Exploiting Transaction Level Models for Observability-aware Post-silicon Test Generation

Farimah Farahmandi[1], Prabhat Mishra[1] and Sandip Ray[2]

[1]Computer and Information Science & Engineering
University of Florida, USA
{farimah,prabhat}@cise.ufl.edu

[2] Strategic CAD Labs
Intel Corporation, USA
sandip.ray@intel.com

***Abstract*—A major challenge in post-silicon debug is to generate efficient tests that activate requisite coverage goals on the target hardware while also producing results that are observable through a given on-chip design-for-debug (DfD) architecture. Unfortunately, such tests cannot be generated by analysis of RTL models, both because of design complexity and since the implementation can be buggy. In this paper, we propose an approach to address this problem by exploiting transaction-level models (TLM). Our approach involves mapping test and observability requirements between TLM and RTL, enabling TLM analysis to generate post-silicon tests. We provide case studies to demonstrate the flexibility and effectiveness of the approach.**

I. Introduction

Post-silicon validation of an integrated circuit (IC) entails running tests on a fabricated, pre-production silicon to ensure that the design functions as expected under actual operating conditions and identify errors that have been missed during pre-silicon validation. A fundamental problem in post-silicon validation is the lack of observability and controllability — only a few hundred among the millions of internal signals of an IC can be directly observed or controlled during silicon execution. This makes it difficult to diagnose bugs from observed failures of post-silicon tests, or even identify whether a test has passed, *e.g.*, if the result of a test affects a signal which is not observable, it is difficult to determine whether the test has executed as expected.

To address this problem, it is critical that post-silicon tests be *observability-aware*, *i.e.*, produce results whose values can be reconstructed from the available observability. Unfortunately, this is difficult to achieve for several reasons. First, in an industrial IC development environment, observability architecture and (post-silicon) directed tests are developed independently and concurrently by different teams at different points of the design life-cycle. It is often impossible for test generation teams to account for silicon observability since the observability architecture may not have been fully developed at the time of test generation. Furthermore, it is difficult to employ automated tools for creating (additional) observability-friendly directed tests after the observability architecture has been defined. Creating the observability architecture entails analysis of the RTL models to identify traceable signals; these signals are then routed through appropriate hardware instrumentation to an observation point such as an output pin or memory [1], [2]. On the other hand, analysis of RTL models directly to identify test generation is typically infeasible. RTL models tend to be large and complicated (typically millions of lines of code) making such analysis beyond the capacity of test analysis tools. RTL models may also contain functional or design errors. Indeed, a key reason for post-silicon directed testing is to identify such errors. Consequently, if one develops the directed tests through analysis of the RTL, then the fidelity of the tests as well as any inference made on their effects on observability, may become questionable.

In this paper, we present a technique for observability-aware post-silicon directed test generation through analysis of pre-silicon design collaterals. Our key approach to overcome the scalability and relevance challenges mentioned above is to exploit more abstract transaction-level models (TLM) for the designs to perform our analysis. TLM definitions are much more abstract, structured, and compact, compared to RTL, which permits effective application of exploration to identify high-quality directed tests. A key challenge is to map design functionality and observability between TLM and RTL so that the tests generated at TLM can be translated to effective, observability-aware tests for RTL. We discuss how to develop this mapping in practice. We provide case studies from a number of different design classes to demonstrate the flexibility and generality of our approach.

The remainder of the paper is organized as follows. We discuss related work in Section II. Section III discusses our overall framework, some of the challenges faced, and our approach to overcome them. Section IV discusses our experimental results. Finally, Section V concludes the paper.

II. Related Work

Test generation has been widely studied for functional validation of integrated circuits. Majority of the test generation approaches are designed for pre-silicon validation [3], [4], [9], [6], [8], [5]. Pre-silicon test generation has been performed at different abstraction levels. A vast majority of test generation efforts are focused on validation of RTL implementation [3]. Recent approaches show how to reuse transaction-level tests for RTL validation [9]. With the growing importance of post-silicon validation, there have been significant interest

This work was partially supported by National Science Foundation (NSF) grant CCF-1218629 and an IBM Faculty Award.

in test generation for post-silicon debug. Sousa and Sen [10] generated TLM testbenches using mutation testing. HYBRO [4] generates tests to cover branches using dynamic simulation data as well as static analysis of RTL control flow graphs. Lin et al. [7] presents how to create post-silicon validation tests that quickly detect bugs in multi-core SoCs. Assertion-based validation is widely used in pre-silicon validation to create potential behavioral scenarios in order to increase coverage criteria [11]. However, in post-silicon validation, it is difficult to determine whether a set of assertions has been covered. There has been many efforts to generate tests for activating assertions. Chen. et al. [9] use model checker to generate directed test using TLM models to overcome the complexity of RTL designs. However, none of these approaches consider observability constraints.

III. Observability-aware Test Generation

Suppose we have an RTL model M, a set of checkers and coverage conditions $\mathcal{A}$ to be exercised in post-silicon, and a set of traceable signals $\mathcal{S}$. Our goal is to develop directed tests for exercising $\mathcal{A}$ such that the results of the test can be inferred by observing the signals in $\mathcal{S}$. Our approach uses TLM models for post-silicon test generation. We impose some constraints on the underlying design for viability of our approach. Our key requirement is the existence of a TLM description of the system (in addition to RTL), where the TLM is assumed to be the "golden" specification. Our second requirement is that TLM and RTL models must have the same external (input-output) interfaces. Since SoC designs re composed of a number of hardware or software intellectual property (IP) blocks, we require the IPs to have the same interface variable definitions in TLM and RTL models. Finally, we will only consider assertions from $\mathcal{A}$ that are stutter-insensitive. The general class of stutter-insensitive properties is LTL$\backslash$X properties (X denotes next operator). Tests for stutter-insensitive properties are natural targets for generation based on TLM since the TLM models are untimed.

Fig. 1 provides an overview of our approach. The approach involves four important steps: i) mapping observability constraints as part of test targets, ii) mapping test targets from RTL to TLM, iii) test generation using TLM description, and iv) translating TLM tests to RTL. The basic idea is to transform a RTL assertion (ϕ) as well as observability constraints (ψ) to create a modified RTL assertion with observability constraints (π). The modified assertion needs to be mapped to TLM assertion (α). The TLM assertion/property would be used to construct a TLM test. Finally the TLM test would be translated to an RTL test. In the remainder of this section, we describe each of the steps in detail.

A. Mapping Observability Constraints

Let M_R and M_T be the RTL and TLM models of a design with (common) primary inputs $I = \langle I_1, I_2, ..., I_n \rangle$ and primary outputs $O = \langle O_1, O_2, ..., O_m \rangle$. Let $R = \langle R_1, R_2, ..., R_l \rangle$ be the set of observable RTL signals. Consider a stutter-insensitive assertion ϕ over I, O and R from $\mathcal{A}$. For the purpose of the discussion below, it is convenient to think of ϕ as an LTL$\backslash$ X formula. Our method for generating observability-aware tests for ϕ involves the following steps.

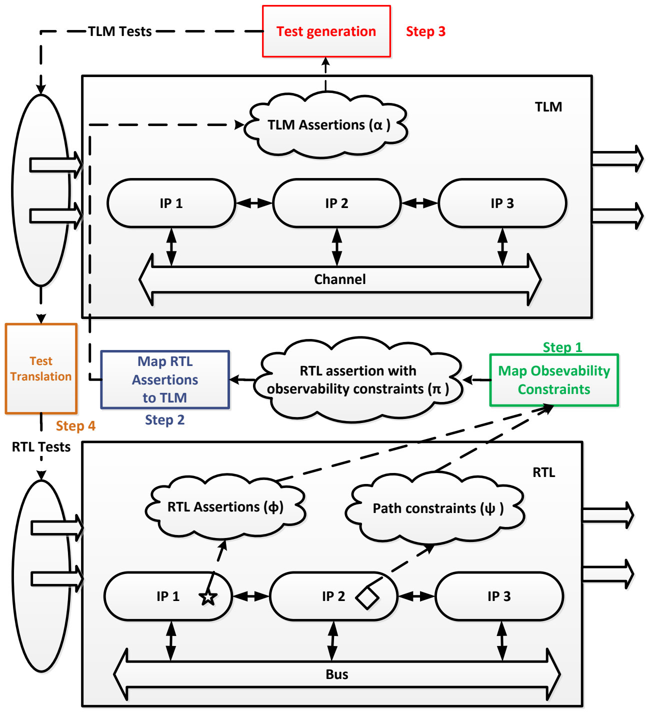

Fig. 1. The proposed methodology with four important steps

1) **Trace Cone-of-influence Calculation:** We traverse the control/data flow of the RTL backwards from the signals in R to the variables in ϕ. This cone-of-influence calculation is made under the constraint that ϕ holds. All signals in R whose cone-of-influence does not include any variable in ϕ are discarded.
2) **Assertion Propagation:** We use symbolic simulation to forward-propagate variables in ϕ along the cone of influence found in Step 1. The result is a restatement of ϕ into a new formula ψ stated in terms of traceable variables (including signals in R and O).
3) **Assertion Abstraction:** We construct a formula π subsuming ϕ and ψ as follows. (i) If ψ is consequent of ϕ, $\pi : (\phi \rightarrow F\psi)$ and vice versa. (ii) If ϕ and ψ can be satisfied concurrently, $\pi : (\phi \wedge \psi)$.

Example 1: Fig. 2 shows a router in RTL and TLM that receives a packet of data from its input channel. The router analyzes the received packet and sends it to one of the three channels based on the packet's address. F_1, F_2 and F_3 receive packets with address of 1, 2 and 3, respectively. Input data consists of three parts: i) parity ($data\_in[0]$ in RTL and $pkt\_in.parity$ in TLM) ii) payload ($data\_in[7..3]$ in RTL and $pkt\_in.payload$ in TLM) and iii) address ($data\_in[2..1]$ in RTL and $pkt\_in.addr$ in TLM). The RTL implementation consists of one FIFO connected to its input port (F_0) and three FIFOs (one for each of the output channels). The FIFOs are controlled by an FSM. The routing module reads the

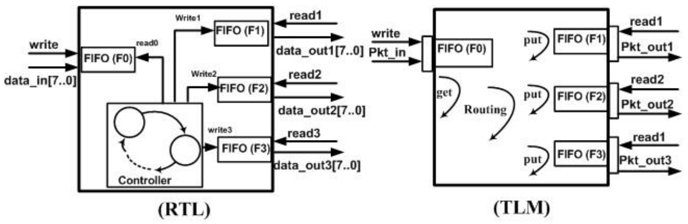

Fig. 2. Router design, RTL and TLM implementations

input packet and asserts the corresponding target FIFO's write signal (write1, write2 and write3). Consider generating a test to check that signal $read0$ from F_0 (which is internal signal) is not stuck at zero. The corresponding assertion, written as an (LTL\ X) formula, is (ϕ : $\mathsf{F}\, read0$). Suppose that the address part of input data of F_1 ($F_1.data\_in[2..1]$) is selected as a trace signal. In order to observe activation's effect of ϕ through $F_1.data\_in[2..1]$, the following predicate must be true two cycles after $read0$ becomes true: $\psi : F_1.write1 \wedge F1.pkt\_in[2..1] = 1$. Thus, following the above steps we get:

$$\pi : Fread0 \rightarrow XX(F_1.write1 \wedge F1.data\_in[2..1] = 1)$$

B. Mapping Test Targets (Assertions) from RTL to TLM

The key challenge in mapping assertions from RTL to TLM is to bridge the abstraction level between the two designs. We achieve this by exploiting the commonality of interfaces. Our goal is to find TLM property α that is *test equivalent* of RTL assertion π constructed in the previous section. Here by *test equivalence* we mean that they generate equivalent tests or counterexamples. The problem reduces to transforming π into a formula α such that (i) α is an LTL\X property over I and O, and (ii) If a test T is a counter-example of α in TLM then T is also a counter-example to π in RTL. If π contains internal RTL variables, we need to turn it to a test equivalent RTL LTL\X property where it is only over the variables in the interface.

We can define α through symbolic simulation of variables in π analogous to the previous section, but this time over the TLM model. Suppose that π is a temporal logic formula over $P_1, P_2, \ldots, P_n$ where each P_i shows one condition on interface or internal variables. For propagation to interface variables, CDFG is traversed backward from point/points that P_i is true to reach primary inputs. [1] As a result, each of P_i be restated as a temporal formula θ over $Q_i = q_1, \ldots, q_m$ where q_j denotes a condition on interface variables. The next step is to remove exact timing notation from θ. Our approach is based on the observation that the original assertion ϕ is stutter-insensitive. Thus distributive property is applied such that their operands are atomic (e.g. $X(q_i \wedge q_j) \equiv X(q_i) \wedge X(q_j)$). In addition, we apply the following rules.

- $F\ (Xp) = F\ p$. Thus, operator F can subsume X.
- A property $p \rightarrow XX...Xq$ can be replaced by $p \rightarrow Fq$.
- A property $p \wedge X\neg p$ can be replaced by $p \wedge F\neg p$.

[1]For some assertions like $P_i \rightarrow P_j$, backward traversal from P_i and forward traversal from P_j would be beneficial. However, in most of the cases performing one of them is enough.

- A property $p_1 \wedge X..Xp_2$ can be replaced by $p_1 \rightarrow F(p_2)$ when p_2 is a condition on variables from set O.

The modified assertion is an LTL\X that contains conditions on interface variables so it can be applied on TLM. In fact, assertion π is mapped as a sequence of *put* and *get* transactions. The next step is to perform name mapping when the interface signal names are not identical. The resultant assertion is our desired assertion α.

Example 2: Consider assertion π from Example 1, we want to turn it to TLM assertion α which is time insensitive an it is formulated over interface variables. From CDFG traversal we know that signal $read0$ (shown in Fig. 2) is asserted when F_0 is not empty. Thus, $X(Xread0 = 1) \equiv (\neg F_0.empty)$. Having non-empty F_0 implies that $write$ signal of F_0 has been asserted before. Thus, we can rewrite the formula as $XX(F0.read0 = 1) \equiv X(\neg F_0.empty) \equiv (F_0.write)$ Using the knowledge $(F1.data\_in[2..1] = 1 \wedge F_1.write1) \equiv (X(F1.read1) \wedge XX(F1.data\_out1[2..1] = 1)$. we get the following formula:

$$\theta : write \rightarrow \mathbf{F}XX(read1 \rightarrow \mathbf{F}Xdata\_out1[2..1] = 1)$$

Finally, assertion α (after name mapping) can be written as:

$$\alpha : Fwrite \rightarrow \mathbf{F}(read1 \rightarrow \mathbf{F}pkt\_out1.addr = 1)$$

C. Test Generation at TLM Level

An assertion α represents a functional property which holds in the design and violation of it exhibits a design fault. Assertion based test generation methods take property $\neg\alpha$ and use model checkers to generate a counter-example for $\neg\alpha$. In other words, checking property $\neg\alpha$ leads to generate a test which can activate the scenario of the property α. Therefore, proper set of assertions results in higher fault coverage and guarantees the success of property based test generation.

In this paper, we make use of SMV as a formal specification to model TLM. SMV model checker [12] is utilized to find the counter-example of property $\neg\alpha$ over SMV model of TLM. The counterexample's assignments to primary inputs is the TLM test case.

D. Translate TLM tests to RTL tests

The final step is to map TLM test vectors to the RTL tests. Since TLM test lacks the timing information in RTL implementation, they cannot be applied to RTL directly. The mapping process consists of two parts. First, the input/output variables are mapped. Next, templates are utilized to map TLM transactions to sequence of RTL computations. The template enables addition of timing relationship. This process is the inverse of our RTL to TLM assertion mapping. The timing relationship in templates can be provided by the designers or can be extracted by design analysis tools [9].

IV. Case Studies

We discuss the application of our approach on two case studies: a NoC switch protocol and a pipelined processor. In these experiments we make use of Bounded SMV Model Checker [12] to optimize test generation time.

TABLE I
TEST GENERATION'S TIME FOR SAFETY PROPERTIES IN NETWORK WITH FOUR SWITCHES

Prop.	Random TG	Our Proposed Method	
		Directed TG	TG with Obs. Constraints
	(min)	(min)	(min)
Property 1	> 600	4.83	6.11
Property 2	> 600	3.45	7.72
Property 3	205	0.49	2.45
Property 4	502.6	1.85	4.73
Property 5	> 600	8.54	13.49

A. *Wormhole Protocol on NOC Switches*

Switches are used as the building block of a Network on Chip (NoC). They receive packets as input and forward it to respective output ports. In this case study, the router uses wormhole routing protocol. We consider test generation for five intersting properties: property 1 is related to reservation of output port of channels, property 2 is about making two internal FIFO's full at the same time, property 3 is about receiving a packet with a specific value, property 4 is related to forcing the acknowledgment signal true and property 5 is related to deadlock detection.

Table I shows the effectiveness of our method in generating observability-aware tests for these five properties. Since there are no prior efforts for observability-aware post-silicon test generation, we have tried to show the usefulness of our approach in two ways: (i) our approach (with observability constraints) takes reasonable test generation time compared to directed test generation (without observability), (ii) random test generation may be infeasible to activate buggy scenarios and propagate their effects to trace signals. We also tried to generate the test directly from RTL when the RTL is buggy; however, test generation failed because the counterexample cannot be produced. This observation emphasizes the importance of test generation in golden TLM. Table I provides statistics of test generation on four other selected properties. The column $DirectedTG$ shows the needed time for generating directed test without considering observability constraints. The time consumption is comparable with the proposed approach but the effect is not observable on trace signals so these test are not useful for post-silicon. Table I also shows that TLM random test generation is drastically worse than our approach and in most of the cases it cannot activate the scenario.

B. *Pipelined Processor*

We have applied our method on a MIPS processor with 5 stages: Fetch (it fetches the new instruction from memory), Decode (it decodes the instruction and reads the possible operands), Execute (it executes the instruction), MEM (it is responsible for load and store operations) and WriteBack (stores the results in instruction's target register). These stages are implemented by one or more IP block in both RTL and golden TLM implementations. These IP blocks are connected by FIFOs together. The result of test generation based on properties related to testing fetch, decode execution units of the processor are reported in Table II. It is obvious that test generation with observability constraints is most beneficial for post-silicon validation.

TABLE II
TEST GENERATION'S TIME FOR SAFETY PROPERTIES IN A PIPELINED PROCESSOR

Prop.	Random TG	Our Proposed Method	
		Directed TG	TG with Obs. Constraints
	(min)	(min)	(min)
Property 1	> 600	0.83	1.90
Property 2	92.10	1.12	1.17
Property 3	297.05	3.00	7.47
Property 4	416.02	1.12	1.36

V. CONCLUSION

We have presented a high-level directed test generation method based on a golden TLM model. The method generates directed test cases at TLM level and maps them back to RTL. The tests not only activate buggy scenarios (especially the scenarios that are hard to activate), but also ensure that the effect of buggy scenario can be transferred observable points in order to help the debugger to root-cause the source of faults. The approach has several merits. First, it enables test generation for buggy RTL designs since our tests are generated using golden TLM model. Second, our method overcomes the scalability limitations of creating automated directed test generation methods at RTL level since TLM models are significantly less complex than RTL implementation. Finally, our test generation takes observability into consideration by forcing results of the buggy scenario activation to the trace signals. Our case studies demonstrate that TLM analysis for observability-aware test generation is feasible for many practical designs.

REFERENCES

[1] K. Basu and P. Mishra, "Restoration-Aware Trace Signal Selection for Post Silicon Validation," *IEEE Trans. on VLSI*, 21(4):605-613, 2013.
[2] K. Rahmani, S. Proch, and P. Mishra, " Efficient Selection of Trace and Scan Signals for Post-Silicon Debug," *IEEE Trans. on VLSI*, 2015.
[3] M. Chen et al., *System-level Validation - High-level Modeling and Directed Test Generation Techniques*, Springer, 2012.
[4] L. Liu and S. Vasudevan, "Efficient Validation Input Generation in RTL by Hybridized Source Code Analysis," *DATE*, 2011.
[5] F. Farahmandi and P. Mishra, "Automated Test Generation for Debugging Arithmetic Circuits," *DATE*, 2016.
[6] X. Qin and P. Mishra, "Directed Test Generation for Validation of Multicore Architectures," *ACM TODAES*, 17(3), June 2012.
[7] D. Lin et al., "Quick Detection of Difficult Bugs for Effective Post-silicon Validation," in *DAC*, 2012.
[8] M. Chen and P. Mishra, "Functional Test Generation using Efficient Property Clustering and Learning Techniques," *IEEE TCAD*, 29(3), 2010.
[9] M. Chen, P. Mishra, and D. Kalita, "Automatic RTL Test Generation from SystemC TLM Specifications," *ACM TECS*, 11(38), July 2012.
[10] M. Sousa and A. Sen, "Generation of TLM Testbenches using Mutation Testing," in *CODES+ISSS*, 2012.
[11] N. Bombieri et al., "RTL Property Abstraction for TLM Assertion-based Verification," in *DATE*, 2015.
[12] *SMV Model Checker*, http://www.kenmcmil.com.