Formal Verification of Partial Good Self-Test Fencing Structures

Adrian E. Seigler, Gary A. Van Huben, and Hari Mony

***Abstract*— The concept of applying partial fencing to logic built-in self test (LBIST) hardware structures for the purpose of using partially good chips is well known in the chip design industry. Deceptively difficult though is the task of verifying that any particular implementation of partial fencing logic actually provides the desired behavior of blocking down-stream impact of all signals from fenced interfaces, and also ensuring that the partial fencing does not inadvertently preclude any common logic from being fully tested.**

In this paper we discuss a case study for a verification method which exploits the power of formal verification to prove that any given partial fencing design satisfies all behavioral expectations. We describe the details of the verification method and discuss the benefits of using this approach versus using traditional simulation methods. We also discuss the testbenches created as part of applying this new method. Furthermore, we discuss the formal verification algorithms that were employed during application of the method along with the tuning that was done to enable efficient completion of the verification tasks at hand.

***Index Terms*— fencing, formal verification, self test**

I. Introduction

Logic Built-In Self Test (LBIST) [1],[2] is an inherent part of today's chip design and fabrication process. With the increasing density of chip die, it's now routine to implement multiple self contained units, cores or even systems within a single physical chip boundary. The most prominent example in the industry is multiple processor cores on a single CPU chip [3]. Such an example is shown in Figure 1. The so-called common logic in the figure comprises elements shared by the cores such as cache and logic for maintaining system coherency. With chips of this nature, running LBIST on the entire chip is adequate to ascertain whether the whole die is functional. However, to enable the use of a partially good chip[4] (such as a case where one or more cores are damaged and all the damage is contained within the boundaries of the damaged cores), the design must implement the concept of partial LBIST [5] fencing.

Partial LBIST fencing allows for self contained areas of a chip to be electrically isolated from the remainder of the chip in cases where such areas are damaged. In this manner, a procedure is employed such that if the LBIST of the entire chip indicates damage, then partial fences can be used to electrically quarantine the affected regions. The partial LBIST signatures will be repeatable assuming that the remainder of the chip is functional and that the partial fencing is properly implemented.

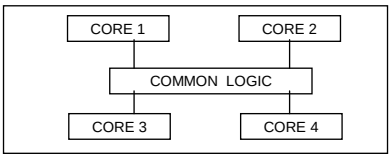

Fig. 1. Multiple core chip with common logic.

Since the risk exists that the interfaces connecting the damaged regions to the remainder of the chip are electrically unpredictable, it is imperative that the partial fencing be implemented correctly. Otherwise, a single missing fencing gate on an interface signal could result in unpredictable signatures. This would in turn result in perfectly usable partial good chips being discarded in the fabrication process.

There are several challenges in developing a robust and easy to use methodology for verifying partial LBIST fencing. The methodology has to scale to be applied on multiple-core chips with hundreds of millions of gates. This places restrictions on the tools that can be used and the algorithms that can be applied. Moreover, the methodology should not depend on having knowledge of the specific LBIST sequences used on the chip. Finally, the methodology should eliminate the need for verification engineers to spend significant time writing drivers and assertions, yet provide full coverage of the design to be tested.

The need for scalability, portability, and full coverage make traditional approaches like simulation [6] unattractive for the purpose of verifying partial LBIST fencing. It has already been shown that proper combination of formal and semi-formal algorithms can enable large-scale pervasive logic verification [7] using formal reasoning. Note however that prior work in this area does not cover verification of partial fencing. With these motivating factors in mind, a new and vastly superior methodology was developed and applied to an IBM z-Series multi-core chip, The description and results of this new approach are discussed here as we present a highly automated, scalable, reusable, and flexible methodology that performs functional formal verification using sequential equivalence checking to verify partial LBIST fencing.

Manuscript received May 8, 2007.
Adrian E. Seigler, Gary A. Van Huben, and Hari Mony - all from IBM Corporation.

II. METHOD CONCEPTS

A. Verification using Sequential Equivalence Checking

We selected the IBM semi-formal and formal verification tool SixthSense [8] as our primary tool for verification. It has already been shown that SixthSense is highly scalable with ability to verify designs with more than 100,000 state elements in the cone of influence. SixthSense provides various ways to abstract the design, including *blackboxing*, which is essential to enable application of our methodology on chip-level models. Moreover, SixthSense provides a wide variety of transformation algorithms for iterative simplification as well as verification algorithms for falsification and proofs.

To ensure portability and a high degree of automation, our methodology uses sequential equivalence checking [9] to perform verification of partial LBIST fencing. The verification objective is to ensure that all interface signals from the damaged portions of the chip are properly fenced. To set up the sequential equivalence checking run, we first develop two models of the design. Model 1 includes the original design with all the fence and interface signals driven to an inactive state. Model 2 includes the original design with all fence signals driven to an active state and all interface signals driven to non-deterministic values. In a typical sequential equivalence checking run, the outputs of the two designs are checked for equivalence. Since we needed to ensure that partial LBIST fencing worked properly, we configured the equivalence checking run to check the Multiple Input Signature Registers (MISR) [2] within the LBIST portion of the design for equivalence. In effect, the MISR is the output of LBIST as it contains the resulting signature from self-test.

There are several advantages in using sequential equivalence checking for partial LBIST verification. First of all, no dedicated manual effort is required to create the model or write assertions. Secondly, there is no need for complex *drivers* representing input assumptions. Third, it ensures portability ; the only project specific items are identification of interface and fence signals and the MISR registers for checking equivalence. Finally, equivalence checking algorithms may be tuned for high scalability [11].

B. False Failures due to Scan Chain Inversions

It is important to understand that if inversions in any of the latches comprising any of the MISR scan chains are allowed, then it is possible for the equivalence check described earlier to produce false failures. Figure 2 illustrates one example of how this could occur.

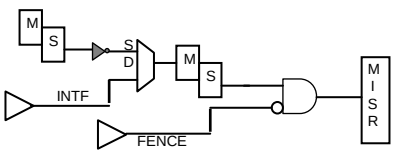

Fig. 2. Example of scan chain inversion using master-slave latches

The shaded inverter in the scan path will have the effect of causing a “0” from the first latch to be received as a “1” on the scan input of the second latch. Since the Model 1 mode of operation is such that all fence and interface signals are driven to an inactive state (i.e. a logical “0” assuming positive active logic), this means that a “1” received by the second latch will subsequently be propagated into the MISR. However, since the same MISR bit is guaranteed to be “0” in Model 2 due to the fact that the fence bit is always active in Model 2, the two models would therefore not be equivalent. The mismatch is not the result of an improperly implemented fence, but is due solely to the scan-chain inversion. Note that this type of mismatch can occur only when the system is in scan mode, thus causing the multiplexer in Figure 2 to select scan signal S instead of data signal D.

In the design used in this case study, the design under test (DUT) contained no scan chain inversions. Nonetheless, the general method described here is equipped to handle scan chain inversions. This can be done for example by using a scan-chain traversal program to identify all inverting latches in the scan chain and then creating a scan input override list that would be used by SixthSense to ensure that the scan inputs to all the inverting latches are always driven to an inactive state. Note that by overriding scan inputs, the potential exists to mask bugs in the implementation of the partial LBIST fencing logic. To avoid such problems, we propose to make use of ternary (0,1,X)-logic [10]. More specifically, prior to executing any scan input overrides, we drive X values to all interface signals and make use of SixthSense in ternary mode to check whether it is possible to bleed X states into the Model 2 MISR. If such bleeding exists, the design bug(s) causing the bleeding must first be fixed before proceeding to override any scan inputs.

C. Over fencing

The intended use of partial fences is to ensure that interface signals from partial good interfaces do not bleed into MISRs. This is illustrated in the upper portion of Figure 3 where the AND gate in the top of the figure serves to block the interface signal from being propagated downstream into sequential logic and the MISR. However, in an improperly implemented design, it is possible for over fencing to exist such that a fence signal also blocks common logic from impacting downstream logic. Figure 3 shows such an over fencing situation where the same fence signal that is used to properly block the partial good interface signal is also used to block common logic (via the AND gate in the bottom of the figure).

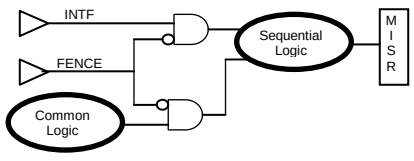

Fig. 3.. Over fencing scenario.

The result of over fencing is that it prevents self-test of common logic, thus allowing for the possibility that any hardware faults that might exist in common logic will go undetected. So, although MISR signatures would be repeatable

in circumstances where there is over fencing, the overall self-test behavior is not the desired behavior. Such a design flaw would not easily be found using traditional validation methods like simulation, but would be quickly found using the equivalence check based verification method used in this case study. Note that if a design contains over fencing bugs, the ternary check for X-state propagation described earlier is not sufficient to detect this.

III. METHOD DESCRIPTION

A. *Method Steps*

The method steps are as follows. Since there were no scan chain inversions in the design used for our case study, Steps 4, 6, and 7 were not required. Nonetheless, all steps for the general method are documented here for completeness.

1. Identify all partial good interfaces for the design whose partial fencing structure is to be verified. Referring to Fig. 1, the chosen interfaces would be the signals connecting the cores to the common logic. For every interface that connects a partial good component with common logic, identify the various control, address and data signals, along with their associated fence signals.
2. Create a wrapper whose inputs and outputs are the previously identified partial good interface signals. The wrapper schematic includes all common logic design components, and excludes all partial good components.
3. Create Model 1 and Model 2 drivers. The Model 1 driver forces all fence and interface signals to an inactive state. The Model 2 driver forces all fence signals to an active state and the interface signals are permitted to assume non-deterministic values.
4. Write an explicit assertion which checks for the propagation of X-states into the Model 2 MISR. Note that this assertion is only meaningful for Model 2 because the partial good interface signals in Model 1 are always inactive and thus are never driven with "X" values.
5. Execute the equivalence check to test that the Model 1 MISR is equivalent to the Model 2 MISR. In parallel, execute checking of the Model 2 X-state assertion. If no failures occur and there are no scan chain inversions, verification is complete. If failures occur with no scan chain inversions, the failures are design problems. If there are scan chain inversions, follow Steps 6 and 7 below to circumvent possible false failures. False failures are indicated by the presence of an equivalence check fail but no fail on the Model 2 X-state assertion.
6. Generate a list of inverted latches in the scan chain and force scan inputs to these latches to the inactive state.
7. Rebuild models and rerun the equivalence check. Any remaining failures are design problems. If no failures occur, verification is complete.

As mentioned already, Step 4 is not required if there are no scan chain inversions in the design. However, it is worthwhile to note here that in addition to the built-in equivalence checking of outputs or selected internal signals, the SixthSense equivalence check mode also provides the ability for users to implement and test customized assertions on either model. Exploiting this capability provides the method with a means to handle potentially false failures that can be produced from the equivalence check when there are scan chain inversions since any failures on the Model 2 X-state assertion are accepted to be design problems and must be rectified.

B. *Advantages of the Method*

There are several benefits to using the above-described method as compared to traditional simulation based approaches. These benefits include:

- **Proof of Correctness** - Our method proves no missing fences and no over fencing. Since it is not practical to exhaustively exercise all combinations of inputs and internal states via simulation in most real world designs, this precludes the possibility of obtaining proofs for the verification properties of interest using simulation.
- **Scalability** - Verification complexity using our method does not increase significantly with complexity or size of the DUT. Unlike simulation, the cost to setup and execute this method is not influenced greatly by the size of the design. This case study shows that application of the method to composite models with more than a million latches is indeed realizable and practical. Attempting to verify the design in this case study using simulation would have consumed considerably more time and would have been inherently incomplete.
- **No knowledge of LBIST sequences required –** Despite the fact that the LBIST sequences used in actual hardware are generally complex sequences that require scanning, our method obviates the need to develop complex drivers as well as the need to manage and update drivers if and when LBIST sequences change. In contrast, simulation often poses the need for one to develop relatively complex testbenches in order to avoid driving invalid input vector combinations and to ensure "interesting" and "corner case" scenarios are tested. By using this formal verification based method, all possible combinations of inputs allowed by the testbench are automatically covered.
- **Setup is very easy -** Once the fence signals and partial good interfaces have been identified (per Step 1 of the method), the verification setup can be auto-generated using scripts.
- **No need for complex assertions: –** Equivalence checking mode is already built into the SixthSense tool, so there is no need to write explicit assertions to verify correctness. The single explicit assertion to check for X-state propagation in Model 2 is trivial to implement.
- **Method supports any partial good self test structure** – The process for creating the composite testbench is the same regardless of design implementation details.

Even though it is possible for designs that contain scan chain inversions to initially produce false failures, such failures are easily identified and subsequently filtered out of the final verification results.

IV. VERIFICATION RESULTS

The proposed verification methodology was applied to an IBM z-Series multi-core chip. Due to the hierarchical nature of the design that was verified, the verification was carried out in two parts – one part at the unit level, and the other part at the chip level. At the unit level, there were two processor cores sharing common logic. At the chip level, there were two units instantiated on the chip with another set of common logic being shared by these two instances. To verify the partial LBIST fencing, one only needs to analyze the common logic shared at the unit and chip levels. Before we built our models, we abstracted all the irrelevant logic using blackboxing. Our application of the methodology was highly successful as we were able to identify a total of 6 design bugs. After these bugs were fixed, we were able to prove that partial LBIST fencing worked properly. Table 1 summarizes the major verification metrics for each of the two models tested.

TABLE 1
VERIFICATION SUMMARY

Verification Metric	Core Level Model	Chip Level Model
# Inputs (thousands)	6.1	41
# Gates (millions)	2	24
# Registers (millions)	2.1	2.8
Run Time	639 sec	1654 sec
Peak Memory Usage	6.8 GB	16.7 GB
# Design Bugs Found	2	4

V. TUNING THE VERIFICATION RUNS

Our two primary challenges when tuning the SixthSense sequential equivalence checking algorithms for verification of partial LBIST fencing were to **1)** find bugs in the design as fast as possible and **2),** efficiently complete proofs of correctness. To address both challenges we used the **EQV** engine, a sequential redundancy removal engine that uses an *assume-then-prove* paradigm to identify and merge gates that are sequentially redundant [11]. The first step is to guess redundant gates using a variety of techniques such as semi-formal analysis, structural analysis, name comparisons, etc. A speculative merge of the redundancy candidates is then performed to create the model to be equivalence checked (the "assume" step). Finally, proof analysis is performed on the speculatively-reduced model to attempt to validate the correctness of redundancy candidates (the "proof" step).

It is well known that SAT-based *bounded model checking* (BMC) is one of the best approaches for falsification. However, we quickly realized that for the sizes of the designs we were interested in verifying, SAT-based BMC quickly runs out of steam. We decided to follow the approach of [11] and applied SAT-based BMC on the speculatively-reduced model for falsification. Speculative-merging enabled deeper BMC and we found all the bugs in the design using this approach. However, LBIST sequences are very long and typically take hundreds of cycles to update the MISRs making it practically impossible to develop a good level of confidence about the design given a few hundred cycles of BMC. This makes it imperative that proofs of correctness be obtained.

The wide variety of synergistic transformation and verification algorithms available in SixthSense were very useful in validating the correctness of redundancy candidates ; Localization in particular was found to be very effective. The SixthSense feature that proved most useful is the ability to identify *causal* redundancy assumptions that make proofs difficult. This is a powerful feature as it enabled the verification engineers to identify certain constraints that the design must satisfy to prove the validity of the causal redundancy assumptions. By adding assertions to check for the satisfaction of the constraints, we were able to figure out illegal driving of certain signals in the design and fine-tune the driver accordingly. Note that the only other way we could have found the illegal driving would be through falsification showing that the MISRs differ. Due to the large fail depth, even BMC on the speculatively-merged model was not successful.

VI. CONCLUSION

The application of our partial fencing verification approach to an IBM z-Series multi-core chip provides solid evidence of ease of use, scalability, and flexibility of the methodology. Six design problems were found and resolved prior to initial chip release. These problems would have almost certainly not been found via simulation, and would have hampered the ability to reliably identify and use partial good chips in real systems. We therefore advocate not only the continued use of this methodology going forward, but also strongly encourage the continued exploration and application of formal verification to other non-trivial verification tasks .

REFERENCES

[1] G. A. Van Huben, "The role of two-cycle simulation in the s/390 verification process," *IBM Journal of Research and Development*, vol. 41, no 4/5, 1997.

[2] W. V. Huott et al., "Advanced microprocessor test strategy and methodology," *IBM Journal of Research and Development*, vol. 41, no 4/5, 1997.

[3] J. A. Kahle et.al., "Introduction to the Cell microprocessor", *IBM Journal of Research and Development*, vol. 49, no 4/5, 2005.

[4] L. Farnsworth et. al., "Partial good integrated circuit and method of testing same", in *U.S. Patent US20050047224A1*, 2005.

[5] M. Riley et. al., "Testability Features of the First-Generation Cell Processor," in ITC, Nov, 2005

[6] B. Wile, J. Goss,,W. Roesner, Comprehensive Functional Verification, San Franciso, CA, Elsevier, pp. 141-197, 439-485.

[7] T. Glokler et al. "Enabling Large-Scale Pervasive Logic Verification through Multi_Algorithmic Formal Reasoning," in FMCAD, Nov. 2006

[8] H. Mony et. al., "Scalable Automated Verification via Expert System Guided Transformations", FMCAD 2004

[9] A. Kuehlmann , C. van Eijk, "Combinational and Sequential Equivalence Checking", in Logic Synthesis and Verification. Kluver Academic Publishers, 2004.

[10] A. Jain et. al., "Testing, verification and diagnosis in the presence of unknowns", in VLSI Test Symposium, 2000.

[11] H. Mony, J. Baumgartner, V. Paruthi, R. Kanzelman, "Exploiting Suspected Redundancy without Proving It", DAC 2005.