of debugging tools or struggle with a new hardware interface.

The mirror port data were received by an internally developed custom observability chip. (See Figure 4 for architecture illustrating this.) The development of this chip was separate from that of the Nehalem family of products. This chip provided several services for the debugger. It replaced bulky logic analyzers for much of the debugging, which lowered lab costs. The chip functioned as a protocol analyzer allowing debuggers to trigger, sequence, count, filter, and store traces. Protocol-aware triggering, with 64 mask and match patterns and a 12-deep sequencer, along with nearly 6MB of SRAM meant that many traditional logic analyzer functions could be undertaken with a small observability pod and a host connected to a mirror port. The ability to naturally handle bus power states, where the QPI bus goes inactive, was an additional capability not traditionally provided by logic analyzers. To ensure large or more complex problems could also be solved, the chip also provided an interface to a logic analyzer over a 10’ cable. This capability provided more trace storage and better visualization facilities than internally developed solutions.



**Figure 4:** Conceptual drawing of system debug set-up showing mirror port internal pathways, top-side connection, mirror port receiver hardware



On Lynnfield, the integrated member of the Nehalem family, the mirror port was a key debugging tool. Even with QPI being embedded, keeping mirror port functionality was a priority. The benefits were immediate when coherency issues were observed during initial bring-up. A QPI mirror port trace immediately revealed an illegal transaction caused by a low-level BIOS setting. Once the BIOS setting was fixed, the system booted quickly.

### Electrical Testing and Debug of Busses

New to the Nehalem family of processors was the high-speed QPI bus running at 6.4GT/s, DDR3, and later integration of the second generation of PCIe running at 5GT/s. These were all new busses with significantly higher speeds and difficulties than previous-generation busses. The advanced 32nm process had transistors with very thin gate oxides, which make analog circuit design, particularly for I/Os, a unique challenge. In combination with this, package constraints often led to suboptimal pin placements for these high-speed busses.

During the development of Nehalem a set of on-die features collectively named Robust Electrical Unified Test (REUT) was developed to overcome these anticipated problems. REUT enabled maximum bandwidth electrical validation testing that was an improvement on previous capabilities by an order of magnitude. REUT was implemented on QPI and DDR busses. In the case of DDR, because REUT used the functional memory controller path including the schedulers, it was also used to debug and minimize DDR timing parameters and settings. Issues such as DRAM turnaround marginalities could be identified and fixed in seconds with REUT. For example, the margin of the DDR channel, when combined with marginal DIMMs, could be established by using BIOS-based REUT to establish margins where none previously existed.

As with DDR and QPI, REUT capability was built into the PCIe controller to enable EV testing and debug of the 5GT/s interface. With REUT on the PCIe bus, we were able to provide complete coverage of all the main busses on the Nehalem family.

### Survivability

Survivability, as used in this context, is the ability to work around or fix errors or late-breaking issues without needing a new stepping of the silicon. For high-volume microprocessors, the need to distribute such fixes without a new stepping is as important as the fixes themselves. Typically, microcode updates (MCUs) are the mechanism by which these fixes are delivered. With the Nehalem product family, the transition to a new platform architecture and the integration of more traditional chipset functionality onto the CPU die prompted the addition of more survivability hooks. These hooks were meant to address any late compatibility issues, and they were designed to be flexible and deliverable via an MCU. Additionally, because much of the new platform functionality in the Nehalem family was not easily accessible from the traditional microcode patching mechanisms, new capabilities were developed for working around CPU issues.

*“A QPI mirror port trace immediately revealed an illegal transaction caused by a low-level BIOS setting.”*

*“New to the Nehalem family of processors was the high-speed QPI bus running at 6.4GT/s, DDR3, and later integration of the second generation of PCIe running at 5GT/s.”*

*“As with DDR and QPI, REUT capability was built into the PCIe controller to enable EV testing and debug of the 5GT/s interface.”*

*“The transition to a new platform architecture and the integration of more traditional chipset functionality onto the CPU die prompted the addition of more survivability hooks.”*



*“Legacy platform “patch” mechanisms in chipsets relied on BIOS-based updates that involved having downstream hardware detect a problem on a transaction and then attempt to modify behavior with that transaction already in-flight in the platform interconnect.”*

*“Nehalem CPUs added a platform patch mechanism to intercept requests to platform devices at the origination point of those requests in the CPU core.”*

*“Many of the legacy mechanisms used for access between core and the Uncore employed in-band logic, meaning the mechanism might not operate correctly if the machine was in distress.”*

One important change in the Nehalem family that needed more survivability capability was the transition to the new QPI platform architecture and the repartitioning of the platform devices among the CPU and various IO hubs. As such, we added hardware mechanisms to fix compatibility issues related to changes from the platform repartitioning. Legacy platform “patch” mechanisms in chipsets relied on BIOS-based updates that involved having downstream hardware detect a problem on a transaction and then attempt to modify behavior with that transaction already in-flight in the platform interconnect. With the integration of chipset I/O features onto the Nehalem family with Lynnfield, microcode patching could be used to provide a work around.

Nehalem CPUs added a platform patch mechanism to intercept requests to platform devices at the origination point of those requests in the CPU core. This mechanism detects the address of the request *before* the request is issued to the system fabric, and it allows the CPU to provide a cleaner fix with fewer side effects than previous systems. This mechanism has the additional benefit that the fix is delivered via an MCU rather than a BIOS update, as in legacy chipsets, meaning platform patch-based fixes could be distributed via operating-system-based MCUs, and use the established infrastructure for MCU delivery. As a real-world example, we used the platform patch hardware on A0 Lynnfield samples to work around a platform incompatibility issue, where two devices claimed the same address space. Platform patch hardware was able to intercept any requests to the contested range and remap them appropriately.

Because the Nehalem family architecture was designed for more modularity with core reuse and Uncore segmentation, the legacy microcode-based mechanisms were no longer able to access logic in remote portions of the CPU die in a timely manner. With the increase in core count on the Nehalem family and beyond, the ability to communicate to logic in the Uncore or other cores for survivability purposes was becoming untenable with legacy microcode mechanisms. Additionally, many of the legacy mechanisms used for access between core and the Uncore employed in-band logic, meaning the mechanism might not operate correctly if the machine was in distress. The solution involved enabling MCUs to program internal breakpoint hardware related to core/Uncore communication. Leveraging this pre-existing debug communication hardware was a low-cost solution to the problem of how to communicate across the CPU for survivability. As this hardware was part of the internal breakpoint architecture, it allowed signaling between cores and the Uncore out-of-band, meaning it was able to operate when the CPU was in distress. Using the debug breakpoint hardware for communication gave us the additional benefit of making other debug observation hardware available to fix errata. Specifically, this hardware enabled internal transaction matching logic to be used as a building block in resolving issues: in other words, the transaction matching logic could detect transactions and signal the match all across the CPU die by using the breakpoint communication hardware. At that point, the receiving units could perform a predefined action.

However, even with an effective cross-core communication path, microcode-based survivability solutions can be inadequate for the Uncore, which contains

a large amount of complex logic, and might require complicated sequences to solve problems. As such, we provided the power control unit (PCU) the ability to access and control logic in the Uncore, leading to the ability to resolve issues outside of the realm of legacy microcode-based fixes. This access was provided out-of-band, relative to the functional structures. The fix sequences were also deliverable to the PCU via MCU, providing a very flexible mechanism to solve a wide variety of issues. This mechanism allows an agent other than a CPU core to read and react to machine state in the Uncore, and it has the added benefit of not requiring cores to be operational to implement a fix. This means that such fixes can be applied when cores are live-locked, i.e., unable to make forward progress because of an infinitely repeating beat pattern, or when all cores are in a sleep state. To further expand the solution space to dual-processor systems, an additional platform requirement was added to Nehalem family dual CPU platforms to route two spare wires between CPUs, wires that were readable and writeable by the PCU. These two wires were used to communicate between the PCUs on two different CPUs in a dual processor platform to improve package power behavior, without needing a new stepping.

On the 32nm product iteration of the Nehalem family, codename Westmere, we added a new survivability feature to allow the PCU to access a limited set of internal core states, via an out-of-band mechanism. One powerful usage model for this “PCU to Core” access feature is the ability to fix problems while cores are transitioning between active and sleep states, or when the cores are in distress, such as when a core is live-locked. Many core livelocks can be broken by simply perturbing this beat pattern. The PCU to Core feature allows an out-of-band agent (the PCU) to perturb core behavior, providing the change in behavior the core cannot provide on its own. Moreover, since the PCU fix sequences can be updated via the MCU mechanism, fixes can be tailored for a specific erratum, and the fix can be delivered in the field.

These mechanisms all found use at some point in the debugging and test phases of the Nehalem family of products. Additionally, even though the mechanisms were developed for survivability reasons, we were often pleasantly surprised at how they were used for debugging.

## HVM Test Strategy Enabling

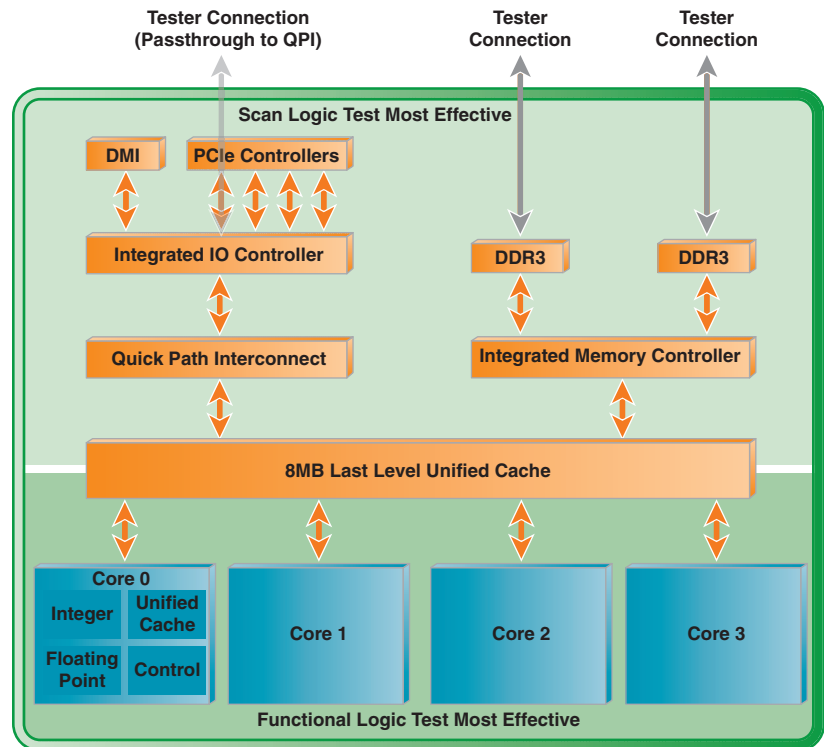
The Nehalem processor family represented a new level of integration for a high-volume mainstream Intel CPU product, bringing together four computational cores, three levels of cache, an integrated memory controller, QPI, and integrated I/O with PCIe. Integration of this wide range of features had a profound effect on HVM test coverage strategies. It required a layered approach to reach Intel quality requirements for the product cost effectively.

HVM logic testing involves a combination of scan and functional test techniques. Functional testing was very effective and efficient at covering features below the last unified cache level on Nehalem (see Figure 5), because assembly code can be preloaded into the cache and executed without

*“On the 32nm product iteration of the Nehalem family, codename Westmere, we added a new survivability feature to allow the PCU to access a limited set of internal core states, via an out-of-band mechanism.”*

*“The Nehalem processor family represented a new level of integration for a high-volume mainstream Intel CPU product, bringing together four computational cores, three levels of cache, an integrated memory controller, QPI, and integrated I/O with PCIe.”*

maintaining a deterministic interface between the Automated Test Equipment (ATE) and the PCIe, QPI, or DDR interfaces. This technique is also more naturally suited to covering micro-architectural features closest to the x86 ISA-like floating point and integer execution units, branch prediction, etc. For this reason, we relied heavily on functional test techniques to achieve coverage goals in the computational cores and memory subsystem below the unified caches. On Nehalem, functional test provided 92% stuck-at fault coverage in these areas, with only a few percent of unique scan coverage value added.



**Figure 5:** Architecture of tester bypass

However, Nehalem’s newly integrated features posed new challenges to the functional test capability. These features were far removed from the x86 ISA, instead requiring sustained amounts of memory traffic across a wide combination of request types and address decode options. This coverage also required a deterministic ATE interface with the CPU to emulate real DDR DIMMs and QPI agents that added significant debug effort above the cache-based core tests. In addition, we had to run a mini-BIOS before using the integrated memory or IO controllers. In fact, to drive reasonable fault coverage, many different mini-BIOS configurations were required to emulate different platform configurations. To keep test time low for HVM, we put significant effort into reducing and optimizing this code, which runs thousands of times in the HVM test socket. Given the challenges in these new feature areas, functional test was able to deliver 75% stuck-at fault coverage, with scan providing a much greater unique value (10%–15%).

*“To drive reasonable fault coverage, many different mini-BIOS configurations were required to emulate different platform configurations.”*

Prior to the integration of the I/O controller on Lynnfield and integration of graphics and memory controller on Arrandale, the majority of the HVM test coverage had been achieved by using QPI. Hiding this interface would have posed considerable problems for re-attaining high HVM coverage on these follow-on products. For this reason, a pass-through mode was introduced — essentially allowing data from the PCIe and DMI pins to be passed directly to QPI. Since the interfaces were exactly the same width, we needed only to add a forwarded clock pin for this strategy to be successful. With pass-through mode working, all HVM test content developed on the original Nehalem product was successfully ported to the follow-on products, with only incremental coverage required to integrate new features.

The new Nehalem family of products included a comprehensive set of DFX capabilities that built upon previous product generations. The standard IEEE JTAG 1149.1 interface provided the most fundamental level of access to all test and debug or validation hooks. Due to its multi-core design, the Nehalem family included multiple TAP finite-state-machine (FSM) controllers, which connect to the JTAG interface. The primary TAP (first in the chain, connected to the package-level JTAG pins) was in the Uncore. This TAP could be configured to either talk directly to the TDO (test data output) pin, or route its output through any or all of the core TAPs in the die in parallel or series. Each of these core TAPs, in turn, provided full access to the embedded DFX hooks (test mechanisms, control, observation, triggering, debug data dump, etc.) buried within its respective portion of the die.

The package-level JTAG pins provide a low-speed (kHz-MHz range) serial sideband interface into the die, which is critical in system debugging, but is generally not enough bandwidth for either debug data dump or HVM test delivery. For production test use and for component debugging access on a high-speed tester, the primary TAP was used to configure the main QPI interface of the die into a test-only parallel access mode. It ran at up to an effective bandwidth of 2GB/s. This mode was configurable to access either the Uncore or the many cores, and to provide test content to all cores in parallel. This test interface provided a very flexible and powerful access mechanism for HVM test.

When later models of the Nehalem family introduced the integration of PCIe and buried the QPI interface deeper into the product, a new access mode was added to pass test data from the PCIe interface through to the internal QPI boundary. With this additional feature, all legacy core and Uncore test content that had been developed on initial versions of the product family could be simply re-used across all new versions of the product. Enabling this feature was challenging, due to the complexity of passing data from one very high-speed domain to another while maintaining deterministic behavior for the tester. However, this proved to be the key feature for enabling uniform structural debug access and HVM test content delivery across the entire family from high-end products all the way down to desktop and mobile SKUs.

*“A pass-through mode was introduced —essentially allowing data from the PCIe and DMI pins to be passed directly to QPI.”*

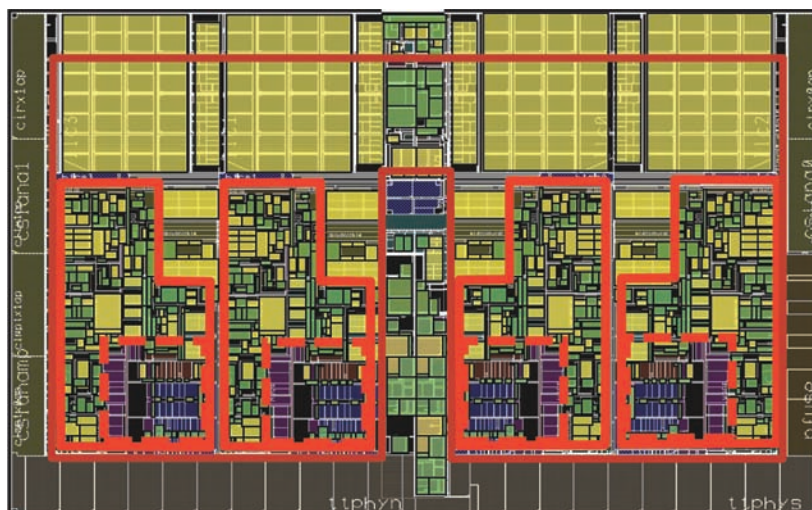
*“Due to its multi-core design, the Nehalem family included multiple TAP finite-state-machine controllers, which connect to the JTAG interface.”*

*“The primary TAP was used to configure the main QPI interface of the die into a test-only parallel access mode.”*

*“When later models of the Nehalem family introduced the integration of PCIe and buried the QPI interface deeper into the product, a new access mode was added to pass test data from the PCIe interface through to the internal QPI boundary.”*

*“These combined test features enabled fault grade coverage targets in the mid 90% range.”*

The main test features within each component of the design included extensive Array DFT, partial Scan design, pre-loadable functional test modes running out of various levels of cache, analog circuit trim overrides, internal signature mechanisms for test result accumulation, and a triggering system for starting or stopping all of the mechanisms synchronously on internal system events. The Array DFT mechanism included programmable built-in self test (BIST) controllers in each major section of the die, which could be used to run tests in major arrays in parallel. The partial scan design was focused primarily on sections of random logic, where timing and area impact would be minimal. The functional test modes were comprehensive in their coverage, and they supported pre-loading of functional patterns from a structural tester, with the patterns architected in such a way as to avoid all external access to memory and relying on internal signature registers for results. These combined test features enabled fault grade coverage targets in the mid 90% range. This high fault grade number allowed aggressive DPM and test time targets to be achieved for the product family. A representative diagram is shown in Figure 6. It illustrates the careful overlay of structural and functional test technologies in the die.



**Figure 6:** Die plan showing overlay of HVM Test features (array = yellow, scan = green, on-die functional = red outlines, grey = IO, tester functional = whole die)

*“Built-in I/O pattern engines were also designed for extensive system debug capability.”*

Built-in IO test capabilities in this product family were extensive. They included on-die programmable pattern engines and both analog and digital loopback modes to fully exercise the IO circuitry when it was connected or not connected to a tester outfitted with very high-speed pin electronics cards. The built-in I/O pattern engines were also designed for extensive system debug capability, allowing signal characterization and signal integrity measurement in system platforms. These capabilities were critical for debug of the new high-speed IO interfaces in the product family.



## Summary and Conclusion

Among the most problematic developments for post-silicon validation and HVM of the Nehalem processor was the high integration and the introduction of the QPI high-speed serial bus. In previous processor generations, validation and manufacturing functions had undergone only incremental improvements. The Nehalem family demanded a rethinking and re-engineering of many validation and manufacturing paradigms.

The leap in speed and protocol embodied in the QPI bus demanded a matching observability solution. We developed the mirror port as a non-perturbing observability agent that provided a similar point of observability as the previously familiar FSB. This turned into a large saving of effort when the even more highly integrated client products, Lynnfield and Arrandale, appeared as the mirror port provided the same point of observability even though the QPI bus was buried in the design.

This high integration precipitated additional innovations in both HVM and validation. In HVM, the TAP architecture was fundamentally changed, and pass-through modes had to be devised to allow reuse of test content in the same way the highly integrated architecture was reusing product designs. In the validation space, the high integration drove extensions of existing techniques, such as snapshot debug, and also new innovations, such as AET. While snapshot debug accelerated the time to determine the root cause of bugs, AET aided debugging when it involved BIOS, drivers, or software. The development of REUT, on the other hand, was used to counter the difficulty of isolating electrical and signaling issues with the new high-speed busses.

With future demand for more advanced CPUs and platforms with more features, high-speed interfaces, and integration of system logic, the continued development of the technologies described in this article that were developed for the Nehalem product family will remain essential.

## References

- [1] Stewart, Animashaun. "Top Side High Speed CMT Tester Interface Solution for Gainestown Processor," *IEEE International Test Conference*, 2008.
- [2] *Intel 64 and IA-32 Architecture Software Developer's Manual*, Volume 2A, 2B. s.l. Intel, March 2010.
- [3] Singhal, Ronak. "Inside Intel Next Generation Nehalem Microarchitecture," *Intel Developer Forum*, Spring 2008.
- [4] Fleckenstein, Chuck; Zeisset, Stephan. "Post-silicon Memory Consistency validation of Intel Processors," *9th International Workshop on Microprocessor Test and Verification*, MTV December, 2008.

*"Among the most problematic developments for post-silicon validation and HVM of the Nehalem processor was the high integration and the introduction of the QPI high-speed serial bus."*

*"We developed the mirror port as a non-perturbing observability agent that provided a similar point of observability as the previously familiar FSB."*

- [5] Amitabha Roy, Stephan Zeisset, Charles J. Fleckenstein, John C. Huang. “Fast and Generalized Polynomial Time Memory Consistency Verification,” *CAV* 2006, pp. 503-516.
- [6] Intel VTune™. Available at <http://software.intel.com/en-us/intel-vtune/>
- [7] Horvath, T.A.; Kreitzer, N.H. “Hardware Emulation of VLSI designs,” *ASIC Seminar and Exhibit*, 1990. In *Proceedings, 3rd Annual IEEE*, P13/6.1-P13/6.4.
- [8] Ganapathy, G.; Narayan, R.; et al. “Hardware emulation for functional verification of K5.” In *Design Automation Conference Proceedings*, 1996, June, pp. 315-318.
- [9] Wagner, Nicholas et al. “Reproducing Post-Silicon Failures in an Emulation Environment to Enable System System Validation,” Internal, *Intel Design and Test Technology Conference*, 2008.

## Authors' Biographies

**Derek Feltham** is a Principal Engineer at Intel Corporation. He is a corporate expert on Design for Test and Design for Debug. He has led architecture development and design and validation of DFT/DFD features through several generations of the Intel® Pentium® family of processors and Intel Core processors. He currently serves as the Advance Test Methods Development Manager in the Intel Technology and Manufacturing Group. Derek received his PhD degree in Electrical and Computer Engineering from Carnegie Melon University.

**Tim Callahan** has been at Intel for 11 years. His primary areas of expertise are functional test Design for Test, Reset for Test, and high-volume manufacturing test content readiness.

**Hermann Gartler** is a Principal Engineer in the Microprocessor Development Group Architecture team. Hermann holds a BS degree in Electrical Engineering and an M.E.E degree, both from Rice University. He joined Intel in 1996 and has worked as a cache controller architect, a debug architect, and is now involved in silicon debug efforts. His email is hermann.gartler at intel.com.

**Lily Pao Looi** is a Principal Engineer at Intel Corporation architecting desktop and laptop microprocessors. She joined Intel in 1990 and has held various lead roles in architecture, design, and performance. She also has extensive experience in silicon design, performance analysis, and debug for various products including server chipsets, supercomputer components, storage products, and flash memory. She received her engineering degree from the University of Michigan. Lily holds 18 patents.



**Keshavan Tiruvallur** is a Chief Validation Technologist at Intel. He is a Senior Principal Engineer working in the post-Si validation group acting as the technical oversight on debug processes and methods. He is currently leading the enhance platform debug capabilities effort (Hotham) within Intel.

**Robert Mauri** is a Principal Engineer in the Platform Validation and Enabling Department. His current focus is with OEM customer platform integration tools and processes on server products. He holds BSEE and MSEE degrees in Computer Engineering from the University of Southern California. His email is Robert.mauri at intel.com.

**Colin Looi** is a Senior Staff Architect and has been working at Intel for 22 years designing and processors and chipsets. He is currently planning and architecting debugging solutions and tools. He has a Masters of EE from Cornell University and a Masters of International Management from Portland State University.

**Chuck Fleckenstein** is a Principal Engineer in Intel's Platform Validation Engineering Group. He has worked as a post silicon validation architect for Intel IA-32 Processors, Intel Itanium Processor Family, and previously was an operating system engineer for Intel's Supercomputing Systems Division. His interests include post-silicon coverage analysis, validation methodology and tools, as well as Design for Coverage hooks.

He joined Intel Corporation in 1993. He received a M.S. degree in Computer Science from Wright State University.

**Michael St. Clair** is a validation engineer at Intel.

**Bryan Spry** is a Senior Staff Architect at Intel and has working for the past 14 years in CPU design, validation, and architecture. Past areas of focus includes Frontside Bus, cache controllers, integrated memory controllers, and most recently PCI Express.

## Copyright

Copyright © 2011 Intel Corporation. All rights reserved.

Intel, the Intel logo, and Intel Atom are trademarks of Intel Corporation in the U.S. and other countries.

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

Requires an Intel® HT Technology enabled system, check with your PC manufacturer. Performance will vary depending on the specific hardware and software used. Not available on Intel® Core™ i5-750. For more information including details on which processors support HT Technology, visit <http://www.intel.com/info/hyperthreading>

Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM). Functionality, performance or other benefits will vary depending on hardware and software configurations. Software applications may not be compatible with all operating systems. Consult your PC manufacturer. For more information, visit <http://www.intel.com/go/virtualization>

Requires a system with Intel® Turbo Boost Technology capability. Consult your PC manufacturer. Performance varies depending on hardware, software and system configuration. For more information, visit <http://www.intel.com/technology/turboboost>

# THE TOOLBOX FOR HIGH-PERFORMANCE CPU DESIGN

## Contributors

**Wayne Clift**

Intel Corporation

**Clark Brooks**

Intel Corporation

**Gohar Waqar**

Intel Corporation

**Vijay Kodithyala**

Intel Corporation

**Sean Mirkes**

Intel Corporation

**Kami Huck**

Intel Corporation

## Index Words

Register Transfer Level

Formal Equivalence

## Abstract

This article reviews the myriad tool and methodology innovations that went into the CPU design system used to develop the the *Intel® microarchitecture code name Nehalem* and *Intel® microarchitecture code name Westmere* products at Intel. We present design tool and flow innovations from the beginning of design to the final production of manufacturing masks. Additionally, we describe enhanced techniques for moving a CPU design to a new manufacturing technology, as well as methods for managing multiple product configurations, all of which were key enablers for delivering the Intel® Core i7 processor in record time.

## Introduction

The Intel® Core i7 processor contains twice as many transistors as the previous Intel Pentium® 4 processor, while delivering significantly increased computational capability, a wide range of new features, and a 50% reduction in power consumption. This increased transistor count is enabled by Moore's law, which predicts an exponential increase in the number of transistors available to a design team for a given cost. However, this increase in transistors comes with an increase in complexity across multiple fronts, including the overall size of the feature set, the validation of each feature and its interaction with other features, the physical constraints of an ever-shrinking transistor, and the need to deliver products at a faster cadence. In this article we do *not* describe how the Core i7 design team used extra transistors to design new features. Instead, we look at how some members of the design team focused on reducing the complexity of the design process itself.

We first present new technology and innovations, ranging from initial design creation to final production. Each advancement enabled significant complexity reduction without sacrificing engineering flexibility or product requirements.

## Logic Design and Validation

The first task in the implementation of a new processor architecture is to convert the architectural specification into a simulation model. This reference model contains all design features, is used to validate architectural correctness, and is then compared to final schematics.

## RTL Design Specification

A Register Transfer Level (RTL) software model is created to simulate the functionality of the final design. Traditional RTL models contain massive amounts of detail, require a long time to develop, and are difficult to maintain and validate. To manage the projected growth in RTL size of the Core i7, we needed to develop new software abstraction capabilities. Current internal and external design languages were inadequate for our needs; therefore, our Core i7 RTL methodology team joined with industry leaders to define and develop what would become the IEEE SystemVerilog (SV) language [1]. SV's use of advanced datatypes, functions, macros, parameterized modules, and behavioral logic descriptions reduced the RTL simulation model by about a third to a half its original size. Figure 1 demonstrates the reversal in the decade-old trend in RTL simulation model size.

Beyond the capabilities enabled by raw language syntax, the Nehalem (codename) RTL Design System utilized a series of internally developed code generators for the following:

1. Automated creation of hierarchy and module connectivity files.
2. Logic decoder minimization.
3. Finite state machines.
4. State/buffer signal repeater insertion.
5. Physical netlist creation.

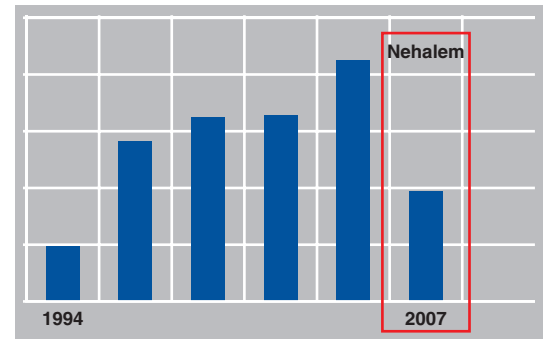
These code generators handled much of the tedious RTL work, allowing designers to focus on design challenges.

In the Nehalem RTL Design System we also made extensive use of simulation assertion checkers that monitor validation tests for design assumption violations, leading to efficient identification of logic bugs. These assertions were later proven to be true through a formal property verification tool.

The structure and management of the RTL source code repository also contributed to design productivity. Careful repository organization and a fully automated incremental model-build and regression suite allowed new features to be incorporated into mainline RTL models without having a negative impact on existing model functionality. By utilizing these tools, we were able to concentrate our effort on creating new features, not debugging poor-quality models.

## RTL Design Validation

While developing the RTL model, the logic design team performed an in-depth analysis of its correctness, a process known as RTL validation. In order to keep pace with the increase in overall design complexity, a new hierarchical testing infrastructure and other validation practices would be needed to deliver the required debug efficiency to complete the task on schedule.



**Figure 1:** Lines of RTL code across major Intel® processor designs

*“The engineers writing and validating RTL code then had a powerful testing infrastructure that reduced turnaround time for finding and fixing bugs.”*

We developed Multiple Test Environments (TEs) to independently validate subsections of the design. We used smaller, faster TEs to test localized features and a larger top-level TE to validate the full architecture. The engineers writing and validating RTL code then had a powerful testing infrastructure that reduced turnaround time for finding and fixing bugs. The benefit of the new TEs was also evident when we faced parallel validation efforts for multiple products, as well as concurrent support of post-silicon debug work—an activity that was impossible with previous tools and methods. Given these results, these validation technologies have been applied to subsequent major CPU design projects at Intel.

To further handle the increased validation complexity, we employed new validation skills. We used Analog Mixed-Signal (AMS) validation to compare new analog circuitry, by using a traditional digital RTL model. Additionally, we employed mathematically-rigorous “formal proofs” to guarantee the correctness of some parts of the logic rather than validating it by testing many example calculations. We also used formal proof techniques to validate 25% of the entire design core, greatly reducing the risk of logic bug escapes. These formal validation methods used on the Intel Core i7 processor proved the value of a new way to conduct validation efforts on future CPU designs.

### **RTL Design Formal Equivalence**

The RTL model is a logical specification of the design to be implemented in silicon. However, the RTL model does not contain any physical gate- or transistor-level circuit details; these are represented elsewhere in circuit schematics. To ensure that these circuit schematics and the RTL model match, a formal equality check, known as a Formal Equivalence Verification (FEV), must be performed between these two different models.

Commonly available FEV technology had a severe drawback. It often forced the degeneration of simple, abstract RTL logic expressions into detailed physical representations, due to the fact that logical registers (state) in the RTL were required to be represented exactly in the circuit schematics. This restriction historically caused an increase in overall lines of RTL code as well as an increase in logic bugs, since RTL code is modified for circuit implementation details.

*“simpler, functional descriptions of logic and register state could be proven equivalent to much more complex circuit designs.”*

The Core i7 RTL methodology team pursued new internally-developed sequential equivalence technology that enabled non-state-matching (NSM) design equivalence checking, called SEQVER [2]. NSM design meant that simpler, functional descriptions of logic and register state could be proven equivalent to much more complex circuit designs. While NSM removed the need to reduce RTL code abstraction, it simultaneously allowed for an increase in abstraction in a significant number of cases. For example, parameterized register file (RF) design garnered the biggest benefit in increased abstraction, allowing a small library of generic RTL array modules to characterize 80% of physical RFs.

By applying SEQVER technology to the Core i7 RTL, simple RTL descriptions were proven equal to complex, sequentially retimed schematics. Late code re-writes, which often injected new bugs, were avoided, and healthy, stable RTL led to timely completion of a more complex design.

## Circuit Design

There are about one billion transistors, and even more wires, on a Core i7 processor. These transistors are divided into hundreds of functional blocks (FUBs), so that the blocks are small enough for a designer to understand and optimize. Some FUBs are devoted to the local memory registers that store the current state of the processor, some FUBs calculate arithmetic, and some FUBs keep track of which calculation or memory access is needed to execute the currently-running software. FUB design for the Core i7 was a major focus for the team, and it is another area in which complexity and effort would have reached unacceptable levels if no innovation occurred.

Since the first Intel Pentium processor was designed, performance was delivered with hand-tuned transistor circuitry in critical portions of the chip, and 75% of the transistors were designed in this way. The high cost of such manual effort has been amortized across a high sales volume; the Pentium 4 family sold more than 500 million units. An alternative design method, Cell Based Design (CBD), has long delivered low-performance circuitry with much better design-effort efficiency, and this method is used throughout the industry. Historically, in Intel processor design, about one-quarter of the transistors on a processor have been designed by using Random Logic Synthesis (RLS), which uses a CBD approach. RLS software translates the logical specification of the block's function into standard logic cells, places the cells in a region of the chip, and generates wiring automatically. RLS is very efficient from the point of view of effort, but it does not deliver the highest-performing silicon, particularly for the arithmetic and memory FUBs.

The Core i7 design automation team, along with key circuit design experts in multiple domains, came together to define a way that CBD could be enhanced to deliver high-performance circuitry and simultaneously maintain CBDs low effort. To succeed, automation tools would have to change, and design engineers would need to be convinced that they could achieve the required performance targets.

## Transistor-level Optimization through Cell Design

Historically, engineers built circuits from small groups of transistors, tuned and placed them tightly on the silicon, then assembled these groups as tightly as possible to complete the FUB. This design approach had to be replaced with a CBD approach that still delivered high performance. Virtually every transistor on the Core i7 came from a library of standard cells that was tuned and laid out once. Those standard cells, such as latches, inverters, and NAND gates, were re-used up to a quarter million times each. This library of standard cells differed from any previously used by an Intel team in that it included

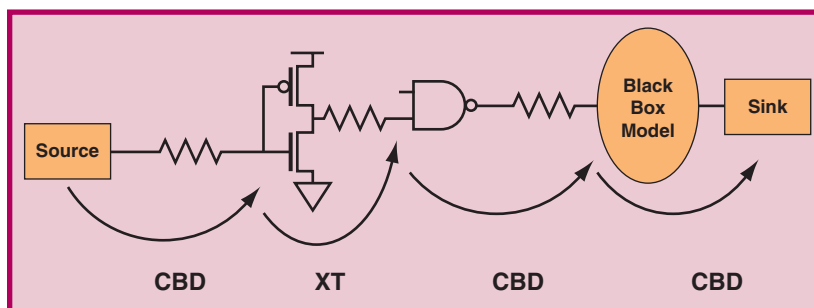
*“There are about one billion transistors, and even more wires, on a Core i7 processor.”*

*“Virtually every transistor on the Core i7 came from a library of standard cells that was tuned and laid out once.”*

*“This mixed analysis capability provided us a way to optimize circuits as they did in the past, with a significant decrease in effort and complexity.”*

a large variety of high-performance and special-use cells. These special cells, not strictly necessary to implement the logical functions, enabled the logical functions to perform at the level needed for a lead microprocessor. By having the FUB designer’s critical circuits built by an expert library team according to a set of CBD-compliant standards, the benefits of CBD with the high-performance requirements needed for CPUs were met at the cell level.

A first CBD standard was the use of a regular grid of placement locations for standard cells, similar to how city blocks are laid out in a grid. This ensured that each cell and each FUB fit well with its neighbors and had adequate power delivery. A second CBD standard required that cells must contain a complete drain-connected network (DCN) to simplify electrical analysis. Although it was relatively easy to move designs based on static CMOS logic (structured data path, or SDP) into a CBD domain, other common circuits, such as register files (RFs), read-only memories (ROMs), and FUBs containing a mixture of analog and digital circuitry, could not be designed with such restrictions due to their distributed DCN nature. We worked to define new analysis tools that could seamlessly handle a mix of CBD and transistor level (XT) analysis, while still leveraging the other benefits of CBD standardization. All tools from design entry (schematics, layout) to timing (Figure 2), power optimization, electrical rule checks, and noise analysis were enhanced to handle mixed CBD and XT designs, by examining the circuit and deciding which cells could be CBD and which cells needed XT analysis. This mixed analysis capability provided us a way to optimize circuits as they did in the past, with a significant decrease in effort and complexity.



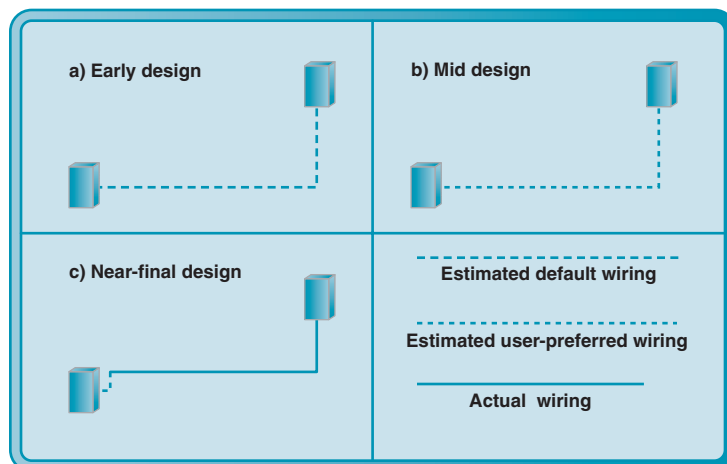
**Figure 2:** Mixture of CBD and XT design—Circuit Analysis stages

### Wire Design

With a new set of tools to manage complexity and effort for cell-level design, the team moved their investigations to the wiring between cells within the FUB. The key insight was that managing and verifying wiring was a long loop, because the wiring needed to be complete before it could be verified. Typically teams have used one set of performance evaluation tools for pre-layout analysis and another set for post-layout analysis. One breakthrough that was delivered on the

Core i7 design tools was calculating the performance effects of cell placement and wiring on each part of the circuit without waiting for the placement and wiring to be complete. By doing this, the circuit designer knew much earlier in the process whether the silicon implementation would work as fast as the circuit schematics simulated. The designer, therefore, could focus on optimizing the most important portions of the circuit, without being distracted by having to complete the less critical portions. We developed a new extraction paradigm that allowed the designer to control the specification for wires in the design, while allowing the merging of a partially completed circuit layout into the FUB analysis. Furthermore, we enhanced the wiring tools to provide CBD automation benefits that exist on RLS flows, within the context of the high-performance circuit wiring that is needed across the chip. We expanded wiring tool capabilities to deliver more predictable, higher-performance connections between circuits, and we enabled manual overrides in cases in which a skilled designer could create a better solution than the one created by the software. By combining these techniques, the designer was able to get the highest-performing transistors and wiring where they were needed, and was able to automate creation and estimation of the rest of the design, thus saving significant effort (Figure 3).

*“We developed a new extraction paradigm that allowed the designer to control the specification for wires in the design, while allowing the merging of a partially completed circuit layout into the FUB analysis.”*



**Figure 3:** Route Estimation/Extraction methods over time

## Layout Design

Layout design is the process of converting CBD and transistor-level schematics into a physical description of the dimensions and orientation of each layer of material deposited on silicon. The dimensions of transistors, the routing of wire connections, widths, and spacing between devices and wires are all carefully controlled by using a complex set of process design rules (DRs) that must be followed precisely. New complex design rules in sub-50nm process nodes made the design of power and signal wiring difficult. Efficiently producing high-quality layout for performance-critical CPU circuitry has been historically



time-consuming, so the Core i7 layout methodology team developed layout assembly and construction techniques that reduced the complexity of DRs as well as significantly improved productivity.

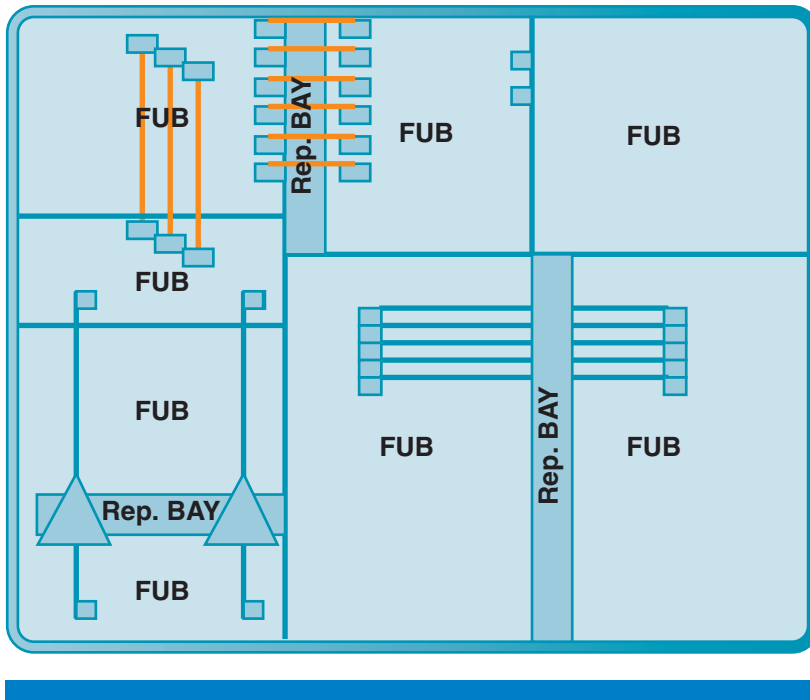
To account for the expanding complexity in the process DRs, we defined a power grid and metal wire pitches for every metal layer and the transition between layers. This plan took into account the needs of power distribution and wire signaling performance in the context of CBD standard library cells. In addition, multiple DR-clean wire patterns were available for the designer to provide them the needed flexibility in circuit design. This basic design is called a tile, and it is arrayed over an entire FUB. FUB cell placement and routing implement the layout according to the restrictions of the tile definition. Because this repeatable tile of power and signal tracks is constructed to account for complex DRs, it automatically prevents nearly all design rule violations (DRVs) for metals.

Several layout automation tools were made much simpler, despite the increasingly complex DRs, by using tiles. Because the tile was defined as a multiple of the library cell height, tile-based design simplified auto cell placement tools, as well as cell check tools that verify manually placed cells. To optimize wiring, the tile was used to define route guides, and DR-clean location wires could exist within a tile, which simplified both manual and auto routing tools.

Additionally, to assist with later assembly of multiple FUBs next to each other, we developed the new concept of a FUB “Halo”. A FUB Halo is a predefined cell-based FUB border that ensures that no DRVs occur when it abuts other FUB layout. No FUB was reopened to fix DRVs inside FUBs during integration. Halos enabled correct-by-construction FUB integration.

## Full Chip Partitioning and Integration

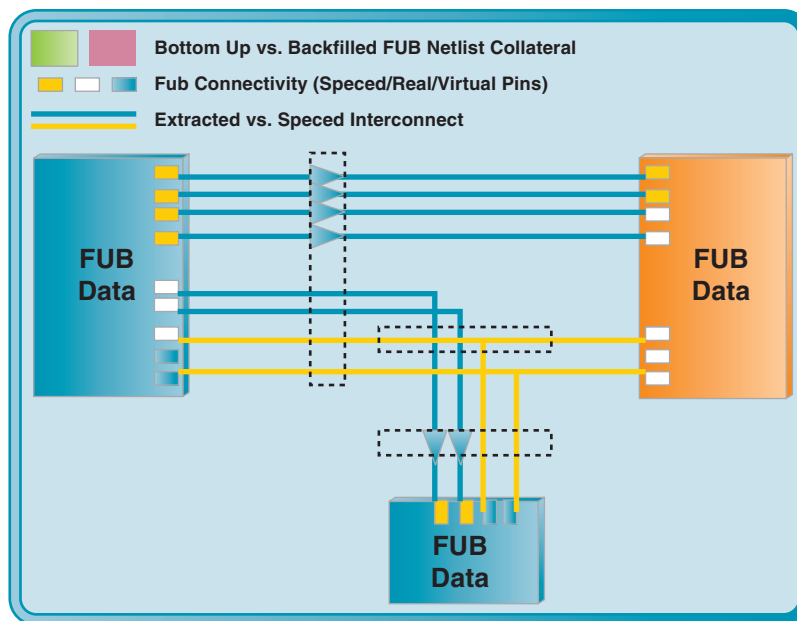
Once we addressed FUB design complexity, we took steps to optimize the design between FUBs. The Core i7 Full Chip (FC) model is a hierarchical integration of FUBs connected by wires. This model also contained special top-level “FUBs”, called Repeater Bays, that contain circuitry to cleanly transmit signals over relatively long distances inside the chip (Figure 4). Each FUB provides an abstract, black-box model for timing, noise, and layout to the FC model for integration. The FC model stitches together FUBs, wires, and repeater bays, and it provides the designer with performance targets between FUBs that can be verified.



**Figure 4:** Full Chip integration view of the Core i7 design

### Full Chip Early Timing Integration

Minimizing the Core i7's overall product design time required maximizing parallelism on multiple design vectors. Early FC design requires the ability to execute in parallel to both the RTL and FUB implementation, and it therefore needs tools and flows to support missing and/or late input collateral. This parallelism was achieved by developing a model in which the logic specified in the RTL could be decoupled from the physical netlist, which forms the basis for the interconnect between FUBs. By having a decoupled model, engineers could commit known connectivity to the FC before a single line of RTL was written, giving FC the ability to work in parallel and resolve issues with wire planning, FUB placement, circuit frequency, and electrical robustness before the RTL or any of the FUBs had completed designs (Figure 5). In order to keep these two models from diverging, we wrote software to flag quality issues and to provide an easy mechanism to review differences and synchronize changes as they occurred.



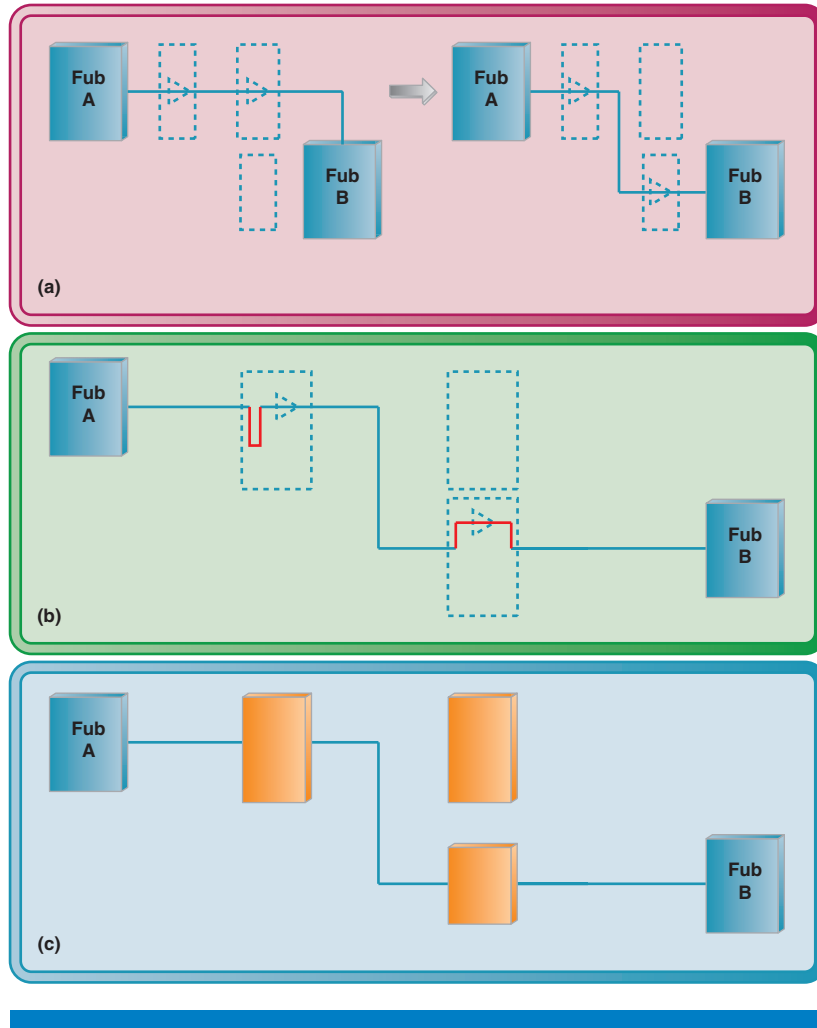
**Figure 5:** Full Chip timing model integrating data with various levels of completeness

### Full Chip Interconnect Convergence

Even after the netlist was decoupled from the RTL, the interconnect still needed to be designed. The Core i7 contains multiple metal layers, each with different performance and density properties. Early in the design process, the FC model undergoes significant change as design progresses, and thus it is important to minimize both the effort to implement the interconnect and the effort to modify existing interconnects. Because of the high wire count on the Core i7 and the fact that each wire could require multiple repeaters, creation and modification of repeaters was a key area of development. Virtual Repeaters (VRs) simplified repeater planning and analysis by representing repeaters as a virtual property on a wire, which had location coordinates and electrical properties but did not modify the actual netlist or wires. The simplicity of VRs provided significant flexibility when dealing with a changing FC model (Figure 6a). To achieve these goals, the planning, extraction and timing analysis tools all had to be enhanced to support VRs. We enhanced the automated engine for repeater insertion to handle VRs, and we were able to deliver over 80% of the repeater design for the product. Experts manually designed the remaining 20%, and once an acceptable solution was achieved, the result was stored with the virtual representation.

Late in the design phase, the VRs must become real transistors and interconnect into the design. Fracture is the process by which the wire is physically broken at the repeater location, both in layout and in the netlist, and segments are connected to the real repeater. This process provided early feedback on circuit connectivity problems. (Figure 6b). We developed additional automation

for relocating VRs to achieve better routability and power delivery. The team could convert the layout database from the virtual to fractured state just four weeks before the first tape-out (Figure 6c), with significantly less manual effort compared to that required for previous Intel CPU projects.



**Figure 6:** Repeaters and Interconnect Design phases

### Full Chip Layout Integration

The tile concept that was applied to FUBs extended naturally to the FC model. Because tiles were used for all FUBs, FC assembly was DR correct-by-construction, and all FUBs perfectly matched power grids, allowable wire locations, and FUB sizes. Empty spaces that were left over in sections, primarily used for FC interconnect routing, were likewise built by using the tile concepts to ease automation and assembly into the FC model.

Some DRVs can only be detected in the FC model, where the area being checked is large enough. Careful consideration of DRs, such as via and metal

*“A tool regression system was implemented to achieve a high standard of quality and to reduce the manual testing effort.”*

*“Once a new-generation CPU is designed, a design team focuses on smaller optimizations and takes advantage of Moore’s Law.”*

density, and proper layout planning of tile structure during the initial tile design helped us avoid occurrence of such DRVs after assembly of blocks at the FC level. This resulted in complexity reduction and effort savings over previous processor design generations, in terms of person-hours spent cleaning up density DRVs.

We developed a new single solution for density checking, one that is robust and independent of floorplan changes (i.e., products containing 2 cores vs. 6 cores, etc). Our solution enabled cleanup of FC density violations in FUBs and sections.

By organizing verification flows into relevant bundles we were able to communicate with the project leaders during execution of each phase of the design. A tool regression system was implemented to achieve a high standard of quality and to reduce the manual testing effort.

A mock integration of the FC layout, synchronized to design execution phases and done with well-defined expectations, enabled us to fine tune the integration tools, FC-level checks, OPC, fracture tools, and related handoff processes. Optimized FC flows and customization of hardware and software configuration kept the turnaround time from last ECO to tape-in to 24 hours.

Early engagement with external teams, co-development of process collateral, such as etch rings and marker cells, and the generation of project-specific quality checks, above and beyond those required by the manufacturing process team, resulted in the Core i7 family of products achieving fully functional silicon from the very first silicon produced.

An internal tool developed to leverage cell-based design, helped us early on to identify and prevent DCC issues on SDP FUBs at the Place and Route stage—much earlier detection than in previous generations of processors. Very few DCC problems were discovered later in the design, because of robust, top-down, power-grid planning and because of the use of the internal tool to identify early DCC issues on SDP FUBs.

An internal tool used for huge section-level RV runs, perfected during the development of the Nehalem technology, was also leveraged to run RV on huge analog FUBs with a divide-and-conquer approach. Basically, the first RV currents were generated for analog blocks and components by using simulations and then by stitching the blocks through the internal tool.

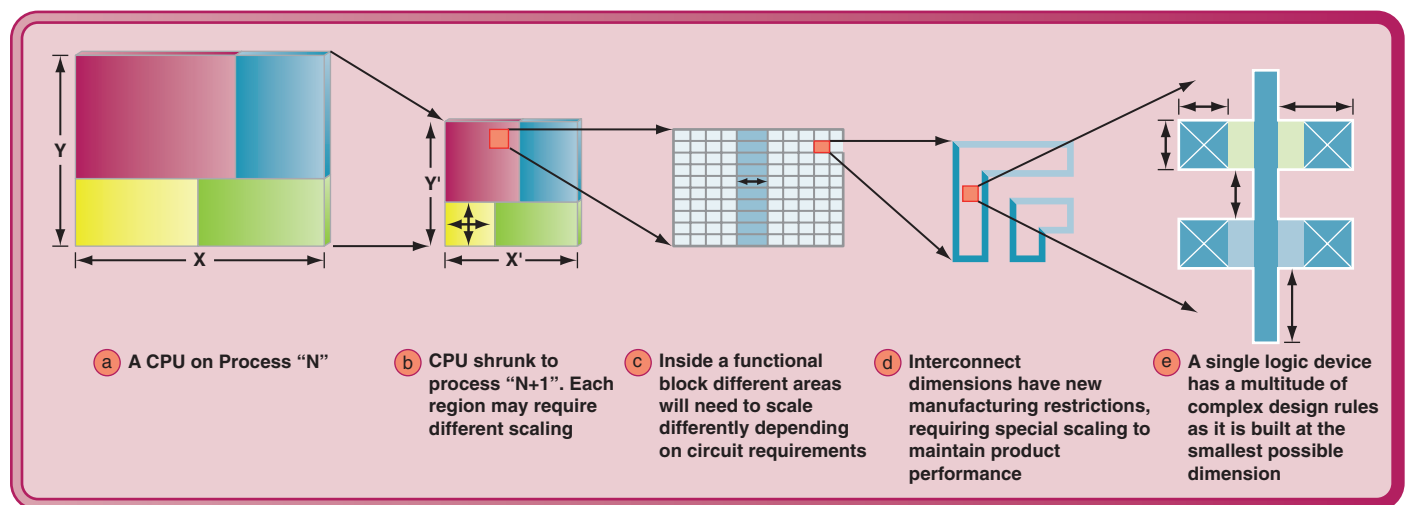
## Design Process Shift

Once a new-generation CPU is designed, a design team focuses on smaller optimizations and takes advantage of Moore’s Law. A Design Process Shift (Pshift) is the name for a set of tools and methodologies, co-optimized with the process definition team, which allows for a CPU to shrink the basic physical dimension of the transistors and wires by a significant factor. Historically, this is 0.7x multiplier in each dimension (Figures 7A and 7B), which results in the same CPU functionality in half the area, as well as in a significant power reduction. Over the past two decades, scaling to this degree has become much

more difficult, as transistors get smaller and smaller. In the early 1990's a Pshift was simply an "optical shrink" where every transistor dimension was scaled to the same degree. In the late 1990's, optical shrinks were no longer possible. Designers used Pshifts to analyze each individual transistor dimension, and software compensated or adjusted for the portions that could not be scaled.

A fundamental challenge to scaling a processor to a new deep sub-micron technology is how to address the increasingly complex process DRs and simultaneously maintain the engineer's design efficiency, chip density, product performance, and power requirements. For a design that contains a wide variety of circuit styles and types, these challenges are even greater. When planning Pshift for the Nehalem family of products, the Pshift team investigated multiple areas related to cell and interconnect scaling. The final Pshift technology was a blend of new and old, and it allowed the family of products, codename Westmere, to deliver and surpass its project goals.

The traditional approach to Pshifting a design by analyzing each transistor or wire in every mask, writing algorithms to adjust these to a new set of finer dimensions, and complying with a more complex set of design rules was reaching its limits (Figures 7D and 7E). The results were less optimal circuitry and layout to begin with, which then required manual effort to improve. Pshift had reached a point where traditional methods were no longer providing a good enough result. The goal for us was to develop a way to rapidly prototype different algorithms and approaches to simultaneously optimize the product, something not possible in prior generations due to the customization of the entire physical design. We set out to migrate key design constraints from the layout while applying the benefits of tile-based layout construction methods used in designing the original Core i7 processor.



**Figure 7:** Scaling a CPU to a new process, shown at different levels of details

*“Because of the standardized wire patterns inherent in the definition of tile-based layout construction, the chip could be broken down into four main regions and analyzed separately for scaling.”*

*“With an easily scalable design starting point, we achieved the fastest time to market for a deep sub-micron CPU project in Intel history,”*

Because the Nehalem family of products uses a highly optimized cell library and also contains custom cells for specific critical functions, it was possible to complete scaling studies very rapidly by using cell abstracts. A cell abstract is a simple high-level model of the cell that contains only basic information about cell size. The impact of cell architecture and circuit design choices could then be studied. As designs were run through this abstract analysis flow, the Pshift team was able to understand which cells were critical to overall design scaling and were able to refine those estimates or modify the circuit topologies to maximize overall benefits. Later, when cell layout was more accurate, additional experiments further refined the library cell definition.

Because of the standardized wire patterns inherent in the definition of tile-based layout construction, the chip could be broken down into four main regions and analyzed separately for scaling. Designing a Pshift for wires has enormous challenges because of the delta between the wavelength of light being used to create the manufacturing masks and the actual physical dimensions of the resultant mask. At the same time, wire choices impact the ultimate performance of the part and must be carefully managed.

Our approach to designing the Pshift was based on identifying regular structures aligned to a vertical or horizontal ‘strands’ (Figure 7C). We defined the strands based on tile-based layout construction, which allowed us to similarly scale each section of the design. Although at first it may seem that optimizing each transistor or wire in a block to the tightest possible dimension would be superior to scaling larger sections in tandem, such an approach fails to recognize two important factors: first, that density in a particular FUB may not reduce the overall dimension of the product, and second, that many FUBs have specific size and routing relationships that must be maintained. By analyzing the multiple wires in a given tile or strand and optimizing the performance of wire patterns in the new process, the Pshift achieved a cell density and wire optimization to ensure critical performance characteristics.

The ability to extract, modify, and experiment rapidly with different metal patterns and cell abstracts allowed for co- optimization on multiple vectors, leading to the high-quality starting layout for the Westmere family of products. With an easily scalable design starting point, we achieved the fastest time to market for a deep sub-micron CPU project in Intel history, while simultaneously delivering or exceeding the product goals for cost, area, performance, and power.

## Design Re-use

The initial design of a microprocessor is a major undertaking; yet, it is only the beginning. Re-use and re-verification are the keys to moving an initial microprocessor product into different markets; once the initial design is complete the team creates multiple steppings. Steppings are simply refinements of the first product, with smaller sets of changes to increase the products value. In steppings the initial design is re-used, processor



frequency is increased, or additional features are added, such as more cores, for greater performance. The number of products created from the initial design, as well as the implementation schedule, brings new levels of complexity to the design process that necessitates an innovative approach to design. We created both an inheritance data management system and a central verification system (CV) to easily track changes in FUBs, while simultaneously re-verifying these FUBs and others indirectly affected by the changes in the FC model for a given product.

### **Inheritance Database for Design Re-use**

Of the hundreds of FUBs in the initial design of the microprocessor we created, 25% never changed, and these were managed entirely by the CV system through six additional silicon steppings. Without a method to manage both inherited and changing data, much effort would have been spent on re-design or copying data—an inherently error-prone task. By using inheritance, 25% of the functional blocks could be virtually ignored by their designers during silicon steppings. We managed the additional designs in an incremental update mode that required much less effort than a re-design of the FUBS from scratch.

The inheritance system was similar to systems used to manage software development. The data had a mainline branch for each product; if a functional block did not change, the next stepping used the previous data. If the functional block did need to change, then the block was branched, and it was no longer inherited from the previous stepping. Over time, the scope of design changes was reduced, such that the last stepping needed changes in only 12% of the FUBs.

Microprocessor CV includes many types of tool runs and a complex integration task to connect the FUBs. All tools and flows accessed the inheritance database to determine the correct FUB and branch for each product. The designer who needed to change a FUB branched the FUB that in turn resulted in database updates propagating throughout the CV system. A central database kept track of every FUB and its inheritance state in order that tools, flows, and the integration task used the correct data.

### **Incremental Verification of Building Blocks**

Building an efficient CV system required automating methods to ensure the FUBs met their targets in all products. Like design, verification is broken down into tool runs on FUBs, with each FUB going through all required checks before the die can be sent to the factory for production. These checks include timing analysis to check frequency targets, logic analysis to verify the logical correctness of the circuit, physical design checks on the mask layout, power analysis to check multiple modes of power consumption, and reliability verification. The number of verification steps required, even after the design is complete, quickly skyrockets given these five verification vectors, 800 functional blocks, integration flows which check interfaces, and parallel products under development at the same time.

*“The inheritance system was similar to systems used to manage software development.”*

*“Like design, verification is broken down into tool runs on FUBs, with each FUB going through all required checks before the die can be sent to the factory for production.”*

*“Behind the scenes the CV system allocates a directory for the FUB to run in, brings input data into the directory, and launches each tool into the netbatch queues.”*

*“In order to further automate the process for a design engineer, the CV system also provides checks that look at input data versus output data.”*

The CV team created and delivered new software in three key areas to manage the verification complexity, a GUI to manage the high-volume runs, a trigger mechanism to detect the need for a FUB to run, and a set of quality checkers to detect a fatal issue in a FUB that would cause it not to function in silicon.

The GUI manages FUB runs by using database inputs, tracking, and outputs, which is different from how design engineers manage their FUBS: designers run FUBs one at a time by using a directory whose data are locally populated. The central verification GUI reads the checked-in data, runs the flows from start to finish, and saves the data in well-defined output locations. To run a FUB, simply find it in the GUI, click on it, and starts the Tools menu. If you want to do multiple runs, select all FUBS and pick the flows to run. The flows write back the status to a SQL database, and this state is reflected in the GUI. The user, therefore, can monitor the status of many FUB runs at one time.

Behind the scenes the CV system allocates a directory for the FUB to run in, brings input data into the directory, and launches each tool into the netbatch queues. As each tool completes and writes data back, the next tool is launched according to a flow that is set up and managed in the SQL database for each FUB.

During steppings we managed the FUBS by section with one design engineer responsible for up to 50 FUBS. The design engineer also benefited from the auto-debug mechanism in the GUI. Traditionally, a user would look in log files for errors if the tools failed to determine what the issue was. The CV handles this task automatically for the user by indicating in a GUI field why the FUB run failed. Once the FUBs have run through the tool successfully, output is generated and rolled up so the design engineer can look for problems that need to be addressed or confirm that the FUB and its neighbors still work correctly.

Any issues with quality were found and run by the CV system. These checks parse output data from the tools, and by using targets defined by the project experts, indicate if the FUB passed a check, if the FUB is waived or approved for a check, or if the FUB failed a check. There were roughly 100 checks run as part of the final verification. The design engineers ran a superset of these checks and drove them clean as part of their closure processes. Then, during steppings, the design engineer watching 50 FUBS is simply looking for incremental issues caused by a change in the FUB or a neighboring FUB.

In order to further automate the process for a design engineer, the CV system also provides checks that look at input data versus output data. When the inputs change, the system automatically re-runs the FUBS (and doesn't wait for the engineer to click in the GUI and run them). This ensures that if relevant inputs have changed for a FUB, the data is refreshed. Many database shifts can require a re-run, including a change in the actual design input, or a shift in the FC database. This triggering mechanism managed FUB runs until two weeks before Tape-in. At that point we refreshed all FUBS to ensure all issues were found, re-run, and fixed.

## Conclusion

The Intel Core i7 family of CPUs was designed by using a suite of tools crafted to minimize design effort:

1. Abstract RTL design and verification.
2. Sequential formal equivalence.
3. Tile-based circuit and layout design.
4. Early integration and analysis techniques.
5. Advanced design choices to enable future re-use.

These tools and techniques minimized both the local effort at each stage of the design as well as the global effort, thereby, tying the design together and facilitating an easy transition between each generation of the Core i7 processor family.

## Complete References

- [1] IEEE 1800™. “Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language,” *IEC 62530*. ISBN 2-8318-9349-6, 2007.
- [2] Daher Kaiss, Silvian Goldenberg, Zurab Khasidashvili. “Seqver: A Sequential Equivalence Verifier for Hardware Designs,” *ICCD 2006*.

## Author Biographical Sketches

**Wayne Clift** is a Principal Engineer with Intel and has 19 years of experience in high-performance CPU logic design and methodology development.

Wayne’s current interest is in RTL Design Abstraction and Formal Design Equivalence. Wayne holds six CPU register-renaming architecture patents. He received BS and MS degrees in Electrical and Computer Engineering from Brigham Young University in 1991. His email is wayne.clift at intel.com.

**Clark Brooks** is a Senior Component Design Engineer with Intel. He has 18 years of experience in CAD Tool design automation, helping deliver world-record ispec\_base performance chips in 1995, 2001, 2004, and 2009. He has published several papers at Intel’s Design and Test Technology Conference. He received an MS degree in Computer Science from Caltech in 1992. His email is clark.brooks at intel.com.

**Gohar Waqar** is an Intel Technical Lead in the Timing Analysis domain. He has 13 years of high-performance CPU circuit design and methodology experience. He received BS and MS degrees from Arizona State University in 1996. His email is gohar.waqar at intel.com.

**Vijay Kodithyala** is a Principal Engineer in the Physical Design and Verification, Analog Design Methodology Group at Intel. He received an MSEE degree from Auburn University in 1993. His email is vijaykumar.kodithyala at intel.com.

**Sean Mirkes** is a Principal Engineer with Intel and has 14 years of high-performance CPU circuit design and micro architecture experience. Sean's current areas of interest are circuit design efficiency and hardware accelerated cryptography. Sean holds three patents in the area of computer architecture. He received his BS degree in Computer Engineering from the University of Washington in 1996. His email is sean.p.mirkes at intel.com.

**Kami Huck** is an Engineer with Intel with 18 years of experience in high-performance CPU design and methodology development. Kami's current interest is in methodologies to enable quick-turn products from initial silicon and in tools and methods for high-volume central verification. Kami holds four patents in the area of microprocessor design. She received her MS degree from Washington State University in 1988. Her email is kamla.k.huck at intel.com.

**Kiron Pai** is the Technical Lead of the Database, Environment and Infrastructure Design Automation Team at Intel. Kiron's interests are in Design Data management and software engineering. He received an MS degree in Computer Science from Florida Tech in 1993. His email is kiron.pai at intel.com.

**Marijan Peršun** is an Intel Principal Engineer with 15 years of high-performance CPU circuit design and design methodology experience. He received BS and MS degrees in Electrical and Computer Engineering from the University of Zagreb, Croatia and from Portland State University, respectively. Marijan holds three US patents. His email is marijan.persun at intel.com.

**Alexey Gaydyukov** is a full-chip interconnect design automation expert with Intel with 10 years of experience in high-performance CPU design automation. Alexey's current interest is in interconnect analysis automation and post-silicon power validation. He received BS and MS degrees in Applied Physics and Math from the Moscow Institute of Physics and Technology in 2002. His email is alexey.a.gaidiukov at intel.com.

## Copyright

Copyright © 2011 Intel Corporation. All rights reserved.

Intel, the Intel logo, and Intel Atom are trademarks of Intel Corporation in the U.S. and other countries.

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

Requires an Intel® HT Technology enabled system, check with your PC manufacturer. Performance will vary depending on the specific hardware and software used. Not available on Intel® Core™ i5-750. For more information including details on which processors support HT Technology, visit <http://www.intel.com/info/hyperthreading>

Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM). Functionality, performance or other benefits will vary depending on hardware and software configurations. Software applications may not be compatible with all operating systems. Consult your PC manufacturer. For more information, visit <http://www.intel.com/go/virtualization>

Requires a system with Intel® Turbo Boost Technology capability. Consult your PC manufacturer. Performance varies depending on hardware, software and system configuration. For more information, visit <http://www.intel.com/technology/turboboost>