Connecting Pre-silicon and Post-silicon Verification

Sandip Ray
Department of Computer Sciences
University of Texas at Austin
sandip@cs.utexas.edu

Warren A. Hunt, Jr.
Department of Computer Sciences
University of Texas at Austin
hunt@cs.utexas.edu

***Abstract*— We present a framework for post-silicon analysis, that provides a formal, bidirectional communication with pre-silicon verification. We show how to exploit the framework to provide a formal guarantee on post-silicon verification accuracy under limited observability. In particular, we partition a pre-silicon assertion checker (with full observability) into (1) a limited-observability checker and (2) an in-silicon integrity unit. The composition of the two units is guaranteed to provide the same accuracy as a pre-silicon checker. We apply the framework in the verification of a cache system.**

I. Introduction

Hardware verification research has traditionally focused on pre-silicon analysis. Unfortunately, the complexity of modern systems precludes catching all design errors at pre-silicon; consequently, post-silicon verification is a key component of the verification tool-flow. Post-silicon verification uses first-pass fabricated silicon to catch design errors missed at pre-silicon. Post-silicon simulations run at target clock speeds, permitting exploration of deeper design states than afforded in pre-silicon. Unfortunately, post-silicon verification is constrained by observability: only a few of the thousands of internal signals are observable during normal chip operation. The situation is exacerbated by the trend away from bus-based architectures towards point-to-point links with no central observation points [1]; many of the links exist within sockets encasing multiple processor cores and are not observable.

Post-silicon verification in industry is relatively isolated from pre-silicon. Current observability enhancement techniques entail on-chip instrumentation or hooks to monitor internal registers or bus transactions [2], [3]; however, the hooks are typically added without analysis of design invariants.

The goal of our research is to develop a unified framework connecting pre- and post-silicon analysis. We model hardware designs using a formal Hardware Description Language (HDL) deeply embedded in a formal logic. The netlist is a constant with semantics specified by a formal evaluator. This deep embedding permits analysis of functional and non-functional properties of the same design by associating different evaluators for the logical constant: our HDL has evaluators for (1) functional evaluation, (2) And/Inverter Graph, (3) gate delay, and (4) information flow [4]. We view the design artifact as a database with associated annotations, specifications, and mechanically checked theorems, providing an integrated environment for property checkers, coverage monitors, and regression and analysis routines. The database is initialized with pre-silicon results, and is interrogated during post-silicon verification. The connection between pre- and post-silicon is bidirectional: results from post-silicon augment the database, facilitating further pre-silicon analysis. To our knowledge, our work represents the first connection between pre- and post-silicon verification within a unified logical foundation.

This paper discusses one component of this framework, *e.g.*, the use of formalized assertion checkers and coverage monitors to transfer verification collateral from pre- to post-silicon. We show how to partition a pre-silicon check (with full observability) into (1) an on-chip *integrity unit*, and (2) an external unit. The integrity unit has full observability, but must have little hardware overhead. The external unit has partial observability but can assume that in-silicon analysis has succeeded. The two units are mechanically certified to provide the same guarantee as the original check. The framework provides a disciplined chip instrumentation mechanism with a formal guarantee on verification accuracy. The framework is being mechanized using the ACL2 theorem prover.

A. A Trivial Example

As an example of partitioning a property check, consider a split-transaction bus, and consider checking that the number of outstanding memory requests does not exceed a threshold. Fig. 1 shows three ways of performing the check. One approach is to snoop the bus and use external circuitry to check the count of outstanding requests (Fig. 1(A)); the observability requirement is significant, *e.g.*, all outstanding transactions. At another extreme, building the checker in silicon (Fig. 1(B)) ameliorates observability but incurs significant hardware cost. However, we can build only the up-down counter in silicon, and the comparator externally (Fig. 1(C)). The composition of the two units is provably equivalent to the original monitor; but the observability requirement is only the *count* of outstanding requests and the only hardware cost is the counter.

The above example is illustrative but pedagogical. We now list several questions that a robust partitioning mechanism must address. We will discuss our approaches to addressing them in the remainder of the paper.

- *How can we use design invariants to aid in partitioning?* Our example partition split the checker. In general, one must take into account design invariants to effectively winnow the set of observed signals.
- *How can we make the partitioning flexible to satisfy different observability requirements?* Our example exhibited

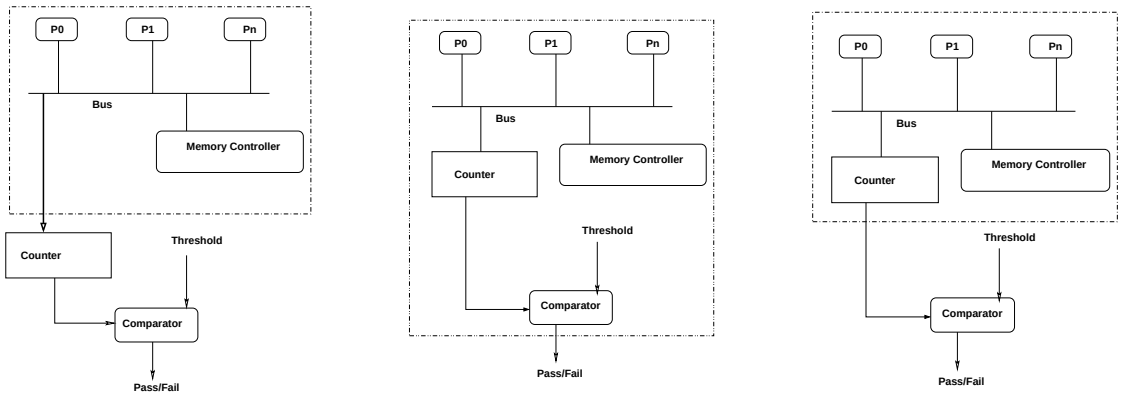

Fig. 1. Example of partitioning a post-silicon check. (A) Checker implemented externally. (B) Checker built entirely as an integrity unit in silicon. (C) Partitioned checking with a portion of checker built into silicon. In each case, the box with dotted outline contains the components built into silicon.

a single partition but in practice, different partitions might be necessary to cater to available observability.

- *How can we automate the decomposition of the monitor?* The above partition was trivial: a part of the checker circuitry was built into silicon. In practice, we need automated decomposition of monitors for complex conditions.

II. Partitioning Framework

Monitor partitioning involves (1) pre-silicon monitor $\mathcal{M}$, (2) post-silicon monitor $\mathcal{P}$, and (3) integrity unit $\mathcal{I}$.

Remark 1: In the description below, we leave implicit the set S of states and set I of inputs but assume that each state and input is represented by a tuple of Booleans. We assume the existence of a state transition function $\mathit{step} : S \times I \rightarrow S$, such that $\mathit{step}(s, i)$ returns the design state one clock cycle from s. Given a formalized HDL, $\mathit{step}(s)$ is derived by running the functional evaluator on the constant representing the design from state s. An execution from state s is a sequence of states obtained by repeated application of step for a corresponding input sequence. We restrict executions to be finite.

Pre-silicon Monitor: For our purpose, a pre-silicon monitor $\mathcal{M}$ is a function that takes a finite execution trace and returns a Boolean. We say that $\mathcal{M}$ *passes* an execution τ if it returns *true*; otherwise $\mathcal{M}$ *fails* τ. We restrict ourselves to monitors that can be defined within a decidable theory.[1] In ACL2, we use a decidable subclass of the logic called SULFA [5], which contains the basic logical constructs (*e.g.*, conjunction, disjunction, conditionals), embeds the theory of lists, and permits (bounded) recursive definitions. The evaluation of a SULFA expression reduces to a bounded check on a finite-state system which can be efficiently synthesized into a netlist. We discuss in Section III how we use this capability.

Remark 2: By the term *monitor* we include both assertion checkers and coverage monitors. In practice, checkers and monitors serve different purposes. A checker must return *true* on each execution; a failure indicates a bug. A coverage monitor determines if a specific corner case has been exercised during testing. However, the distinction is irrelevant to partitioning mechanisms.

Post-silicon Monitor: At post-silicon, only a subsequence of the execution is observable. We refer to such a trace as a *partial trace*. Formally, a partial trace σ is a subsequence of a pre-silicon execution trace τ, with guaranteed observability only of a certain pre-determined sequence of transitions. For a memory system, the guarantee might only include key communication events between the processing elements and memory controller, *e.g.*, signal transitions indicating initiation of a cache request or grants. A post-silicon monitor $\mathcal{P}$ is a function that returns a Boolean on any partial trace σ. Informally, if $\mathcal{P}$ passes a partial trace σ, then we want to guarantee the following: "σ is the subsequence of a pre-silicon trace τ such that a given (pre-silicon) monitor $\mathcal{M}$ passes τ."

Integrity Unit: It is normally impossible to guarantee post-silicon accuracy merely by analyzing a partial trace σ. Suppose σ elides a transaction that enforces a design invariant: then it is impossible to determine from σ whether the invariant is in fact enforced. The use of an integrity unit $\mathcal{I}$ circumvents this problem. $\mathcal{I}$ is a (relatively inexpensive) hardware monitor built into silicon. Given a partial trace σ, the external monitor $\mathcal{P}$ can analyze σ under the assumption that σ is a subsequence of a trace τ that passes the integrity check implemented by $\mathcal{I}$. In practice, the definition of $\mathcal{I}$ is culled from design artifacts used to enforce integrity policies: one common unit deployed in chip-sets monitors the traffic between the processor and the chip-set controller, interrupting the processor if the number of unacknowledged messages exceeds a threshold. The integrity unit does not require additional pins or I/O.

III. Post-silicon Analysis of a Memory System

We illustrate the partitioning approach on the cache coherence protocol of a multiprocessor memory system. The

[1]The decidability restriction is not germane to the framework, but imposed to facilitate the use of SAT as explained in Section III. In practice, the assertions are finite-state hardware invariants which satisfy this restriction.

protocol is loosely based on the German protocol. It uses three communication channels between processes (*clients*) and the memory controller (*home*): channel 1 carries access requests; channel 2 carries grant messages and invalidate requests; channel 3 carries invalidate acknowledgements. We implement channels by a bus-based communication interface; we also implement datapath and memory. The following communication events are relevant, and will be referred to as *critical events*.

- A client p requests a (shared or exclusive) cache block c by setting the request line for c to high in channel 1.
- Home initiates processing a request for block c from process p by setting this request line low.
- When c is granted, home signals its availability to p by setting the grant line for c to high in channel 2.
- When p receives access of c, it sets the grant line low.

The German protocol has been extensively used as a verification benchmark [6], [7], [8]. However, this paper is not about the verification of the protocol; we use it to illustrate some facets of the partitioning mechanism.[2]

Consider checking the following assertion, which is identified at pre-silicon as an invariant necessary for coherence.

Assertion: Home processes cache requests in a sequential order. That is, once Home initiates processing access request of process p for cache block c, it services no further cache access request for c until p has been granted access to c.

It is not difficult to design a pre-silicon monitor $\mathcal{M}$ for this assertion. $\mathcal{M}$ snoops the bus, tracking requests and grants between clients and the home. On the other hand, at post-silicon, observability restrictions obviously preclude recording all bus transactions. In our first simplistic scenario, we restrict our post-silicon monitor $\mathcal{P}$ to record only the critical events for a cache block c. The following theorem connects $\mathcal{P}$ with $\mathcal{M}$. The theorem has been mechanically certified by ACL2.

Theorem 1: Let $\mathcal{P}$ be a monitor that records only the critical events for cache line c. Let τ be any bounded sequence of bus transactions and σ be a subsequence of τ containing the critical events for c. If $\mathcal{P}$ passes σ then $\mathcal{M}$ passes τ.

In spite of being simplistic, $\mathcal{P}$ can uncover useful bugs; indeed, $\mathcal{P}$ was used in an early implementation to uncover the design error shown in Fig. 2. Note that the bug involves several cycles for a specific communication sequence, and is difficult to hit by random simulation (at pre- or post-silicon).

Remark 3: $\mathcal{M}$ and $\mathcal{P}$ are implemented in a decidable theory, *e.g.*, the SULFA subclass of ACL2. A consequence is that Theorem 1 is a formula in a decidable fragment of the ACL2 logic and can be discharged by SAT solving; the automation is critical to the robustness of the framework since the large number of monitors associated with a design preclude the use of theorem proving to discharge individual correctness. Furthermore, we can use SAT-based image computation on SULFA models; from the error isolated by $\mathcal{P}$ on a partial trace, image computation reconstructs a complete trace to exhibit the bug in simulation. Finally, decidability makes it possible to synthesize some functions in the definition of $\mathcal{P}$ into silicon; we return to this point after discussing integrity units.

[2]In previous work, we in fact verified a model of the protocol. That proof is not germane to this work, but it helped identify certain protocol invariants.

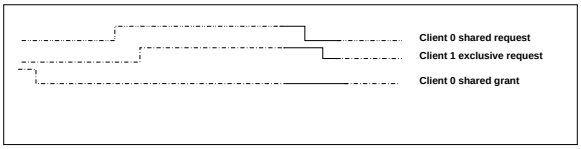

Fig. 2. A post-silicon bug uncovered by $\mathcal{P}$. Home sets request line (for shared access) from Client 0 to low before it sets the request line (for exclusive access) from Client 1 low. Yet, the request line for Client 1 is set to low before the grant line of Client 0 is set to high, violating **Assertion**. Only events represented by fall of solid edges for Clients 0 and 1 are observed. Solid lines constitute a a partial counterexample trace. Dotted lines extend this to a complete execution trace which illustrates the bug in simulation.

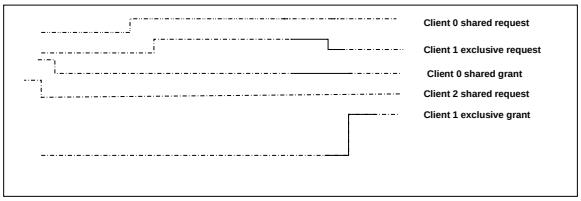

Fig. 3. A certified partially observed post-silicon trace. Only the events corresponding to the rise or fall of a solid edge are observed. Here Home initiates processing the request of Client 1 first, and does not set the access request line for any other client before granting access to Client 1. Dotted lines represent a possible extension of the trace.

Theorem 1 requires σ to contain all critical events. In practice, the number of observable events is restricted to a finite threshold. However, we reconstruct unobserved events through the integrity unit. For our assertion, the following unit $\mathcal{I}_T$ suffices. Let T be the upper bound on the number of observable bus transactions; $\mathcal{I}_T$ counts the number n of critical events among outstanding transactions, signalling an (observable) exception if n exceeds T. The following theorem (certified by ACL2) extends Theorem 1 by accounting for $\mathcal{I}_T$.

Theorem 2: Let $\mathcal{P}_T$ be a monitor that records only the critical events for a cache line c up to a threshold T. Let τ be a bounded sequence of bus transactions, σ be a subsequence of τ containing the critical events for c. If $\mathcal{I}_T$ does not signal exception on τ and $\mathcal{P}_T$ passes σ then $\mathcal{M}$ passes τ.

Theorem 2 is a statement of partial correctness, with the integrity unit providing the logical precondition. Fig. 3 shows a partial trace passed by the combination of $\mathcal{P}_T$ and $\mathcal{I}_T$. Theorem 2 guarantees that the trace indeed satisfies assertion.

The simplicity of Theorem 2 perhaps belies its flexibility. $\mathcal{P}_T$ and $\mathcal{I}_T$ merely represent a *logical* decomposition of $\mathcal{M}$. However, both are compositions of SULFA functions each of which can be mechanically synthesized into hardware description with provably equivalent semantics. Thus, when silicon real-estate is available, we can mechanically produce an augmented hardware design with some functions in the definition of $\mathcal{P}_T$ built into silicon. In the absence of available hardware, $\mathcal{I}_T$ itself can be further decomposed into in-silicon

and external components without affecting the formal guarantee: in the example, $\mathcal{I}_T$ is essentially similar to the checker in Fig. 1; thus, Fig. 1(C) represents one way of its decomposition.

The discussion above underlines the trade-off between hardware cost and logical guarantee. As the in-silicon component is augmented to include a functionality of $\mathcal{P}_T$, observability is ameliorated at the expense of hardware: building the entire $\mathcal{P}_T$ into silicon provides a complete verification accuracy at prohibitive hardware cost. In practice, the in-silicon unit is designed to provide a deterministic communication pattern for the portion of the trace not visible to the external analyzer.

IV. Related Work

One of the earliest uses of formal methods to improve post-silicon observability is by Gopalakrishnan and Chou [1]. They use constraint solving and abstract interpretation to compute state estimates for memory protocols. Ahschlager and Wilkins [9] use model checking techniques for post-silicon debug: the approach is to develop a formal property describing the observed bug and use make use of model-checking to generate a trace that encounters the bug. Safarpour *et al*. [10] use SAT solving to find circuit locations to automatically detect and repair errors using a stuck-at fault model. There has also been work on developing on-chip monitors for enhancing observability [11], [12], [13]; however, there has been little work on decomposing such monitors into on-chip and off-chip components. De Paula *et al*. [14] use SAT-solving techniques to successively "backspace" from a crashed post-silicon state to provide an execution trace that is used for off-line debugging. However, they do not address verification or provide a means for a formal guarantee on post-silicon accuracy.

V. Conclusion and Future Work

We have presented a framework for connecting pre- and post-silicon verification through formally certified partitioning of monitors. Our approach provides a flexible mechanism for transferring pre-silicon verification collateral to post-silicon, and giving a formal assurance on post-silicon accuracy. To our knowledge, our work represents the first connection between pre- and post-silicon analysis through formal proof.

One strength of our framework is the ability to reuse extant design artifacts: pre-silicon checkers and monitors, post-silicon assertions and coverage events, on-chip integrity units, etc., are available either as components of functional verification or as policy enforcement hardware. We use them judiciously to provide an integrated framework for certifying partial traces.

Our work is motivated by the desire to enable a tightly integrated, formal, bidirectional communication between pre-silicon and post-silicon. A deeply embedded HDL with formal semantics permits the interrogation of pre- and post-silicon results and annotations as database queries. Our partitioning mechanism exploits this foundation by using formal proofs as a conduit between pre- and post-silicon verification.

Our framework is in very early stages. In future work, we will tighten the connection between pre- and post-silicon verification and explore opportunities for further automation. For instance, we have only used functional verification artifacts to assist in post-silicon checking, but scalability may require cooperation from the protocol design, *e.g.*, time stamps on messages to determinize communication to assist in reconstruction of unobserved trace fragments.

Acknowledgements: This work has been funded in part by the Semiconductor Research Corporation under Grant No. 08-TJ-1849. We thank Ganesh Gopalakrishnan for insightful discussions.

References

[1] G. Gopalakrishnan and C. Chou, "The Post-silicon Verification Problem: Designing Limited Observability Checkers for Shared Memory Processors," in *5th International Workshop on Designing Correct Circuits (DCC 2004)*, M. Sheeran and T. Melham, Eds., Mar. 2004.

[2] R. Leatherman, "On-chip Instrumentation as a Verification Tool," in *FMCAD 2006 Workshop on Pre- and Post-silicon Verification: Methods and Research Opportunities*, G. Gopalakrishnan, Ed., Nov. 2006.

[3] M. Abramovici, P. Bradley, K. Dwarkanath, P. Levin, G. Memi, and D. Miller, "A Reconfigurable Design-for-Debug Infrastructure for SoCs," in *Proceedings of the 43rd Design Automation Conference (DAC 2006)*. ACM/IEEE, 2006, pp. 7–12.

[4] R. S. Boyer and W. A. Hunt, Jr., "The E Language," in *International Workshop on Hardware Design and Functional Languages (HFL 2007)*, A. K. Martin, C. Seger, and M. Sheeran, Eds., 2007.

[5] E. Reeber and W. A. Hunt, Jr., "A SAT-Based Decision Procedure for the Subclass of Unrollable List Formulas in ACL2 (SULFA)," in *Proceedings of the 3rd International Joint Conference on Computer-Aided Reasoning (IJCAR 2006)*, ser. LNAI, U. Furbach and N. Shankar, Eds., vol. 4130, 2006, pp. 453–467.

[6] E. A. Emerson and V. Kahlon, "Exact and Efficient Verification of Parameterized Cache Coherence Protocols," in *Proceedings of the 12th International Conference on Correct Hardware Design and Verification Methods (CHARME 2003)*, ser. LNCS, D. Geist, Ed., vol. 2860. Springer-Verlag, July 2003, pp. 247–262.

[7] A. Pnueli, S. Ruah, and L. Zuck, "Automatic Deductive Verification with Invisible Invariants," in *Proceedings of the 7th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 2001)*, ser. LNCS, T. Margaria and W. Yi, Eds., vol. 2031. Springer-Verlag, 2001, pp. 82–97.

[8] S. K. Lahiri and R. E. Bryant, "Constructing Quantified Invariants via Predicate Abstraction," in *Proceedings of the 5th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI 2004)*, ser. LNCS, B. Stefen and G. Levi, Eds., vol. 2937. Springer-Verlag, 2004, pp. 267–281.

[9] C. Ahlschlager and D. Wilkins, "Using Magellan to Diagnose Post-Silicon Bugs," *Synopsys Verification Avenue Technical Bulletin*, vol. 4, no. 3, pp. 1–5, Sept. 2004.

[10] S. Safarpour, H. Mangassarian, A. Veneris, M. H. Liffiton, and K. A. Sakallah, "Improved Design Debugging Using Maximum Satisfiability," in *Proceedings of the 7th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2007)*, J. Baumgartner and M. Sheeran, Eds. Austin, TX: IEEE Computer Society, Nov. 2007, pp. 13–19.

[11] M. Boule, J. Chenard, and Z. Zilic, "Adding Debug Enhancements to Assertion Checkers for Hardware Emulation and Silicon Debug," in *International Conference on Computer Design*. IEEE Computer Society, 2006, pp. 294–299.

[12] A. J. Hu, J. Casas, and J. Yang, "Efficient Generation of Monitor Circuits for GSTE Assertion Graphs," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD 2003)*. IEEE/ACM, Nov. 2003, pp. 154–163.

[13] S. Park and S. Mitra, "IFRA: Instruction Footprint and Recording for Post-silicon Bug Localization in Processors," in *Proceedings of the 45th Design Automation Conference (DAC 2008)*. ACM/IEEE, 2008, pp. 373–378.

[14] F. M. De Paula, M. Gort, A. J. Hu, S. Wilton, and J. Yang, "BackSpace: Formal Analysis for Post-Silicon Debug," in *Proceedings of the 8th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2008)*, A. Cimatti and R. B. Jones, Eds. Portland, OR: IEEE Computer Society, Nov. 2008, pp. 1–10.