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.

"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."

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.

"The Nehalem family platform transition forced a fundamental reliance on in-silicon debug hooks and related tools."

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.

"Control is gained through the test access port (TAP) that traditionally had to interface with only one or two CPU cores."

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, 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.

"All previously developed test content for the Nehalem could be reused for Lynnfield and Arrandale client products."

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.

"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."

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.

"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.

"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."

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.

"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 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.

"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."

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.

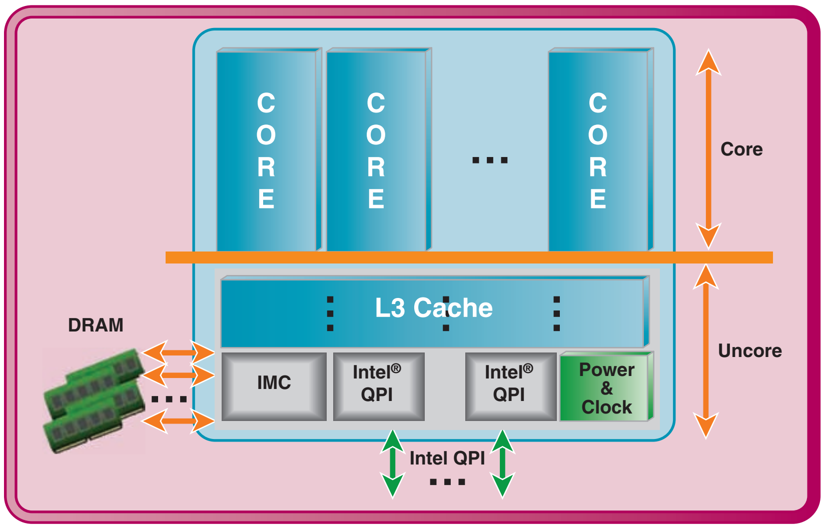

Figure 1: Nehalem modular design [3]

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."

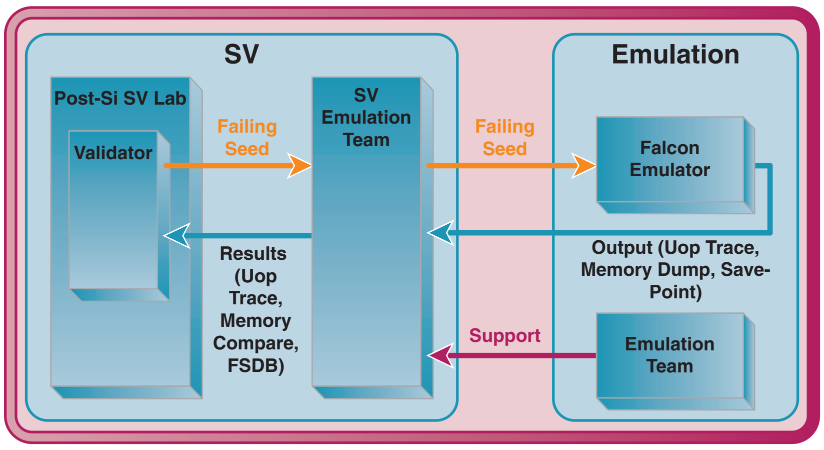

Figure 2: High-level block diagram of Emulation Reproduction Flow

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 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.

"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." They 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.

"The Nehalem team developed a method of combining the ability to freeze writes to arrays with a freeze of the clock."

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.

"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.

"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. 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.

"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

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.

"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 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.

"Defeaturing *is the ability to selectively disable specific features throughout the processor; it can greatly help with bug isolation.*"

Defeaturing

Defeaturing 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 defeaturing 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.

"The ability to observe major busses has historically been a significant part of debugging processors and processor platforms."

As with prior projects, we found that the defeaturing 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).

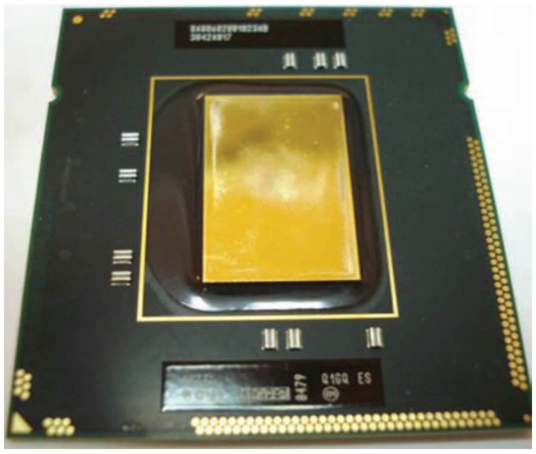

Figure 3: Topside of package showing mirror port pins on the bottom and right edges.

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.

"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."

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.

"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 mirror port bus consists of mirrored versions of the QPI Rx and Tx links that are selectable when there is more than one pair."

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."

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.

"To ensure large or more complex problems could also be solved, the chip also provided an interface to a logic analyzer."

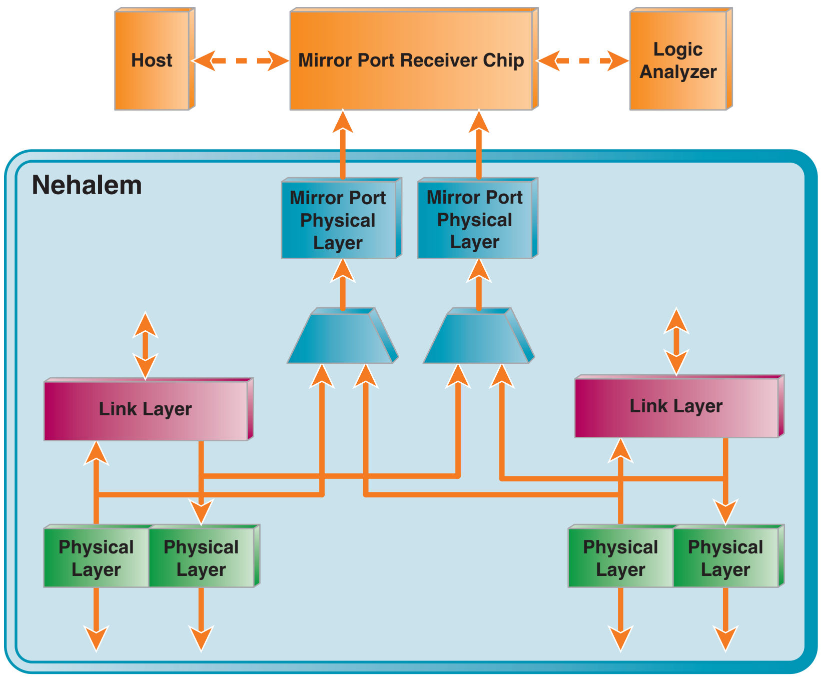

Figure 4: Conceptual drawing of system debug set-up showing mirror port internal pathways, top-side connection, mirror port receiver hardware

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.

"A QPI mirror port trace immediately revealed an illegal transaction caused by a low-level BIOS setting."

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.

"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."

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.

"As with DDR and QPI, REUT capability was built into the PCIe controller to enable EV testing and debug of the 5GT/s interface."

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.

"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."

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.

"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."

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.

"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."

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.

"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."

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 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.

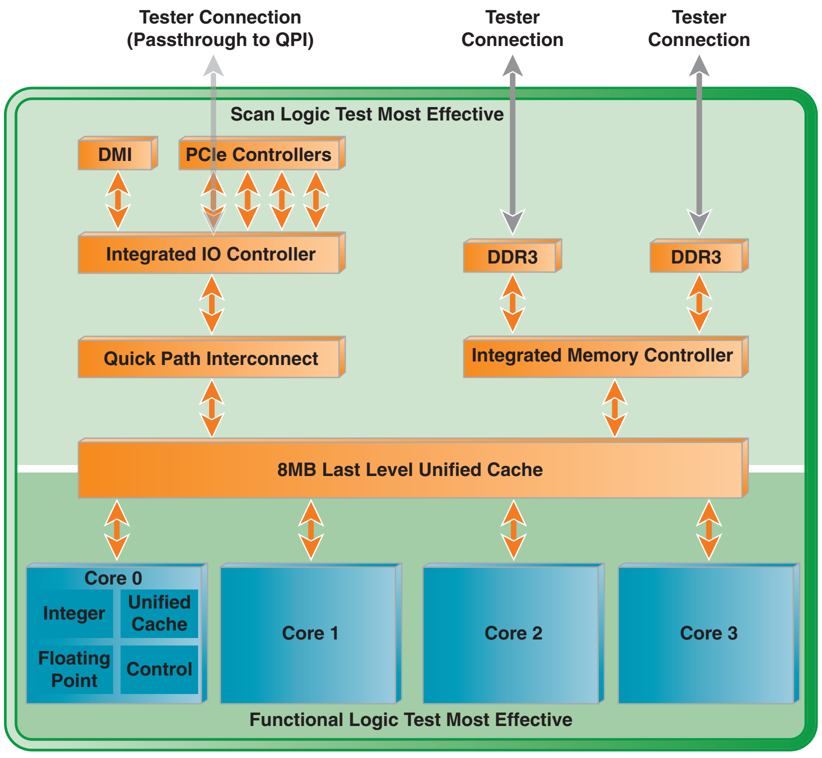

Figure 5: Architecture of tester bypass

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.

"A pass-through mode was introduced —essentially allowing data from the PCIe and DMI pins to be passed directly to QPI."

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.

"Due to its multi-core design, the Nehalem family included multiple TAP finite-state-machine controllers, which connect to the JTAG interface."

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.

"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. 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.

"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."

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.

"These combined test features enabled fault grade coverage targets in the mid 90% range."

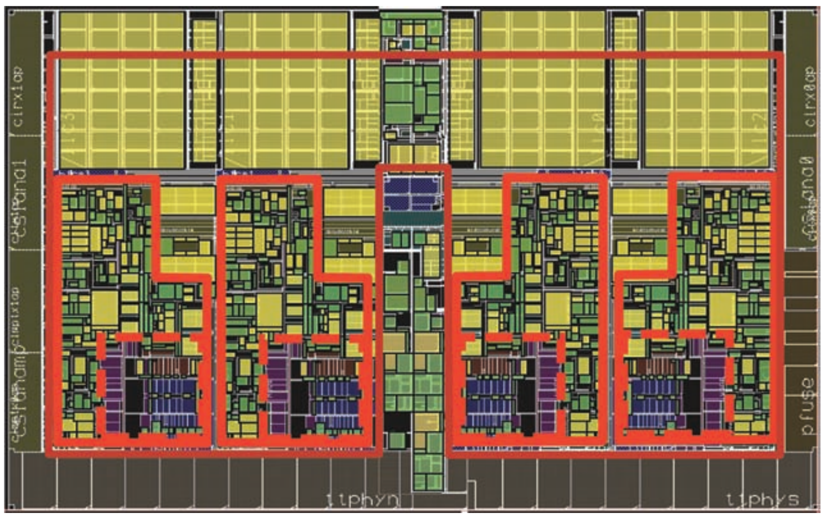

Figure 6: Die plan showing overlay of HVM Test features (array = yellow, scan = green, on-die functional = red outlines, grey = IO, tester functional = whole die)

"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.

"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."

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.

"We developed the mirror port as a non-perturbing observability agent that provided a similar point of observability as the previously familiar FSB."

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.

[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