# Validating the Intel® Pentium® 4 Microprocessor

Bob Bentley
Intel Corporation
2111 N.E. 25th Avenue
Hillsboro, Oregon 97124, U.S.A.
bob.bentley@intel.com

## ABSTRACT

Developing a new leading edge IA-32 microprocessor is an immensely complicated undertaking. In the case of the Pentium® 4 processor, the microarchitecture is significantly more complex than any previous IA-32 microprocessor and the implementation borrows almost nothing from any previous implementation. This paper describes how we went about the task of finding bugs in the Pentium® 4 processor design prior to initial silicon, and what we found along the way.

## General Terms

Management, Verification.

## 1. INTRODUCTION

Validation case studies are relatively rare in the literature of computer architecture and design ([1] and [2] contain lists of some recent papers) and case studies of commercial microprocessors are even rarer. This is a pity, since there is as much to be learned from the successes and failures of others in the validation area as in other, more richly documented, fields of computer engineering. In fact, given the cost of an undetected bug escaping into production silicon - where cost is measured not only in a narrow monetary sense but more broadly in the impact on a society that is increasingly dependent on computers – it can be argued that the validation field deserves much more attention than it has received to date.

The microarchitecture of the Pentium® 4 processor is significantly more complex than any previous IA-32 microprocessor, so the challenge of validating the logical correctness of the design in a timely fashion was indeed a daunting one. In order to meet this challenge, we applied a number of innovative tools and methodologies which enabled us to keep validation off the critical path to tapeout while meeting our goal of ensuring that first silicon was functional enough to boot operating systems and run applications. This in turn enabled the post-silicon validation teams to quickly "peel the onion", resulting in an elapsed time of only 10 months from initial tapeout to production shipment qualification – an Intel record for a new IA-32 microarchitecture.

## 2. OVERVIEW

The Pentium® 4 processor is Intel's most advanced IA-32 microprocessor, incorporating a host of new microarchitectural features including a 400-MHz system bus, hyper-pipelined technology, advanced dynamic execution, rapid execution engine, advanced transfer cache, execution trace cache, and Streaming SIMD (Single Instruction, Multiple Data) Extensions 2 (SSE2).

For the most part, we applied similar tools and methodologies to validating the Pentium® 4 processor that we had used previously on the Pentium® Pro processor. However, we developed new methodologies and tools in response to lessons learnt from previous projects, and to address some of the new challenges that the Pentium® 4 processor design presented from a validation perspective. In particular, the use of Formal Verification, Cluster Test Environments and focused Power Reduction Validation were either new or greatly extended from previous projects; each of these is discussed in more detail in a section of this paper.

### 2.1 Timeline

A brief timeline of the Pentium® 4 processor project, as it relates to this paper, is as follows:

- Structural RTL (SRTL) work began in late 1996 at the cluster level, with the first full-chip SRTL integration occurring in the spring of 1997.
- Structural RTL was largely completed (except for bug fixes and rework due to speed path fixes) by the end of Q2 1998.
- A-step tapeout occurred in December 1999.
- First packaged parts arrived in January 2000.
- Initial samples were shipped to customers in the first quarter of 2000.
- Production ship qualification was granted in October 2000.
- Pentium® 4 processor was launched in November 2000 at frequencies of 1.4 and 1.5 GHz.

### 2.2 Staffing

One major challenge that we faced right away was the need to build the pre-silicon validation team. We had a nucleus of 10 people who had worked on the Pentium Pro® processor, and who could do the initial planning for the Pentium® 4 project while at the same time working with the architects and designers who were refining the microarchitectural concepts. However, it was clear that 10 people were nowhere near enough for a 42 million-transistor design that ended up requiring more than 1 million lines of SRTL code to describe it. So we mounted an extensive recruitment campaign (focused mostly on new college graduates) that resulted in approximately 40 new hires in 1997 and another 20 in 1998. Not only did this take a large amount of effort from the original core team (at one stage we were spending on aggregate 25% of our total effort on recruiting!), but it also meant that we faced a monumental task in training these new team members. However, this investment repaid itself handsomely over the next few years as the team matured into a highly effective bug-finding machine that was responsible for finding almost 5000 of the 7855 total logic bugs that were filed prior to tapeout. In doing so, they developed an in-depth knowledge of the Pentium® 4 microarchitecture that has proved to be invaluable in post-silicon logic and speedpath debug and also in fault-grade test writing.

## 2.3 Validation Environment

Pre-silicon logic validation was done using either a cluster-level or full-chip SRTL model running in the *csim* simulation environment from Intel Design Technology. We ran these simulation models on either interactive workstations or compute servers – initially, these were legacy IBM RS6Ks running AIX, but over the course of the project we transitioned to using mostly Pentium® III based systems running Linux. The full-chip model ran at speeds ranging from 05-0.6 Hz on the oldest RS6K machines to 3-5 Hz on the Pentium® III based systems (we have recently started to deploy Pentium® 4 based systems into our computing pool and are seeing full-chip SRTL model simulation speeds of around 15 Hz on these machines). The speeds of the cluster models varied, but all of them were significantly faster than full-chip. Our computing pool grew to encompass several thousand systems by the end of the project, most of them compute servers. We used an internal tool called *netbatch* to submit large numbers of batch simulations to these systems, which we were able to keep utilized at over 90% of their maximum 24/7 capacity. By tapeout we were averaging 5-6 billion cycles per week and had accumulated over 200 billion (to be precise, 2.384 * 1011) SRTL simulation cycles of all types. This may sound like a lot, but to put it into perspective, it is roughly equivalent to 2 minutes on a single 1 GHz CPU!

## 3. FORMAL VERIFICATION

The Pentium® 4 processor was the first project of its kind at Intel to apply Formal Verification (FV) on a large scale. We decided early in the project that the FV field had matured to the point where we could consider trying to use it as an integral part of the design verification process rather than only applying it retroactively, as had been done on previous products such as the Pentium® Pro processor. However, it was clear from the start that we couldn't formally verify the entire design – that was (and still is) way beyond the state of the art for today's tools. So we decided to focus on the areas of the design where we believed that FV could make a significant contribution – in particular, the floating-point execution units and the instruction decode logic. As these areas had in the past been sources of bugs that escaped detection and made it into released silicon, this allowed us to apply FV to some real problems with real payback.

One of the major challenges for the FV team was to develop the tools and methodology needed to handle a large number of proofs in a highly dynamic environment. For the most part we took a model-checking approach to FV, using the prover tool from Intel's Design Technology group to compare SRTL against separate specifications written in FSL. By the time we taped out we had over 10,000 of these proofs in our proof database, each of which had to be maintained and regressed as the SRTL changed over the life of the project. Along the way, we found over 100 logic bugs – not a large number in the overall scheme of things, but about 20 of them were "high quality" bugs that we do not believe would had been found by any other of our pre-silicon validation activities. Two of these bugs were classic floating-point data space problems:

- The FADD instruction had a bug where, for a specific combination of source operands, the 72-bit FP adder was setting the carryout bit to 1 when there was no actual carryout;

- The FMUL instruction had a bug where, when the rounding mode was set to "round up", the sticky bit was not set correctly for certain combinations of source operand mantissa values, specifically:

src1[67:0] := X*2(i+15) + 1*2i

src2[67:0] := Y*2(j+15) + 1*2j

where i+j = 54, and {X,Y} are any integers that fit in the 68-bit range

Either of these bugs could easily have gone undetected[1] not just in the pre-silicon environment but in post-silicon testing also. Had they done so, we would have faced the prospect of a recall similar to the Pentium® processor's FDIV problem in 1994.

We put a lot of effort into making the regression of the FV proof database as push-button as possible, not only to simply the task of running regressions against a moving SRTL target but because we viewed reuse as being one of the keys to proliferating the quality of the original design. This approach has already paid off: a regression of the proof database on the first proliferation of the Pentium® 4 processor has yielded a complex floating point bug.

## 4. CLUSTER-LEVEL TESTING

One of the fundamental decisions that we took early in the Pentium® 4 processor development program was to develop Cluster Test Environments (CTEs) and maintain them for the life of the project. There is a CTE for each of the 6 clusters into which the Pentium® 4 processor design is logically subdivided (actually, microcode can be considered to be a seventh logical cluster, and it too has a test environment equivalent to the other CTEs). These CTEs are groupings of logically related units (e.g. all the execution units of the machine constitute one CTE) surrounded by code that emulates the interfaces to adjacent units outside of the cluster and provides an environment for creating and running tests and checking results.

The CTEs took a good deal of effort to develop and maintain, and were themselves a source of a significant number of bugs (not counted in the 7855 total). However, they provided a number of key advantages:

- First and foremost, they provided **controllability** that was otherwise lacking at the full-chip level. An out of order,

[1] The probability of hitting the FMUL condition with purely random operands is approximately 1 in $5*10^{20}$, or 1 in 500 million trillion!

speculative execution engine like the Pentium® Pro or Pentium® 4 processor is inherently difficult to control at the instruction set architecture level. Assembly-language instructions (macroinstructions) are broken down by the machine into sequences of microinstructions that may be executed in any order (subject to data dependencies) relative to one another and to microinstructions from other preceding or following macroinstructions. Trying to produce precise microarchitectural behavior from macroinstruction sequences is like pushing on a piece of string. This problem is particularly acute for the back end of the machine – the memory and bus clusters which lie beyond the out-of-order section of the microarchitecture pipeline. CTEs allowed us to provoke specific microarchitectural behavior on demand.

- Second, CTEs allowed us to make significant strides in **early validation** of the Pentium 4 processor SRTL even before a full-chip model was available. Integrating and debugging all the logic and microcode needed to produce even a minimally functional full-chip model was a major undertaking; it took more than 6 months from the time we started until we had a "mostly functional" IA-32 machine that we could start to target for aggressive full-chip testing. Because we had the CTEs, we were able to start testing as soon as there was released code in a particular unit, long before we could have even tried to exercise it at the full-chip level.
- Even after we had a full-chip model, the CTEs essentially **decoupled validation** of individual unit features from the health of the full-chip model. A killer bug in (say) the front end of the machine did not prevent us from continuing to validate in other areas. In fact, though, we rarely encountered this kind of blockage; our development methodology required that all changes be released at cluster level first, and only when they had been validated there did we propagate them to full-chip. Even then, we required that all full-chip model builds pass a mini-regression test suite before they could be released to the general population. This caught most major cross-unit failures that could not be detected at the CTE level.

## 5. POWER REDUCTION VALIDATION

From the earliest days of the Pentium® 4 processor design, power consumption was a concern. Even with the lower operating voltages offered by P858, it was clear that at the operating frequencies we were targeting we would have difficulty fitting within the "thermal envelope" that was needed so that a desktop system would not require exotic and expensive cooling technology. This led us to include in the design two main mechanisms for active power reduction: clock gating and thermal management.

Clock gating as a concept is not new: previous designs have attempted to power down discrete structures like caches when there were no accesses pending. What was different about the Pentium® 4 processor design was the extent to which clock gating was taken. Every unit on the chip had a power reduction plan, and almost every functional unit block contained clock-gating logic – in all, there were around 350 unique clock-gating conditions identified. Every one of them needed to be validated from several different perspectives:

- We needed to verify that each condition was implemented as per plan and that it functioned as originally intended. We needed to verify this not once, but continually throughout the development of the Pentium® 4 processor, as otherwise it was possible for power savings to be eroded over time as an unintended side effect of other bug or speedpath fixes. We tackled this problem by constructing a master list of all the planned clock gating features, and writing checkers in proto for each condition to tell us if the condition had occurred and to make sure that the power down had occurred when it should have. We ran these checkers on cluster regressions and low-power tests to develop baseline coverage, and then wrote additional tests as necessary to hit uncovered conditions.
- While establishing this coverage, we had to make sure that the clock gating conditions did not themselves introduce new logic bugs into the design. It is easy to imagine all sorts of nightmare scenarios: unit A is late returning data to unit B because part of A was clock gated, or unit C samples a signal from unit D that is undriven because of clock gating, or other variations on this theme. In fact, we found many such bugs, mostly as a result of (unit level) design validation or full-chip microarchitecture validation, using the standard set of checkers that we employed to catch such implementation-level errors. We had the ability to override clock gating either selectively or globally, and developed a random power down API that could be used by any of the validation teams to piggyback clock gating on top of their regular testing. Once we had developed confidence that the mechanism was fundamentally sound, we built all our SRTL models to have clock gating enabled by default.
- Once we had implemented all the planned clock gating conditions, and verified that they were functioning correctly, we relied primarily on measures of clock activity to make sure that we didn't lose our hard-won power savings. We used a special set of tests that attempted to power down as much of each cluster as possible, and collected data to see what percentage of the time each clock in the machine was toggling. We did this at the cluster level and at full-chip. We investigated any appreciable increase in clock activity from model to model, and made sure that it was explainable and not due to designer error.
- Last, but by no means least, we tried to make sure that the design was cycle-for-cycle equivalent with clock gating enabled and disabled. We had established this as a project requirement, to lessen the likelihood of undetected logic bugs or performance degradation caused by clock gating. To do this, we developed a methodology for temporal divergence testing which essentially ran the same set of tests twice, with clock gating enabled and disabled, and compared the results on a cycle-by-cycle basis.

We organized a dedicated Power Validation team to focus exclusively on this task. At peak, there were 5 people working on this team, and even in steady-state when we were mostly just regressing the design it still required 2 people to keep this activity going. However, the results exceeded our fondest expectations: clock gating was fully functional on initial silicon, and we were able to measure approximately 20W of power saving in a system running typical workloads. The Power Validation team filed over 200 bugs themselves as a result of pre-silicon validation (we filed "power bugs" whenever the design did not implement a power-saving feature correctly, whether or not it resulted in a functional failure).

## 6. METHODOLOGY ISSUES

### 6.1 Full-chip Integration and Testing

With a design as complex as the Pentium® 4 processor, integrating the pieces of SRTL code together to get a functioning full-chip model (let alone one capable of executing IA-32 code) is not a trivial task. We developed an elaborate staging plan that detailed what features were to be available in each stage, and phased the integration over a roughly 12-month period. The architecture validation (AV) team developed tests that would exercise the new features as they became available in each phase, but did not depend upon any as-yet unimplemented IA-32 features. These tests were throwaway work - their main purpose was to drive the integration effort by verifying basic functionality.

Along with these tests we developed a methodology which we called feature pioneering: when a new feature was released to full-chip for the first time, a validator took responsibility for running his or her feature exercise tests, debugging the failures and working with designers to rapidly drive fixes into graft (experimental) models, bypassing the normal code turn-in procedure, until an acceptable level of stability was achieved. Only then was the feature made available for more widespread use by other validators. We found that this methodology greatly speeded up the integration process; as a side effect, it also helped the AV team develop their full-chip debugging skills much more rapidly than might otherwise have occurred.

Once a fully functional full-chip SRTL model was available (in mid-1998) these feature pioneering tests were discarded, and replaced by a new suite of over 12,000 IA-32 tests developed by the AV team, whose purpose was to fully explore the architecture space. Previous projects up to and including the Pentium® Pro processor had relied on an "ancestral" test base inherited from the past, but these tests had little or no documentation, unknown coverage and doubtful quality (in fact, many of them turned out to be bug tests from previous implementations that had little architectural value). We did eventually run the "ancestral" suite as a late cross-check, after the new suite had been run and the resulting bugs fixed, but we found nothing of consequence as a result, indicating that it can at long last be retired.

### 6.2 Coverage-Based Validation

We attempted wherever possible to use coverage data to provide feedback on the effectiveness of our tests, and tell us what we had and had not tested; this in turn helped direct future testing towards the uncovered areas. Since we relied very heavily on directed random test generators for most of our microarchitectural testing, coverage feedback was an absolute necessity if we were to avoid "spinning our wheels" and testing the same areas over and over again while leaving others completely untouched. In fact, we used the tuple of {*cycles run*, *coverage gained* and *bugs found*} as a first-order gauge of SRTL model health and tapeout readiness.[2]

Our primary coverage tool was *proto* from Intel Design Technology, which we used to create coverage monitors and measure coverage for a large number of microarchitecture conditions. By tapeout we were tracking almost 2.5 million unit-level conditions, and more than 250,000 inter-unit conditions, and succeeded in hitting almost 90% of the former and 75% of the latter. We also used proto to instrument several thousand multiprocessor memory coherency conditions (combinations of microarchitecture states for caches, load and store buffers, etc.), and, as mentioned above, the clock gating conditions that had been identified in the unit power reduction plans.

We also used the *pathfinder* tool from Intel's Central Validation Capabilities group to measure how well we were exercising all the possible microcode paths in the machine. Much to our surprise, running all of the AV test suite yielded microcode path coverage of less than 10%; further analysis revealed than many of the untouched paths involved memory-related faults (e.g. page fault) or assists (e.g. A/D bit assist). When we thought about it, this made sense - the test writers had set up their page tables and descriptors so as to avoid these time-consuming functions (at 3 Hz, every little bit helps!). We modified our tests and tools to cause them to exercise these uncovered paths, and did indeed find several bugs in hitherto untested logic. This reinforced our belief in the importance of using coverage feedback and not just assuming that specified conditions are being hit.

## 7. RESULTS

We compared the bugs found by pre-silicon validation of the Pentium® 4 processor with those found in the equivalent stage of the Pentium® Pro development. From one microprocessor generation to the next, we recorded a 350% increase in the number of bugs filed against SRTL prior to tapeout. Cluster-level testing proved to be a big win, as 3411 of the 5809 bugs found by dynamic testing were caught at the CTE level with the other 2398 being found on the full-chip SRTL model. Code inspection was, as always, a highly effective technique that accounted for 1554 bugs, with the remaining 492 being found by Formal Verification, SRTL-to-schematic equivalence verification, and several other minor categories.

We observed a somewhat different bug breakdown by cluster: on the Pentium® Pro processor microcode was the largest single source of bugs, accounting for over 30% of the total, where as on the Pentium® 4 processor it was less than 14%. We attribute this difference primarily to the fact that on the Pentium® Pro processor we had to develop from scratch all of the IA-32 microcode algorithms for an out-of-order, speculative execution engine; the Pentium® 4 processor was able to leverage many of the same algorithms, resulting in far fewer microcode bugs.

For both designs, the Memory Cluster was the largest source of hardware bugs, accounting for around 25% of the total in both cases. This is consistent with data from other projects, and

[2] We had an extensive list of tapeout-readiness criteria, which we developed and reviewed within the Pentium® 4 development team more than a year before tapeout. Experience has taught us that development of such criteria should not be delayed until tapeout is imminent.

indicates that we need to apply more focus to preventing bugs in this area.

We did a statistical study [3] to try to determine how bugs were introduced into the Pentium® 4 processor design. The major categories, amounting to over 75% of the bugs analyzed, were:

- Goof (12.7%) - these were things like typos, cut and paste errors, careless coding when in a hurry, or situations where the designer was counting on testing to find the bugs.
- Miscommunication (11.4%) - these fall into several categories: architects not communicating their expectations clearly to designers, misunderstandings between microcode and design as well as between different parts of design (e.g. misassumptions about what another unit was doing).
- Microarchitecture (9.3%) - these were problems in the microarchitecture definition.
- Logic/Microcode changes (9.3%) - these were cases where the design was changed, usually to fix bugs or timing problems, and the designer did not take into account all the places that would be impacted by the change.
- Corner cases (8%) – as the name implies, these were specific cases which the designer failed to implement correctly.
- Power down issues (5.7%) – these were mostly related to clock gating.
- Documentation (4.4%) - something (algorithm, micro-instruction, protocol) was not documented properly.
- Complexity (3.9%) – although some of the bugs categorized under the "Miscommunication" or "Microarchitecture" headings were undoubtedly the result of complexity in the design, these were bugs whose cause was specifically identified as being due to microarchitectural complexity.
- Random initialization (3.4%) – these were mostly bugs caused by state not being properly cleared or initialized at reset
- Late definition (2.8%) - certain features were not defined until late in the project. This led to shoehorning them into working functionality (similar to the logic/microcode changes category). Also, because they were defined late, they were sometimes rushed and were not always complete or fully thought out.
- Incorrect RTL assertions (2.8%) - these refer to assertions (instrumentation in the SRTL code) that were either wrong, overzealous, or had been working correctly but were broken by a design change (usually timing induced).
- Design mistake (2.6%) - the designer misunderstood what he/she was supposed to implement. Normally this was a result of not fully reading the specification or starting implementation before the specification was complete.

Despite (or, perhaps, because of) the large number of bugs found by pre-silicon validation, the Pentium® 4 processor was highly functional on A-0 silicon, and received production qualification only 10 months after initial tapeout.

## 8. ACKNOWLEDGMENTS

The Pentium® 4 microprocessor validation program was the work of a large team of people over a 4-year period. It is impossible to list here the names of everyone who contributed to this effort, but I am especially indebted to Blair Milburn, Keiichi Suzuki, Rick Zucker, Bob Brennan, Tom Schubert and John Bauer, both for their leadership of the pre-silicon validation activities and for their valuable feedback on early drafts of this paper.

## 9. REFERENCES

[1] Fournier, L, Arbetman, Y and Levinger, M: "Functional Verification Methodology for Microprocessors Using the Genesys Test-Program Generator - Application to the x86 Microprocessors Family", *Proceedings of the Design Automation and Test in Europe Conference and Exhibition (DATE99)*, March 1999.

[2] Ho, R, Yang, H, Horowitz, M, and Dill, D: "Architecture Validation for Processors", *ISCA 95: Internaltional Conference on Computer Architecture*, June 1995.

[3] Zucker, R, "Bug Root Cause Analysis for Willamette*", Intel Design and Test Technology Conference*, August 2000.