Large-scale Application of Formal Verification: From Fiction to Fact

Viresh Paruthi
IBM Systems and Technology Group, Austin TX, USA.
e-mail: *vparuthi@us.ibm.com*

***Abstract*—Formal verification has matured considerably as a verification discipline in the past couple of decades, becoming a mainstream technology in industrial design and verification methodologies and processes. In this paper we chronicle the evolution of formal verification at IBM from being a specialized side activity with a narrow focus, to achieving a broad-based usage as a core verification technology helping to significantly improve design and verification productivity. We showcase what is possible in the application of formal verification in a commercial/industrial setting by highlighting the success we had in leveraging the technology extensively on IBM's POWER7^{TM} microprocessor and systems. We touch upon the methodology and execution aspects of the unprecedented use of formal verification on the POWER7 program, and depict ways in which the technology positively impacted pre-silicon design quality and facilitated root causing of bug escapes to silicon. Furthermore, we outline where we see applied formal verification evolving towards at IBM, and the challenges thereof.**

I. Introduction

IBM has a rich history developing robust formal and semi-formal verification technologies, and applying those effectively to the verification of microprocessor designs and systems. Since its advent almost a decade and a half ago Formal Verification (FV), inclusive of functional formal verification and sequential equivalence checking, has evolved from being a specialized technology in the hands of experts, to a widely deployed technology with a broadened user base to include design and functional verification engineers.

Functional Formal Verification (FFV) at IBM dates back to the POWER3 (1996) microprocessor where it was applied on an experimental basis, followed by a larger and more defined effort on the POWER4 [13] program. On both of these projects the application was limited to small-sized logic partitions requiring the creation of intricate testbenches comprising complex "environmental assumptions". Dramatic improvements to the FFV toolset, and the debut of semi-formal technologies, allowed for increased application and leverage on subsequent programs such as POWER5 [21] and POWER6. The mode of application in all of these programs was similar with FFV being a standalone side activity driven by skilled formal verification engineers - albeit with scaling to bigger logic partitions, and greater portions of the chip logic subjected to FFV analysis. Sequential Equivalence Checking (SEC) technology [4] became available around the 2004 time frame and quickly became a huge productivity advantage. It facilitated proving non-functional design changes (e.g., timing, power) without the need to rerun (lengthy) regression buckets, and enabled key new methodologies (e.g., sequential synthesis, infer clock-gating opportunities).

The mandate coming into the POWER7 program, based on analysis of bugs on past projects, was to apply formal verification technology more extensively to improve pre-silicon design quality and minimize bug escapes into silicon. The result was a step function of integrated and broader usage resulting in the largest and most successful ever application of FV on any project at IBM. FV assumed a central role in the verification of large parts of the design culminating in flushing out hundreds of bugs, many of which would have been extremely difficult to find using traditional verification methods, and ensuring correctness of the logic by way of obtaining proofs. FV was exploited at all levels of the design hierarchy encompassing all areas of the chip. Such a widespread use of the technology has been enabled by IBM's suite of state-of-the-art formal and semi-formal verification tools, SixthSense [18] and RuleBase PE [5], [22], which are fully integrated into the methodology.

In this paper we describe large-scale application of functional formal and semi-formal verification and sequential equivalence checking with experiences from leveraging the technologies on the POWER7 microprocessor and systems. POWER7 [11] is a complex high-end eight-core processor chip with four-way Simultaneous Multi-Threading (SMT4) per core, scalability to 32 sockets, and an aggressive memory subsystem design. It implements a modular structure with heavy use of asynchronous interfaces, and new power-management and RAS (Reliability, Availability, and Serviceability) mechanisms across the chip and system.

FFV and SEC application on the project can be best summarized as a combination of an up-front defined methodology, and results from deploying the methodology during project execution. We start with a brief description of the verification methodology as it relates to FV in the next section. We then outline aspects of FFV and SEC application on POWER7 and the benefits realized in sections 3 through 6. We conclude with glimpses into our strategy for further leveraging FV to address future challenges.

II. Verification Methodology

The base of all verification disciplines inclusive of simulation, hardware accelerated simulation and formal and semi-formal verification is a cycle-based execution model of the design under test (Figure 1). Having one single, consistent interpretation model of the RTL specification, regardless of

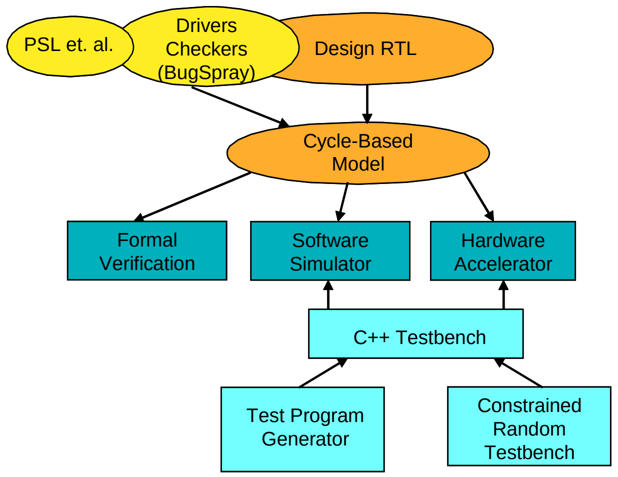


Figure 1. Core Methodology Flow

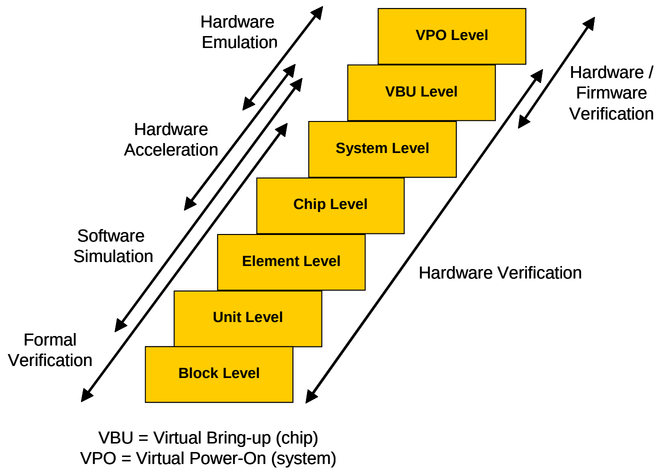


Figure 2. Verification progression

which tool or technology is used for a particular verification task enables reuse of verification objects (assertions, coverage events) and results (e.g., waveform signatures) across disciplines. Such a tight integration coupled with dramatically increased model capacity of the formal and semi-formal toolset has allowed the application of the technology to span designer-level verification, deep verification of design blocks, sequential equivalence checking and formal verification of large design partitions like the complete Floating-Point Unit (FPU) dataflow.

Internally IBM uses an extension of VHDL for functional coverage and assertion instrumentation, called *BugSpray*. BugSpray is used by the design and verification teams alike to efficiently annotate the RTL with assertion and coverage events. BugSpray enables verification objects to be portable across verification disciplines and across hierarchies, and allows for their reuse with design. For example, majority of the coverage events are provided by the design team with the goal of grading the verification effort given their knowledge of the design implementation. In addition, Property Specification Language (PSL) [9], standardized as IEEE 1850, may be used for the purposes of design instrumentation.

Extensive verification is undertaken at all levels of the design hierarchy [13], depicted in Figure 2. Verification at the lower levels of the hierarchy tends to be more productive due to the smaller size of the design under test, and greater controllability of the interfaces yielding higher state coverage and exploration of corner cases/boundary conditions. Verification objects from lower levels are selectively enabled at the higher levels. Formal methods are leveraged at various levels of the design/verification hierarchy to achieve different goals.

At the *block level* FFV is applied widely to prove design components. This may be driven in part by the designers themselves by way of assertion-based verification, and FFV environments inherit all of the designer assertions and coverage events. The downside of verifying logic at the block level is the need to model complex interface interactions between logic blocks requiring intricate testbenches (environment assumptions and properties), and coping with churn at those interfaces as the design evolves. Clear and precise documentation at this level may be somewhat lacking, making verification a more laborious process.

Increased speed and capacity of formal and semi-formal verification toolset has allowed FFV to scale to the *unit level* selectively. This enables taking advantage of relatively stable and well-documented interfaces, creation of simpler constraint-based [20], [16] drivers, and focus on more encompassing micro-architectural properties. The checking may entail creation of a reference model to equivalence check the implementation against, such as the IEEE floating point specification to verify the FPU, or specify a rich set of properties to constitute a specification of the unit as a whole.

The *element level* comprises design elements such as the processor core, cache and memory sub-system. FFV is primarily employed to verify multi-unit interactions and architectural aspects at this level, for instance hangs and stalls, starvation, bus protocols. We endeavor to reuse simulation RTL models by exposing only the logic of interest and effectively deleting logic not pertaining to the verification task at hand. Portions of the logic may need to be abstracted, and replaced with behavioral models to reduce logic size and complexity. The *chip level* incorporates multiple processor cores along with interconnect and storage sub-system, and the *system level* consists of multiple chips, memory and I/O chips per actual machine configurations. At these levels high level mathematical reasoning, manual proof techniques, and specialized models (e.g. Murphi model [8]) are used to verify features such as chip/system deadlock/livelock, cache and memory coherence, message routing and traffic flows across (asynchronous) interfaces.

"Pervasive" logic (e.g., initialization, scan/debug, RAS, power-management) which may span block, unit, element and chip boundaries poses numerous challenges to verification as it can be sequentially very deep, and may have large numbers of inputs to be verified effectively with simulation. FFV has demonstrated strength in this domain [12], and is applied at all levels of the hierarchy to verify complex pervasive logic.

Because the cost of finding a bug is lower at lower levels of the hierarchy every major bug found at a higher level is treated as an escape of the lower levels, and every attempt is made to reproduce it at the lower levels. This helps to “harden” the lower level environments and make them more resilient. This translates into formulating design assertions at the block level to expose the flaw with FFV wherever applicable, and prove conclusively that the logic fix fixes it.

In a similar fashion, FFV is extensively applied to recreate post-silicon test floor failures at the block level, and verify fixes thereof. Typical fails on the test floor require many events to line up (as otherwise the problems would have been caught in pre-silicon verification), and it is non-trivial to produce the sequence of events leading up to the fail with higher level environments as those don’t have direct control over the interfaces coming into a logic block. FFV has unique strengths to quickly root cause a defect once it is understood and the general area of the logic exhibiting the failure has been localized.

III. Integrated Approach

The cornerstone of large-scale application of formal and semi-formal verification is the pursuit of an integrated approach with design and simulation.

In the past FFV was a side activity with an execution plan owned and managed solely by the FFV team working closely with the designers to infer correctness properties, and to obtain interface specs to facilitate creation of FFV environments. The apparent disconnect with simulation-based verification can be attributed to a focus on verification at the block level with FFV, and at higher levels of the design hierarchy with simulation. This permitted a limited interaction with simulation teams, and coordination of the overall verification process across disciplines.

We changed all that on POWER7 by positioning FFV synergistically alongside simulation and making unit verification teams responsible for defining and owning (project managing) FFV plans based on their respective needs. This allowed for the plan to be dynamically altered, addressing specific requirements and deficiencies in real time, to make FFV application more effective. It was our endeavor to apply FFV early on complex logic blocks identified for formal checking in an attempt to provide value-add upfront to complement simulation-based verification. The widespread application of FFV at various levels of the hierarchy furthered this interlock. A rigorous process was instituted whereby FFV environments were reviewed closely with designers, architects, FV experts and functional verification engineers to ensure completeness of the checking and correctness of the assumptions.

The portability of synthesizable coverage and assertion specification between simulation, hardware accelerated simulation and formal and semi-formal verification was critical to our approach to cross-link the different verification efforts more effectively (Figure 3). It enabled us to aggressively drive an assertion-based verification paradigm which brought together designer-level verification, formal verification and simulation in a unified integrated methodology. For example, designer assertions and coverage events used to grade simulation environments were utilized in formal verification

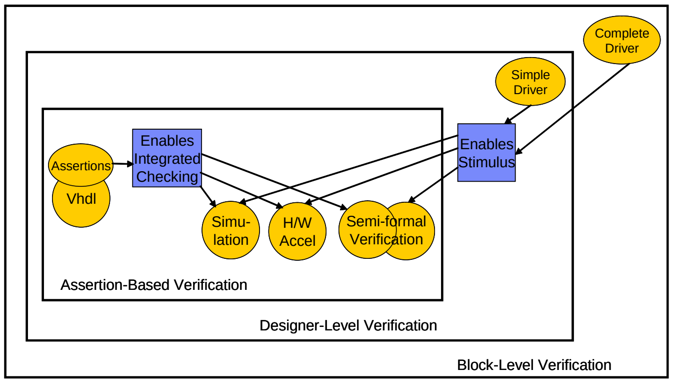


Figure 3. Integrated (with Design) Approach

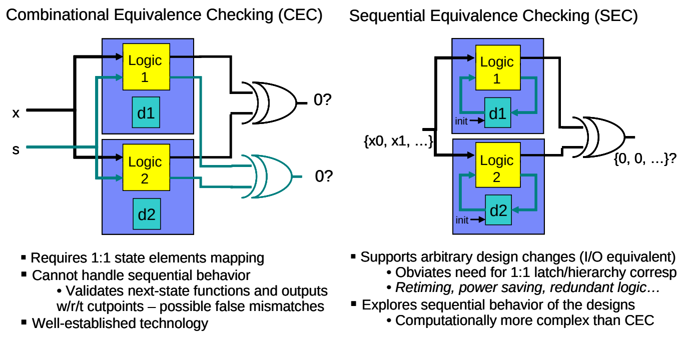
environments. In turn, formal environment checkers (properties) and assumptions were added to the simulation environment to be cross-checked and cross-leveraged. Artifacts of simulation-based verification, such as biases to specify relative probabilities of inputs switching (e.g., a reset line should not be active frequently as it would restart the simulation effectively), or initializations based on machine configurations, are taken advantage of in formal and semi-formal verification (as applicable) to make it more productive.

We have been pushing for a broad adoption of FFV natively by design and verification teams and achieved significant progress on POWER7. FFV was leveraged extensively by designers as an assertion-based verification vehicle to check their designs before making them available to verification teams. This helped to improve productivity significantly by breaking the costly cycle of: designer checks RTL code into the repository, simulation runs with the design and uncovers errors and logs issues, the designer fixes the design and again makes it available. In a number of cases designers developed comprehensive block level FFV environments to prove the design. For others the designers inherited the environments from FFV experts and took ownership to continue to regress with those, and enhance the environments as needed. Simulation verification teams, for their part, took advantage of FFV to fill gaps in their verification testplans - e.g. chip-wide networks to transmit debug information which require large numbers of patterns to verify with simulation are easy for FFV to handle.

IV. Demonstrated Best Practices

We continue to leverage and extend successful applications of FFV from past projects. The POWER7 program saw substantial deployment of these proven engagements.

Complex logic blocks on the different units were identified and prioritized for a targeted “deep dive” verification by FV experts [13], [21]. The core strengths of block level verification are the small size of the design under test, and direct control on testing as the driving stimulus is applied directly to the interfaces of the block with no “filtering” effect of upstream logic. The former translates into proofs to ensure correctness of the aspects verified. The latter facilitates exploring all areas of the interface’s state space equally easily, whereas some of those areas would have been exercised rarely in the larger


Combinational Equivalence Checking (CEC)
Sequential Equivalence Checking (SEC)
x
s
Logic 1
d1
Logic 2
d2
0?
0?
{x0, x1, ...}
init
{0, 0, ...}?

- Requires 1:1 state elements mapping
- Cannot handle sequential behavior
  - Validates next-state functions and outputs w/r/t cutpoints – possible false mismatches
- Well-established technology

- Supports arbitrary design changes (I/O equivalent)
  - Obviates need for 1:1 latch/hierarchy corresp
  - *Retiming, power saving, redundant logic…*
- Explores sequential behavior of the designs
  - Computationally more complex than CEC

Figure 4. Combinational and Sequential Equivalence Checking

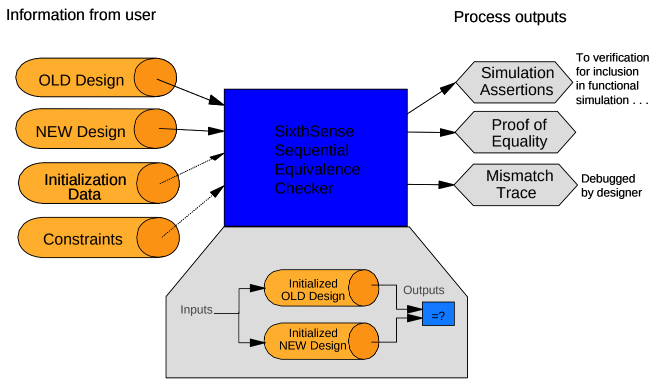


Figure 5. Sequential Equivalence Checking Set-up

context - e.g., with the block connected in the unit. This is invaluable to exercise low level interactions and window conditions given challenging micro-architecture features (e.g., simultaneous multi-threading, out-of-order execution).

The blocks are chosen in consultation with design and verification teams based on their complexity, difficulty in verifying them with other verification disciplines, and logics that have exhibited late bugs on previous projects. Block level verification ranges from white-box checking of the logic by FFV experts based on a deep understanding of the micro-architecture (e.g., instruction prefetch/fetch, memory management unit), to end-to-end checking to verify conformance of the function against a specification (e.g., queues, error correcting codes), to verification of the algorithmic correctness of the logic (e.g., least recently used cache replacement). Our endeavor is to make the verification high level to the extent possible by formulating properties that are conceptual/architectural, hence independent of the implementation. Application areas include all areas of the chip (core units, caches, pervasive) and hierarchies (blocks, unit, element, chip).

POWER7 saw more blocks verified on more functional units than on any previous project.

Proven and established methodologies were utilized to verify (complete) function of logics such as FPU dataflow [14] and arithmetic functions (e.g., adder, multiplier, divider), by comparing the high performance implementation against a high level reference model. We expanded the list of pervasive logics verified with FFV building atop past successes [12] to include additional areas (e.g., chip-wide sensor networks), and created automated methodologies to improve productivity – e.g., Debug Bus verification via automated testbench creation given a specification in a custom language.

In a number of cases FFV was the technology of choice to be relied upon heavily/solely to ensure correctness of, often times critical, logic (e.g., arithmetic dataflow, arbitration, least recently used). The environments created to verify blocks proved very useful later to quickly root cause post-silicon problems seen on the laboratory floor. FFV has assumed a central role over the years in triaging bugs in silicon, and assuring the correctness of the logic fixes.

V. Sequential Equivalence Checking

Sequential transformations are widespread in hardware design flows to address needs such as performance, power, area, debug and test. Established frameworks such as Combinational Equivalence Checking (CEC) are unable to handle such sequential changes as it requires a 1:1 pairings of the state elements. Industrial demand for robust Sequential Equivalence Checking (SEC) solutions is thus becoming increasingly prevalent. SEC is a paradigm to help offset the limitations of CEC (Figure 4). SEC performs a true sequential check of input/output equivalence, hence is not limited to operation on designs with 1:1 state element pairings.

SEC [4] is widely deployed at different stages of the project to achieve various goals, such as verify non-functional design changes (e.g., power, timing, area), verify external IP conversion over to IBM's clocking and latching methodology, ensure mode latches indeed revert the design back to a previous function. SEC has also been key to several new methodologies which leverage the power of sequential transformations [25], [10], [4].

With POWER7 we made this easy-to-use yet powerful technology available in the hands of the designers to improve productivity substantially by obviating the need to rerun costly regressions to verify non-functional changes to the design. In later stages of the project when all function was completed, we instituted a rigorous end-to-end SEC process (Figure 6) starting at the macros and working its way up to the chip level (with black-boxing lower levels of the hierarchy) to establish that inadvertent functional changes did not get introduced in subsequent releases of the RTL. This facilitated avoiding (tools and technology) capacity issues with running large partitions, and enabled designers to run SEC on interfaces/design sections they are the most familiar with. It helped to improve the debug cycle by way of producing short, precise and localized mismatch traces, as opposed to needing large numbers of cycles before mismatches propagate to an observable outputs.

The process is flexible enough to permit definition of hierarchies to run SEC at, which may or may not align with the design hierarchy. This allows to verify behavior-preservation of changes across design entities, such as moving logic between entities with bundling the entities together in a custom wrapper, which is then equivalence checked and black-boxed at higher levels.

Any assumptions required to get the equivalence check to succeed at any level of the hierarchy are independently

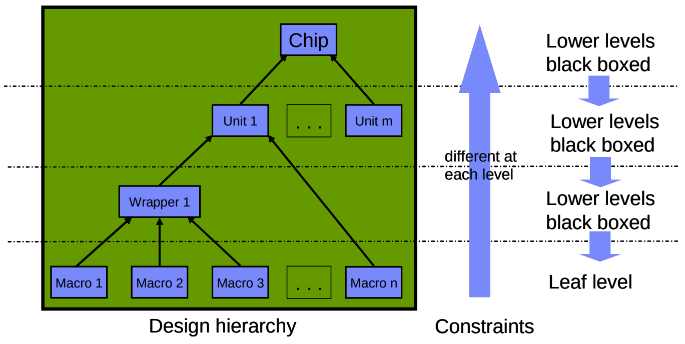


Figure 6. Hierarchical Decomposition

validated with functional verification by converting the constraints into assertions (Figure 5) which are then pulled into the simulation/FFV environments.

This end-to-end SEC process has proved to be an invaluable tool on multiple projects to verify remaps of the design to a newer technology without the need to set-up functional verification environments yielding huge resource savings, productivity improvement and fast turnaround. Low level functional verification environments are done away with completely and a few higher level ones are used for the purposes of a quick sanity check and to verify SEC assumptions – the portability of assertions across design hierarchies allows those to be carried to higher level verification environments.

VI. New Exploration Areas

We continue to push the envelope to scale FV application to areas which hitherto have been beyond the scope of FV. We endeavored to push the (capacity) limits of FV application by exploring several new areas on the POWER7.

We pursued creation of (constraint-based) environments at higher levels of abstraction, for example (sub-) unit level, to take advantage of well-defined interfaces, and increase interplay with simulation. The latter is achieved by leveraging FFV to weed out unreachable coverage events, or obtain hints to hit tough coverage events to help enhance simulation test buckets, and dramatically increase state-space coverage via semi-formal verification.

We investigated verification of system level aspects in several contexts. These were, for example, timing protection windows to ensure coherence by enumerating system topologies and studying multi-chip interactions, and deadlock free operation of the system using direct mathematical analysis on dedicated high level models. These efforts helped flush out architectural issues early on, and assisted in setting-up effective simulation testbenches to check for violations. As another example, asynchronous interfaces are a formidable challenge to functional verification due to the fact that they are not exercised adequately at the system level. We attempted to reason about the effect of asynchronous interfaces (e.g., unpredictable traffic flow across the interface may manifest as buffer overflows) by modeling these interfaces using system level models with design details suitably abstracted.

We pioneered various innovative and reusable techniques/methodologies by way of creating "off-the-shelf" verification IP which can be applied out-of-the-box for similar logics on other parts of the chip, or on future products. Examples include a method to expose starvation in complex arbitration logics using successive property strengthening and underapproximations [1], and to verify correctness and performance of such arbiters by accurately computing request-to-delay bounds and ascertaining the fairness requirements of the arbitration scheme [15]; systematic methods to verify complex 64Byte error correcting codes, least recently used replacement and hardware data structures such as linked lists, queues, buffers, etc.

Block level FFV testbenches may require modeling complex interfaces which can make them non-trivial and error prone. Specifying those at higher levels of abstraction can help alleviate this to a great extent. Towards this goal we undertook the creation of a rich library of functions (implemented as VHDL packages) to enable specifying testbenches at higher levels. The parameterized functions encapsulate commonly used logic constructs (such as counters, zero/one-hot detectors/generators, oscillators, biased non-deterministic generators, to list a few) and synthesize correct-by-construction logic implementing the functionality. While it is desirable to raise the level of abstraction of the design logic itself, it is not-so-easy for optimized custom logic used in high-end microprocessors, more so given the interwining with "non-mainline" functions such as pervasive logic. We are selectively applying the concept of functions pursued in the context of testbenches to the design domain by creating a library of functions optimized with respect to desired features such as area, timing, logic depth.

VII. Future Directions

With having established FFV and SEC firmly as a mainstream verification discipline integrated in the design and verification methodology, we expect to derive increasing leverage from its application on future projects. Following are example areas where we foresee investments.

We plan to build upon the integrated approach further and position formal and semi-formal platform as the technology of choice for Designer-level Verification (DLV). Towards this goal we have enhanced the technology and the supporting infrastructure to cover the entire spectrum of DLV from block level simulation with applying deterministic patterns to study input-output behavior, to selectively randomizing signals, to creating comprehensive FFV environments to reason conclusively about assertions and coverage events.

Given the successes and the mind share FFV has to be an effective verification paradigm, we will continue down the path of "booking" the verification of more logics in FFV, and take those off the simulation plate altogether. This allows to maximize productivity across the various disciplines by avoiding duplicate work. We have undertaken a detailed analysis of testplans across FFV, simulation and performance verification to optimize them by making trade-offs with regard to what aspects of the logic get checked where, with the intent of taking maximal advantage of the strengths of the various technologies.

We expect to build upon the theme of "off-the-shelf" checkers to verify logics in an implementation agnostic manner. We have created a persistent compendium of verification IP and we plan to generalize it to verify the logics end-to-end with high level micro-architectural and architectural checkers. In some cases we are attempting to package the IP as a parameterized library which can be applied easily and productively to logics implementing the same function. We will continue to evolve reusable and automated methodologies to make verification of certain logics push-button.

More and more verification transcends checking for functional correctness of the logic to include aspects such as performance, throughput, power, etc. FFV has unique strengths to be able to provide insights into the various aspects by approaching the verification task in a unified manner. An example is the combined verification of performance and correctness of arbiters as described in [15] by establishing an upper bound on the request-to-grant delay. Another example is examining traffic throughput across an (asynchronous) interface to decide on machine configurations/settings such as to not clog buffers on the receive side and queue up traffic in the system. Our goal is to leverage FFV and SEC to provide value add beyond checking for correctness of the logic, including as a general purpose reasoning engines to enable new methodologies.

It is our endeavor to pursue a "formal design" paradigm, especially for newly designed logics, to evolve methods to guarantee its correctness. This can be achieved by formally verifying a high level model independently, and ensuring the implementation conforms to the verified model by virtue of an equivalence check. The high performance RTL may be derived from the high level model via automated (iterative) transformations which are verified with SEC. Alternatively, we may decompose the design into smaller modular pieces in a manner such that each piece can be reasoned about exhaustively standalone, and the proofs of the individual pieces imply correctness of the logic.

We continue to expect to innovate and evolve methods to scale to the complex logics showing up on the next generations of our systems, e.g., wide operand non-linear arithmetic such as used in cryptography accelerators. The power of theorem proving augmented with model checking [24] is a key ally in scaling to such tough problems, as demonstrated in [23].

System level issues such as deadlocks/livelocks are a particular concern on large multi-processor systems as simulation methods cannot produce the kinds of traffic the hardware would experience, and the kinds of interactions between the various traffic sources as a consequence. High level analysis and ways to model traffic in such multi-chip systems, especially given asynchronous interfaces, to study traffic flows or lack of forward progress is an effective method to uncover problems, and will be a focus area.

Significant improvements to the speed and capacity of the formal and semi-formal toolset in the form of improvements to the core engines [17], [3], [19], addition of new algorithms [7], [6] and significant features (e.g., native array support [2]), has enabled FV to address the outlined applications. We continue to expect to see rapid advances to scale to future challenges.

Acknowledgments

The authors would like to thank the members of the FV community at IBM for their relentless pursuit of leverage from the technology. Thanks to Wolfgang Roesner, Klaus-Dieter Schubert, Jason Baumgartner and Ali El-Zein for helping shape FV methodology, and to members of the FFV and SEC team, Gadiel Auerbach, Mark Firstenberg, Paul Roessler, Jo Lee, Shiri Moran, David Levitt, Fady Copty, Steven German, Krishnan Kailas for their contributions to the POWER7 effort.

References

[1] G. Auerbach, F. Copty, and V. Paruthi. Formal verification of arbiters using property strengthening and underapproximations. In *FMCAD*. IEEE, Oct. 2010.

[2] J. Baumgartner, M. Case, and H. Mony. Coping with Moore's law (and more): Supporting arrays in state-of-the-art model checkers. In *FMCAD*. IEEE, Oct. 2010.

[3] J. Baumgartner, H. Mony, and A. Aziz. Optimal constraint-preserving netlist simplification. In *FMCAD*, Nov. 2008.

[4] J. Baumgartner, H. Mony, V. Paruthi, R. Kanzelman, and G. Janssen. Scalable sequential equivalence checking across arbitrary design transformations. In *ICCD*. IEEE, Oct. 2006.

[5] S. Ben-David, C. Eisner, D. Geist, and Y. Wolfsthal. Model checking at IBM. *Formal Methods in System Design*, 22(2):101–108, 2003.

[6] M. Case, A. Mishchenko, R. Brayton, J. Baumgartner, and H. Mony. Invariant-strengthened elimination of dependent state elements. In *FMCAD*, Nov. 2008.

[7] M. Case, H. Mony, J. Baumgartner, and R. Kanzelman. Enhanced verification through temporal decomposition. In *FMCAD*, Nov. 2009.

[8] X. Chen, S. German, and G. Gopalakrishnan:. Transaction based modeling and verification of hardware protocols. In *FMCAD*. IEEE, Nov. 2007.

[9] C. Eisner and D. Fisman. *A Practical Introduction to PSL*. Integrated Circuits and Systems. Springer-Verlag, 2006.

[10] C. Eisner, A. Nahir, and K. Yorav. Functional verification of power gated designs by compositional reasoning. In *CAV*, July 2008.

[11] Wikipedia The Free Encyclopedia. Power7. http://en.wikipedia.org/wiki/POWER7.

[12] T. Gloekler, J. Baumgartner, D. Shanmugam, R. Seigler, G. Huben, H. Mony, P. Roessler, and B. Ramanandray. Enabling large-scale pervasive logic verification through multi-algorithmic formal reasoning. In *FMCAD*, Nov. 2006.

[13] J. Ludden et al. Functional verification of the POWER4 microprocessor and POWER4 multiprocessor systems. *IBM Journal of Research and Development*, Jan. 2002.

[14] C. Jacobi, K. Weber, V. Paruthi, and J. Baumgartner. Automatic formal verification of fused-multiply-add FPUs. In *DATE*, March 2005.

[15] K. Kailas, V. Paruthi, and B. Monwai. Formal verification of correctness and performance of random priority-based arbiters. In *FMCAD*. IEEE, Nov. 2009.

[16] H. Mony, J. Baumgartner, and A. Aziz. Exploiting constraints in transformation-based verification. In *CHARME*, Oct. 2005.

[17] H. Mony, J. Baumgartner, A. Mishchenko, and R. Brayton. Speculative-reduction based scalable redundancy identification. In *DATE*, Apr. 2009.

[18] H. Mony, J. Baumgartner, V. Paruthi, R. Kanzelman, and A. Kuehlmann. Scalable automated verification via expert-system guided transformations. In *FMCAD*, Nov. 2004.

[19] V. Paruthi, C. Jacobi, and K. Weber. Efficient symbolic simulation via dynamic scheduling, don't caring, and case splitting. In *CHARME*, 2005.

[20] Carl Pixley. Integrating model checking into the semiconductor design flow. In *Electronic Systems Technology & Design*, 1999.

[21] R. Gott and J. Baumgartner and P. Roessler and S. Joe. Functional formal verification on designs of pseries microprocessors and communication subsystems. *IBM Journal of Research and Development*, July. 2005.

[22] IBM Research. Rulebase parallel edition. https://www.research.ibm.com/haifa/projects/verification/RB_Homepage/.

[23] J. Sawada. Automatic verification of estimate functions with polynomials of bounded functions. In *FMCAD*. IEEE, Oct. 2010.

[24] J. Sawada and E. Reeber. Acl2six: A hint used to integrate a theorem prover and an automated verification tool. In *FMCAD*. IEEE, 2006.

[25] A. Seigler, G. Van Huben, and H. Mony. Formal verification of partial good self-test fencing structures. In *FMCAD*. IEEE, Nov. 2007.