ISA-Independent Post-Silicon Validation for the Address Translation Mechanisms of Modern Microprocessors

George Papadimitriou [1] Athanasios Chatzidimitriou [1] Dimitris Gizopoulos [1] Ronny Morad [2]

[1] Computer Architecture Laboratory, University of Athens, Greece {georgepap | achatz | dgizop}@di.uoa.gr
[2] IBM Research Labs, Haifa, Israel morad@il.ibm.com

***Abstract*—Post-silicon validation complements traditional simulation-based pre-silicon verification and offers very high throughput since validation programs run at the speed of the actual hardware. Detection of bugs in the address translation subsystem of a microprocessor is much less straightforward than other hardware blocks because the address translation is an implicit process, which does not have an easily observable output to architecture or program visible locations. Validation of the correctness of the address translation mechanisms (ATMs) of microprocessors is both very important and challenging problem. In this paper, we present an ISA-independent methodology for the post-silicon validation of the ATMs in modern microprocessors. We first capture the effects of design bugs in address translation, by presenting actual bugs scenarios reported for commercial chips. We also describe an effective method for the detection of bugs in all address translation hardware blocks. The validation programs of the method are self-checking, i.e. do not require a bug-free model to compare with. Our experimental evaluation on Gem5 simulator shows the effectiveness of the methodology in detecting bugs in the address translation hardware of an x86-64 microprocessor model.**

Keywords—post-silicon validation; address translation mechanisms

I. Introduction

All major microprocessor vendors regularly extend the ISAs for new functionalities and the microarchitectures with extra hardware support for performance and programmability. Virtual memory is a very mature abstraction that is widely implemented through dedicated ISA and microarchitecture support in most computing systems today [15]. Virtual-to-physical addresses translation is the key step that a microprocessor is required to implement correctly and efficiently for a successful realization of virtual memory. With the galloping adoption of virtualization today (and the complexity its support adds to ISAs and microarchitectures now requiring multi-level address translation), the performance and correctness of address translation gets critical. Larger TLBs and more complex microarchitectural caching structures are used to improve the speed of address translation, which is invoked multiple times as instructions and data are fetched from the memory hierarchy. The address translation subsystem of a modern microprocessor is one among several complex mechanisms realized in state-of-the-art microarchitectures today. However, address translation is probably the most difficult subsystem for correctness validation [4], [10]. Unlike other hardware components (functional blocks, registers, caches) the output of the address translation mechanism (ATM) of a microprocessor (the physical address) is not observable to architecturally visible locations (registers, memory). Moreover, the address translation process involves several stages and several different hardware and design bugs in each of these blocks can lead to wrong address translations.

Post-silicon validation offers validation throughput at the actual chip speed (millions of validation programs can be executed) unlike simulation-based pre-silicon verification, which is very detailed (all internals of a design can be observed) but is several orders of magnitude slower that program execution on the real hardware [6], [11], [12], [20], [21]. The major downside, however, of post-silicon validation is the very limited observability of the internal signals and the difficulty to reproduce the execution of a failing validation program on a simulator to identify the actual cause of a failure. Recent microprocessor post-silicon validation approaches [16], [22], [37] have delivered self-checking validation programs, i.e. programs that do not require previously calculated (through simulation or emulation) golden/correct responses. Such responses are extremely hard to be available because millions of validation programs can be executed on the hardware prototypes but not on the extremely slower architectural simulators.

Although post-silicon validation has been employed to validate the correctness of several important hardware structures of a microprocessor [13], [16], [17], [19], [22], [34], [35], [37] there is no public work on the post-silicon validation of the ATM of modern microprocessors. Major microprocessor vendors (Intel, AMD, and IBM) have reported in chip's errata sheets several ATM-related bugs that slipped into volume production of microprocessors [18], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32].

In this paper, we propose an ISA-independent methodology for the generation of self-checking validation programs for the ATM hardware structures. The methodology is based on: (a) established verification/validation program generators (such as [2], [10], [14]); (b) the construction of a dedicated memory image (and corresponding custom page table) to achieve the self-checking property of the programs and thus decouple the method from simulation. We implement our method on the x86-64 model of the Gem5 simulator by fully resembling a post-silicon validation bare-metal setup and evaluate its effectiveness in the detection of bug.

II. Address Translation

A. An Overview

In a modern computing system with virtual memory and virtualization support, the role of address translation is crucial

Chip	Bug description	Effect
AMD Athlon64/Opteron [23]	Possible use of stale translations even after software has performed a TLB flush	Unpredictable system failure
IBM PowerPC 750GX/750GL [27]	A mtsr or mtsrin operation, followed closely by an instruction that causes data page address translation, can cause contention for the segment registers, which proceeds using data from the wrong segment register	Access to incorrect PA or false translation and data access exceptions
Intel® Xeon® Processor 5100 Series [29]	When an unaligned access is performed on paging structure entries, accessing a portion of two different entries simultaneously, the processor may live lock	The processor may live lock causing a system hang
AMD Athlon64/Opteron [23]	INVLPG instruction with address prefix does not correctly invalidate the requested translation in the TLB	Unpredictable system failure
AMD K10 (Family 10h) [24]	The processor may use an incorrect cached copy of translation tables when it is in legacy Physical Address Extension (PAE) mode and the guest address translation tables reside in physical page zero	Unpredictable system behavior

Table 1: Published ATM-related bugs from official errata sheets.

for the correctness and the performance of programs execution. The CPU-memory performance gap is successfully bridged by multi-level caches hierarchies, but the need for frequent virtual-to-physical addresses translation may become a serious performance bottleneck. Every running process generates multiple accesses to its virtual address space (including references to code, stack, and heap segments). If the translation of the virtual addresses to the physical memory space of the system is not fast enough, the execution throughput of the system can dramatically decline. The address translation mechanisms (ATMs) of modern microprocessors:

- Map the virtual pages (VPs) of software processes to the physical frames (PFs) of the system memory.
- Keep track of the access permission rights for each page (user vs. kernel pages, etc.).
- Record the pages status (accessed, modified, etc.).

Altogether, the mapping, permissions and status information about the virtual pages and the physical frames of the system are stored in page tables (PTs) which can have different organizations and contents in different microprocessors. Every page table contains Page Table Entries (PTEs) for each virtual-to-physical mapping as well as other attributes (read/write, user/kernel, etc.), which uniquely characterize each frame in the physical memory space. Page tables are stored in main memory and secondary storage, so when an address translation needs to access them the latency is very high (several hundreds of clock cycles).

Caching of address translations is a very effective way to reduce the very high access latency of PTEs. All microprocessors realize address translation caching through associative Translation Lookaside Buffers (TLBs – which store the most recent VP to PF mappings as well as access permissions and page reference status information) and other caching mechanisms [1]. TLBs are accessed through a Virtual Page Number (VPN). On a TLB hit, the translation is promptly available and execution continues. On a TLB miss, the PT of the process is "walked" and the translation is located. When located, the translation is also stored in the TLB for future use. The primary goal of any caching in a microprocessor ATM subsystem is to keep the address translation latency off the critical path of memory access [7].

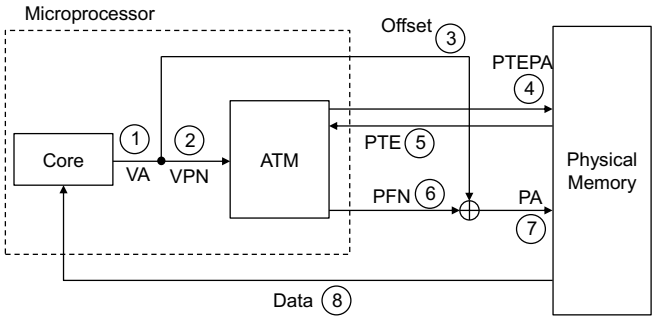

Figure 1: Address translation process overview.

Figure 1 shows a high-level diagram for the address translation process of any microprocessor architecture:

- The CPU core generates a virtual address (VA) ①, which consists of a Virtual Page Number (VPN) ② and an Offset ③ (the Offset shows the position of the word in both the virtual page and the physical frame).
- The ATM calculates the Page Table Entry Physical Address (PTEPA) ④ and fetches the corresponding PTE from the PT in memory ⑤ (this happens when the TLB of the ATM does not already contain the PTE).
- The ATM uses the Physical Frame Number (PFN) ⑥ that the PTE contains along with the same Offset ③ to form the Physical Address (PA) ⑦ and access the memory word ⑧.
- When the ATM's internal TLB already contains the VPN to PFN mapping, steps ④ and ⑤ are skipped and the PA is immediately formed using the TLB contents and the Offset.

B. Design Bugs in ATMs

A design bug refers to a divergence from the expected behavior. The diverging behavior is attributed to a design bug and can appear in any hardware structure involved in address

translation: control registers, translation loockaside buffers (TLBs), segment registers or segment buffers, page table entries (the exact set of hardware components is different for each architecture). Specific categories are also devoted to ATM-related exceptions. We distinguished several ATM-related bugs that slipped into volume production, which are critical for microprocessor's functionality. Table 1 presents a small subset of address translation related bugs in published errata sheets. Bugs in ATMs can be in any form, such as reading or writing a wrong value into a TLB or into a control register, an update to an entry can be fail to complete, so that entry still keeps the stale value, and so on.

III. ATM POST-SILICON VALIDATION

The requirements (and major challenges) for an effective post-silicon validation methodology for ATM hardware in modern microprocessors are the following:

All different address translation scenarios must be excited – This is necessary to guarantee that design bugs are revealed for all modes of operation of the ATM hardware.

Validation programs must be self-checking – Similarly to recent approaches ([16], [22], [37]), ATM bug detection must be performed without resorting to previous known results of the validation programs. Self-checking validation programs de-couple the process from very slow simulations required to generate golden responses as we discuss in Section 1.

Our main contribution is to enhance the ATM validation programs generated by already existing program generators, so that they comply with the self-checking property: ATM bugs detection is signaled by mismatches between the accessed memory items and known (expected by the validation program) values in the memory area being validated. We construct a custom memory image (and corresponding custom translation structures; i.e. page tables) to implement this self-checking mechanism for bug detection. All contributions are ISA-independent and can be applied with minor adjustments to different microprocessor designs. The next subsections provide all details of our methodology.

A. Program Generation

Special purpose verification languages support automatic stimulus generation to enable better specification and design coverage, such as IBM's Genesys-Pro [14] and Synopsys' Vera [33]. These frameworks allow design engineers to express pseudo-random test program generation along with complex event scenarios using generic Test Templates [3], [5], [9]. A Test Template defines any desired verification scenario, and based on the microprocessor's architecture, the corresponding pre-silicon verification or post-silicon validation programs are generated [8].

Figure 2 shows the proposed framework for post-silicon validation of the ATM hardware. As we explain in the next subsection, our methodology does not require previously generated golden responses by a simulator (a major requirement to facilitate the execution of very large numbers of post-silicon validation programs).

The Program Generation Engine (Genesys-Pro like) takes as inputs:

- a Test Template, which specifies the overall silicon validation plan and describes the test scenarios
- the Page Table information (see next subsection)

The above scenarios are translated by the Program Generation Engine into complete post-silicon validation programs. The final self-checking validation programs completely stress and validate the ATM hardware of the microprocessor. Subsequently, the validation programs are loaded into the prototype chip (also referred to as the design under validation – DUV) and the validation process begins.

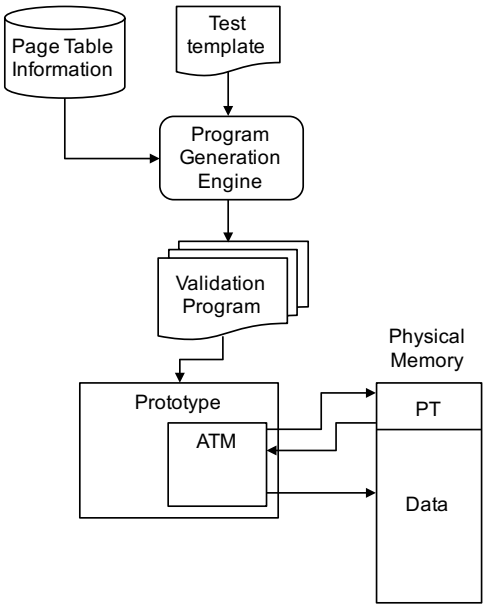

Figure 2: The proposed ATM silicon validation framework.

B. Memory Image

Post-silicon validation of a microprocessor's ATM subsystem is typically performed in a bare-metal configuration, i.e. a single-process validation program is executed at a time directly on the silicon chip without any operating system support[1].

To detect bugs in the ATM subsystem in a bare-metal configuration we need to load a Page Table in the system memory so that the ATM of the microprocessor operates normally and all translation paths are excited. The Page Table contains the VA to PA translations required by the generated validation program. By setting the parameters of the validation program generation flow of Figure 2, we can:

- Cover arbitrary physical memory areas.
- Cover all entries of the ATM hardware structures (TLBs, segment registers, configuration registers, etc.)

[1] Other silicon validation phases focus on full-system configuration with an operating system layer and application software on top of it.

The validation program also contains information about the virtual-to-physical addresses mapping. This is required for the realization of the self-checking property.

Our validation programs are self-checking because *the physical addresses they traverse contain data values equal to the physical address*. Load instructions throughout the validation programs check the correctness of the address translations by simply comparing the loaded value with the known physical address (this is why the validation programs are aware of the virtual to physical address translations).

Before the execution of the validation programs, the physical memory locations that the validation program will go through to validate (other locations than the ones that store the program of course or the Page Table) are stored with data values equal to their actual physical address. For example, if physical address 0x7766554433221100 belongs to the area under validation the data value 0x7766554433221XXX is stored in it. The last 12 bits can have any value since they represent the Offset (in a 4KB page) and do not take place in the address translation process.

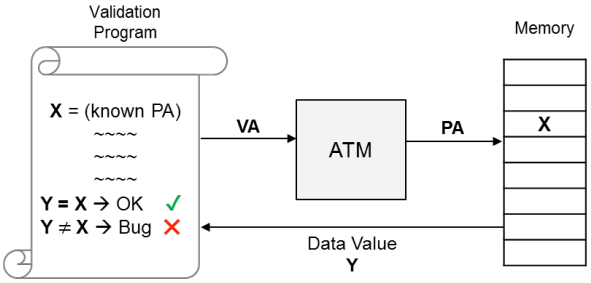

Figure 3: The Validation Program accesses memory through the ATM. The ATM translates the VA to PA and the fetched (pre-stored) data value is returned back to the validation program, which compares it with the expected PA. A mismatch reveals a bug.

Figure 3 outlines the self-checking validation programs operation concept. The data value returned to the program by a load instruction is compared in the program with the known Physical Address (PA) that the ATM should generate. In case of a mismatch in this comparison an ATM bug is detected.

IV. Method Limitations

Due to the self-checking nature of the methodology, ATM-related bugs that do not result to a wrong operation (e.g. wrong translation or incorrect privileges and thus exception) but only lead to performance loss are not detected. For example, if the application asks for a translation that is already cached in the TLB but, due to a bug, a TLB miss happens, the proposed method does not detect it because the correct translation will be returned from the Page Table walk. Of course, such bugs are less severe than bugs that affect correctness.

Another limitation of the proposed method is related to store operations. If the original ATM validation program performs a store that can affect the pre-loaded data values the self-checking property is jeopardized (there will always be a mismatch between the physical address and the stored data in that address without a bug). The validation program process should avoid such cases by tuning the validation program generators.

V. Experimental Evaluation

A. Simulation Setup and Methodology

The proposed post-silicon validation method for microprocessor ATM subsystems can be adapted to any ISA and specific microarchitecture. The baseline validation programs guarantee very high bugs coverage (generated by mature tools [2], [14]). When enhanced with the proposed self-checking mechanism very large numbers of validation programs can be applied to the microprocessor under validation without resorting to simulation-generated golden responses. To further demonstrate the effectiveness of our post-silicon validation method and potential usage scenarios we emulate an actual bare-metal post-silicon validation using the x86-64 model of the Gem5 simulator [36]. We have developed four different validation programs. Two of the validation programs have a more generic scope and validate fundamental principles that apply to any ISAs Virtual Memory and ATM implementations. The other two validation programs are more specialized for the x86-64 ISA and aim to validate ISA-specific (or even design-specific) requirements. We provide details about the programs in the following paragraphs.

We have modified the Gem5 simulator [36] to support injection of design bugs in the address translation structures of the x86-64 model; this way we resemble a realistic "buggy" microprocessor prototype chip. We developed a custom minimal "kernel" to prepare the post-silicon validation "environment" and execute our validation programs. The kernel initiates all the required procedures to set up the paging interface required to operate in Long Mode (x86-64). It also offers a very basic infrastructure for virtual memory mapping. The initial state of the memory (see subsection III.B) is achieved using the simulator's boot loader. A chunk of 32 MB is initialized for the needs of our validation requirements.

B. Validation Programs and Results

Validation Program #1 (VP1 – generic; any ISA): The program starts by mapping different virtual pages (VPs) to physical frames (PFs) of the initialized memory. It then goes through the VP and initiates loads on different offsets. It aims to detect unexpected modifications of the matching TLB entry upon access (right after). The program can only miss the very rare cases when a TLB entry gets corrupted on the last access. For the validation runs of this experiment, Gem5 was modified to corrupt a TLB entry upon its access, after performing a translation, on a rate of 0.01% (an entry corruption every 10,000 accesses), thus emulating the buggy behavior.

Validation Program #2 (VP2 – x86-64 specific): This is an x86-64 ISA-specific scheme to check whether a TLB flush was unexpectedly (due to a bug) initiated. The program first maps a VP to a PF, it requests a TLB flush to update the translation, and then asks for a load on that address, to ensure that the

translation is placed on the TLB. It finally remaps the VP to another PF without flushing the TLB. The TLB should have a stale translation at this point that corresponds to the first mapping. A loop of loads goes through the entire virtual page and compares the read values with the initial translation. If the TLB gets flushed, the translation will be updated to the new mapping and the self-check comparison will report a mismatch. For the runs of this experiment, Gem5 was modified to initiate some unexpected (not normal) TLB flushes, thus emulating the buggy behavior.

Validation Program #3 (VP3 – x86-64 specific): This validation scheme is again x86-64 specific. The program examines if TLB flushes occur upon update of the CR3 register, as described by the ISA specification. The program maps a single virtuial page to a physical frame of the initialized memory. It then requests a TLB flush and performs 16 loads on different offsets of the mapped page. It compares the values with the expected physical addresses. After the 16 loads, it will re-map the page to a different frame and again request a TLB flush. The process goes through the entire 32MB of initialized memory. If the ATM hardware fails to flush the TLB, it will use a stale translation and it will cause a mismatch on the first load instruction of the new page map (offset 0). For the runs of this experiment, Gem5 was modified to skip 1% of the requested TLB flushes, thus emulating the buggy behavior.

Validation Program #4 (VP4 – generic; any ISA): The program intends to detect all of the wrong values delivered by the page walking process. It starts by mapping different VPs to PFs of the initialized memory. It then performs loads on the mapped VA to compare the loaded value with the requested mapping. If the value matches the expected physical address, it moves on to the next frame until going through all of the 32MB of initialized memory. If no mismatches are found, the program reports success. For the runs of this experiment, Gem5 was modified to deliver wrong value bugs during this process in a rate of 0.5%, thus emulating the buggy behavior.

We executed each validation program 1000 times and all reported one or more mismatch in the self-checking comparisons of our method (=bug detection).

Table 2 reports: the number of injected bugs (1000 in each case – 4000 bugs total; second column); the number of virtual pages and physical frames (third and fourth columns) that are covered by the emulated validation process for every different validation program; and the number of self-checking comparisons that each validation program performs (fifth column). The validation programs were also simulated against the golden (unmodified) Gem5 and reported success on the bug-free executions.

Validation programs VP1 and VP2 require a significantly larger number of self-checking comparisons than programs VP3 and VP4. The reason is that programs VP3 and VP4 are designed to detect bugs that are more promptly excited by accesses to the corresponding entries, while programs VP1 and VP2 are designed to detect bugs which require much longer execution before the modified entry is again accessed. Finally, program VP2 requires an excessive number of checks because the entire TLB is flushed compared to VP1.

	Total Bugs	**Virtual Pages**	**Physical Frames**	**Translation Checks**
VP1	1000	7,680	7,680	122,880
VP2	1000	1	7,680 (*remapped frames*)	3,932,160 (*checks for stale translation*)
VP3	1000	1	7,680 (*remapped frames*)	7,680
VP4	1000	7,680	7,680	7,680

Table 2: Experimental results of our silicon validation evaluation on the bare-metal setup in the x86-64 ISA models of Gem5.

VI. Related Work

Recent studies have presented different microprocessor silicon validation methods, such as Instruction-By-Instruction checking [34], Sequence-By-Sequence checking [38], and instruction duplication [37]. Recent approaches have also employed the inherent ISA diversity to generate self-checking silicon validation programs. The original instructions of a validation program are complemented with equivalent instructions, instruction sequences, reverse instructions or diverse data operands [16] [22] [37]. Our work in this paper contributes with self-checking validation programs for the ATM hardware of microprocessors of any ISA.

The work presented in [17] combines self-checking validation programs (which can be generated by any of the previous approaches) with deconfigurable hardware structures to enhance root cause analysis of bugs. The method presented in [19] discusses a post-silicon validation approach for the memory hierarchy in multi-core architectures. Finally, the approach presented in [39] targets the runtime verification of the correctness of address translation for memory consistency, which can be affected by design bugs and physical faults.

VII. Conclusion

Virtual memory is a very mature abstraction, which facilitates easier system and application programming and supports efficient system resources protection among tasks by providing the illusion of an unlimited memory space. Fast virtual-to-physical address translation in a microprocessor requires sophisticated mechanisms and diverse hardware structures, which constantly get more complex as virtualization in computing systems gets popular. To this end, dependable functionality of such mechanism is both very important and challenging task. However, address translation the most difficult subsystem for correctness validation, especially when the chip is taped-out, due to its "transparent" operation in the whole microprocessor (no direct program visible output).

We have presented an ISA-independent post-silicon validation methodology for the Address Translation Mechanisms of modern microprocessors. We present the problem of ATM post-silicon validation by describing a set of published errata in ATM related hardware structures from actual commercial microprocessor chips. We have also presented a post-silicon validation program generation flow that harnesses the detection effectiveness of mature tools employed by all microprocessor vendors; we enhance the

validation programs to be self-checking so that the process is fully de-coupled from time-consuming simulations to produce golden models. Our experimental evaluation in Gem5 x86-64 model demonstrates the effectiveness of the proposed methodology.

ACKNOWLEDGMENTS

This work is supported by the 7th Framework Program of the European Union through the CLERECO Project, under Grant Agreement 611404 and by IBM Research through an IBM Faculty Award. Authors would like to thank Anatoly Koyfman and Vitali Sohkin of IBM Research Labs for their valueable inputs and support.

REFERENCES

[1] T.W.Barr, A.L.Cox, S.Rixner, "Translation Caching: Skip, Don't Walk (the Page Table)", ISCA 2010.

[2] A.Adir, R.Emek, Y.Katz, A.Koyfman, "DeepTrans – A Model-Based Approach to Functional Verification of Address Translation Mechanisms", MTV 2003.

[3] M.Behm, J.Ludden, Y.Lichtenstein, M.Rimon, M.Vinov, "Industrial Experience with Test Generation Languages for Processor Verification", DAC 2004.

[4] Y.A.Katz, A.Koyfman, E.Tsanko, "Method of Verification of Address Translation Mechanisms", US Patent No. 8245164.

[5] A.Nahir, M.Dusanapudi, S.Kapoor, K.Reick, W.Roesner, K.-D.Schubert, K.Sharp, G.Wetli, "Post-silicon validation of the IBM POWER8 processor", DAC 2014.

[6] A.Adir, S.Copty, S.Landa, A.Nahir, G.Shurek, A.Ziv, C.Meissner, J.Schumann, "A Unified Methodology for Pre-Silicon Verification and Post-Silicon Validation", DATE 2011.

[7] T.M.Austin, G.S.Sohi, "High-Bandwidth Address Translation for Multiple-Issue Processors", ISCA 1996.

[8] A.Aharon, D.Goodman, M.Levinger, Y.Lichtenstein, Y.Malka, C.Metzger, M.Molcho, G.Shurek, "Test Program Generation for Functional Verification of PowerPC Processors in IBM", DAC 1995.

[9] L.Fournier, Y.Arbetman, L.Levinger, "Functional Verification Methodology for Microprocessors using the Genesys Test-Program Generator. Application to the x86 microprocessors family", DAC 1999.

[10] A.Koyfman, A.Adir, R.Emek, Y.Katz, M.Vinov, "Modeling Language and Method for Address Translation Design Mechanisms in Test Generation", US Patent No. 7370296.

[11] S.Mitra, S.A.Seshia, N.Nicolici, "Post-Silicon Validation Opportunities, Challenges and Recent Advances", DAC 2010.

[12] J.Goodenough, R.Aitken, "Post-Silicon is Too Late – Avoiding the $50 Million Paperweight Starts with Validated Designs", DAC 2010.

[13] H.Rotithor, "Postsilicon Validation Methodology for Microprocessors", IEEE Design & Test of Computers, 2000.

[14] A.Adir, E.Almog, L.Fournier, E.Marcus, M.Rimon, M.Vinov and A.Ziv, "Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification", IEEE Design & Test of Computers, 2004.

[15] B.Jacob and T.Mudge, "Virtual Memory: issues of implementation", IEEE Computer, 1998.

[16] N.Foutris, D.Gizopoulos, M.Psarakis, X.Vera and A.Gonzalez, "Accelerating Microprocessor Silicon Validation by Exposing ISA Diversity", MICRO 2011.

[17] N.Foutris, D.Gizopoulos, X.Vera and A.Gonzalez, "Deconfigurable Microprocessor Architectures for Post-silicon Validation Acceleration", ISCA 2013.

[18] A.Wolfe, AMD's Quad-Core Barcelona Bug Revealed, Information Week, Dec. 2007.

[19] A.DeOrio, I.Wagner, V.Bertacco, "Dacota: Post-Silicon Validation of the Memory Subsystem in Multi-Core Designs", HPCA 2009.

[20] W.Kadry, A.Koyfman, D.Krestyashyn, S.Landa, A.Nahir, V.Sokhin, "Improving Post-silicon Validation Efficiency by Using Pre-generated Data", Haifa Verification Conference 2013.

[21] B.Bentley, "Validating the Intel® Pentium® 4 Microprocessor", DAC 2001.

[22] I.Wagner, V.Bertacco, "Reversi: Postsilicon Validation System for Modern Microprocessors", ICCD 2008.

[23] AMD. Revision Guide for AMD Athlon64 and AMD Opteron Processors. Publication 25759, Revision 3.79, July 2009.

[24] AMD. Revision Guide for AMD Family 10h Processors. Technical Report 41322, Revision 3.92, March 2012.

[25] IBM. IBM PowerPC 750FX and 750FL RISC Microprocessor Errata List DD2.X, version 1.4, Feb. 2008.

[26] Intel Corporation. Intel Core Duo Processor and Intel Core Solo Processor on 65nm Process Specification Update. Technical Report 309222-020, June 2009.

[27] IBM. IBM PowerPC 750GX and 750GL RISC Microprocessor Errata Notice DD1.X, version 1.8, Jan. 2006.

[28] Intel Corporation. Intel® Pentium® 4 Processor Specification Update. Document Number 249199-071, Aug. 2008.

[29] Intel Corporation. Intel® Xeon® Processor 5100 Series Specification Update. Document Number 313356-026, Dec. 2012.

[30] Intel Corporation. Intel® Xeon® Processor 3400 Series Specification Update. Document Number 322373-009, May 2010.

[31] Intel Corporation. Intel® Xeon® Processor E3-1200 v3 Series Specification Update. Document Number 328908-011, April 2015.

[32] ARM® Cortex-A57 MPCore Processor: Technical Reference Manual, Revision r1p2, 2014.

[33] F. Haque, J.Michelson, and K.Khan, "The Art of Verification with Vera", Verification Central, 2001.

[34] D.Chatterjee, A.Koyfman, R.Morad, A.Ziv, V.Bertacco, "Checking Architectural Outputs Instruction-by-Instruction On Acceleration Platforms", DAC 2012.

[35] D.Lin, T.Hong, F.Fallah, N.Hakim and S.Mitra, "Quick Detection of Difficult Bugs for Effective Post-Silicon Validation", DAC 2012.

[36] N.Binkert, B.Beckmann, G.Black, S.K.Reinhardt, A.Saidi, A.Basu, J.Hestness, D.R.Hower, T.Krishna, S.Sardashti, R.Sen, K.Sewell, M.Shoaib, N.Vaish, M.D.Hill, and D.A.Wood, "The gem5 simulator", SIGARCH Comp. Arch. News, 2011.

[37] T.Hong, Y.Li, SB.Park, D.Mui, D.Lin, Z.A.Kaleq, N.Hakim, H.Naeimi, D.S.Gardner and S.Mitra, "QED: Quick Error Detection Tests for Effective Post-silicon Validation", ITC 2010.

[38] C.-H.Hsu, D Chatterjee, R.Morad, R.Gal, V.Bertacco, "ArChiVED: Architectural Checking via Event Digests for High Performance Validation", DATE 2014.

[39] B.F.Romanescu, A.R.Lebeck, D.J.Sorin, "Specifying and Dynamically Verifying Address Translation-Aware Memory Consistency", ASPLOS 2010.