Enabling Efficient Post-Silicon Debug by Clustering of Hardware-Assertions

M. H. Neishaburi
McGill University
Montreal, QC, Canada
Mh.neishabouri@mail.mcgill.ca

Zeljko Zilic
McGill University
Montreal, QC, Canada
zeljko.zilic@mcgill.ca

Abstract—**Bug-free first silicon is not guaranteed by the existing pre-silicon verification techniques. To have impeccable products, it is now required to identify any bug as soon as the first silicon becomes available. We consider the Assertion Based Verification techniques for the post-silicon debugging based on the insertion of hardware checkers in the debug infrastructure for complex systems on chip. This paper proposes a method to cluster hardware-assertion checkers using the graph partitioning approach. It turns out that having the clusters of hardware-assertions and controlling each cluster selectively during the debug mode and normal operation of the circuit makes integration of assertions inside the circuits easier, and causes lower energy consumption and efficient debug scheduling.**

I. INTRODUCTION

With the increased complexity of systems, and driven by unquenchable demand for a large set of micro-architectural and other features in a system, design errors and bug become frequent and difficult to track. The existing pre-silicon verification techniques which rely on the formal or functional (dynamic verification) cannot guarantee that the first-silicon works perfectly without any bug. Almost two thirds of newly manufactured SoCs product suffers from bugs at the first silicon [1]. Logic errors are still among the main causes of failures, including the fact that pre-silicon verification is applied to the model of the IC, and not the actual product [6].

The increase in the time-to-market of new products may cause the significant loss of market share, or even complete loss of revenue [1]. Therefore, as soon as the first silicon becomes available, it is required to identify any bug either in the design (corner cases) or the bugs related to manufacturing such as process variation and packaging.

Assertion-Based Verification (ABV) is acknowledged as being the instrumental and efficient pre-silicon verification technique. Armed with temporal logic and extended regular expressions, the PSL (Property Specification Language, IEEE 1850 standard) and the SVA (System Verilog Assertions), perform as languages to describe the expected behavior of a design.

To expand the functionality of assertions beyond design and verification at the pre-silicon debugging, a *checker generator* tool can be exploited to get assertions to operate with the CUD described at the RTL level. Individual assertion, once converted into a circuit form, is also referred to as *Hardware-Assertion* or a checker. Here, we have used the MBAC checker generator to generate hardware-checker or hardware-assertion from either PSL or SVA assertions[7].

Post-silicon validation involves three major activities: detecting problem through specific embedded modules for debugging or reutilizing already available scan architecture for the manufacturing test by applying proper stimulus; localizing and identifying the root cause of the problem; and, correcting or bypassing the problem. Post-silicon bug localization involves identifying the location-time pair for the bug and is the most time-consuming step in the process.

For activating these hardware-assertions and capturing their violation signals, debugger module inside a SoC must be provided by a suitable debug infrastructure [2][11].

As systems become larger and more distributed (and possibly wireless [3]), having clusters of hardware-assertions and controlling each cluster selectively during the debug mode and normal operation of the circuit makes integration of assertions easier, and causes lower energy consumption and efficient debug scheduling. In this work, we investigate a method to cluster hardware-assertions based on the graph partitioning approach by using the input cone graph of hardware-assertions.

II. RELATED WORK

Existing post-silicon debug techniques can be divided into two approaches: trace-based and scan-based. Various implementations for either of them have been realized by previous studies [4][6]. The primary goal in the scan based debug techniques is reusing the internal scan chains which were used during the manufacturing test. At the first step, whenever a specific trigger or hardware checker fires all the internal state elements using these available scan chains will be captured; thereafter these captured data can be offloaded using these scan-out operation of these scan chains [9]. Due to the consecutive stops and resumption during scan dump, this method will not provide the real-time debug information [6].

Trace buffer serves as a space to keep a snapshot of system under debug including signals and states of the system whenever a certain events take place [11]. They have been widely used in legacy debug and logic analysis systems.

The integration of hardware assertions in scan-based run-stop debug infrastructure and in a debug trace infrastructure has

978-3-9810801-6-2/DATE10 © 2010 EDAA

been investigated in [2]. However, this study provides no solution for clustering of related hardware assertions. We have observed that there are still two challenging issues that need to be addressed. First how to cluster Hardware-Assertions in post-silicon chip; second how to use of information from firing signal of each cluster of assertion to spatially isolate the candidate error sites and speed up the debug process.

III. PROPOSED METHOD

Our method tries to cluster the hardware-checkers inside the CUD. To apply our proposed clustering approach to the CUD, at the first step, a directed graph from the circuit gate level net-list must be extracted. The idea of extracting graph from the CUD was used before for post-silicon debug in [10]. Each vertex in this graph represents the Flip-Flop in the design and the directed line shows the combinational circuits or wire which connects these Flip-Flops together. Fig. 1 shows the sample circuit and its corresponding directed graph. The combinational parts of the circuit between storage units were represented by edges and for each flip-flop there exist a vertex inside the directed graph.

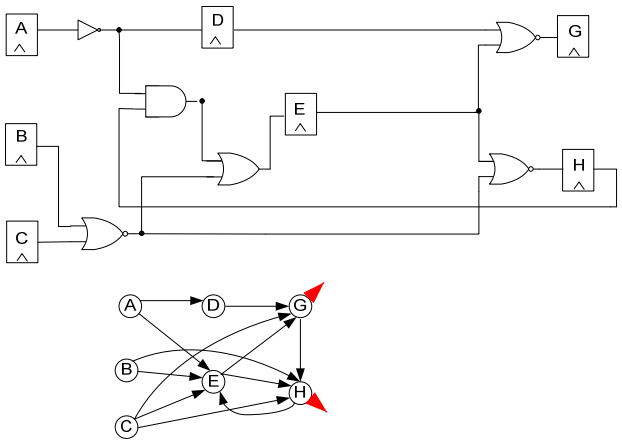

Figure 1. Directed graph Extraction from the given circuit

The second step is extraction of the Fan-in cone graph for each primary output. Fig. 2 shows the extracted fan-in cone graphs for primary outputs of the circuit in Fig. 1.

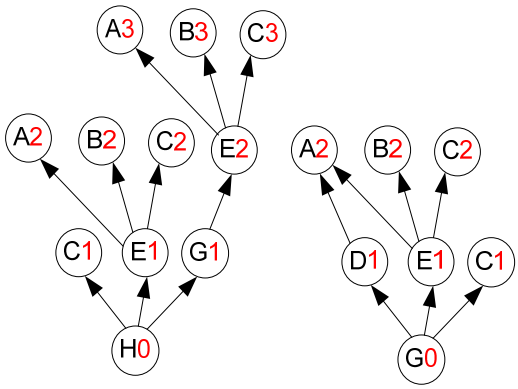

Figure 2. Fan-in cone graph of primary outputs

In this graph, each node represents a storage element inside the CUD. The first primary output is feed by the value of storage elements H0 the node at the first level of this graph. The second primary output is connected to node G0. The directed edge between nodes G1 and H0 in this figure indicates that if there is an error in G1 this error will be propagated after one cycle to the H0. Furthermore, the directed edge between node E2 and G1 implies that an error in E2 will be propagated to G1 and H0 after one and two cycle respectively.

The next step in the process of hardware-assertion clustering is finding out the fan-in cone of each assertion inside the CUD based on the Fan-in cone graph of each primary output. Related nodes to the hardware-assertions input signals should be specified at first in this step.

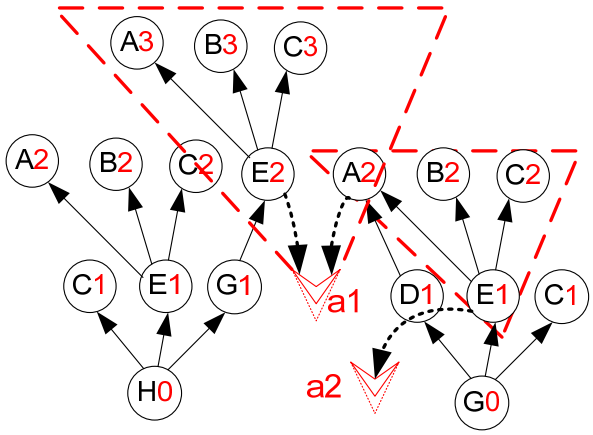

Figure 3. Fan-in cone graph of Hardware-Assertions

Fig. 2 illustrates fan-in cone graphs of each primary output of our simple example of CUD and assertions inside the CUD. Every triangle in this figure represents an assertion inside the CUD. The dotted lines specify the input signals of assertion which are connected to the specific nodes in CM. These nodes are corresponding to the storage elements inside the CUD.

For example in Fig. 3 the input signal of a1 (Assertion1) has been connected to E2 in fan-in cone graph of primary output H and A2 in fan-in cone graph of primary output G; also, a2 (Assertion2) is connected to E1 in the Fan-in cone graph of primary output G.

As soon as hardware-assertions get connected to their related nods, the input cone of all of them inside the CUD can be specified. For instance, in Fig. 3, fan-in cone of A1 is {E2, A3, B3, C3, A2} and fan-in cone of A2 is {E1, A2, B2, C2}.

A. Clustering Hardware Assertions Based on their Fan-in cone graph

In our proposed approach, the placement of Hardware-Assertion and their fan-in cones inside the gate level net-list of CUD is modeled by a new weighted graph called CM (Checker Map) = (V, E).

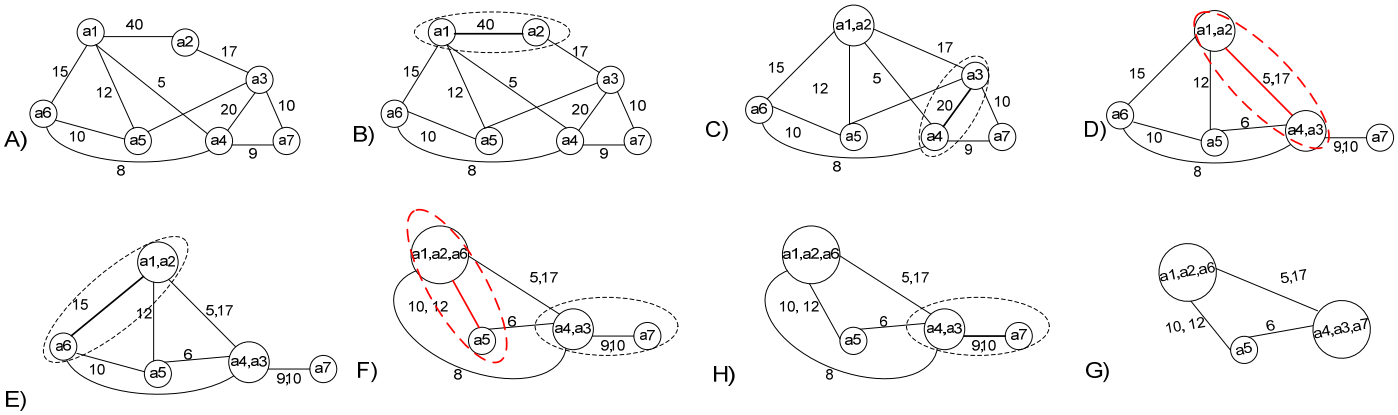

Figure 4. The process of graph clustering on the sample CM(Checker Map) graph

A set of vertices $V = \{v_1, v_2, v_3, \ldots v_n\}$ in CM represents the Hardware-Assertions inside the CUD. The common elements inside the fan-in cone of every coupled hardware-assertion will be indicated by an edge between corresponding two nodes. Furthermore, the weight of that edge represents the number of common elements in the fan-in cone graph of two related hardware-assertions. For instance, the CM graph of our sample circuit at Fig. 1 has two nodes $\{a1, a2\}$ and these two nodes are connected using an edge with the weight of one. To explain the partitioning algorithm on CM graph, we have considered a new CM graph shown in Fig. 4.

This algorithm takes as input the CM graph, the required number of partitions that we want to have, and the maximum number of elements that are allowed to be placed inside each cluster.

In fact, the partitioning algorithm should manage to create the required number of clusters which is given as the first parameter, while it should be also aware that during the iterative process of partitioning the number of elements inside each cluster does not exceed the maximum number of elements that are allowed to be placed inside each cluster.

As it was illustrated in Fig. 4, at each step of the partitioning, the edge with the largest weight is selected. Due to the fact that edge weight represents the common number of storage elements between fan-in cone of two assertions connected by that edge in CM, the larger the weight of the edge the higher the probability that two related assertions fire together and more increases in the chance of extracting required information from firing signal in a cluster of assertion to spatially isolate the candidate error sites.

After finding the edge with the largest weight, two connected nodes by this edge are chosen as a candidate to merge. Thereafter, the partitioning procedure will check whether by merging related nodes the numbers of nodes exceed from the maximum number of allowable nodes.

In Fig. 4 (B) since the weight of the edge between a1 and a2 is larger than that of the others, these two nodes will be merged together after making sure that the number of elements in the new cluster, $\{a1, a2\}$ has less than the maximum number of allowable elements in each cluster.

After merging two nodes, the algorithm should update the CM graph. To update the CM graph any edge that went to $\{a1\}$ and $\{a2\}$ before, now should go to new composite node or a cluster $\{a1, a2\}$.

The iterative partition algorithm performs once again merge operation on the new updated CM graph and the node $\{a3\}$ and $\{a4\}$ is merged together in Fig. 4 (C). In the next iteration, by updating the related CM graph once again the edge with the largest weight is selected during the merge operation in Fig. 4 (D). However, since after merging two concerning clusters $\{a1, a2\}$, $\{a3, a4\}$ the number of elements in the new cluster exceeds the maximum number of allowable elements which is 3, merge Procedure refused to merge these two clusters. Therefore, the next largest edge is selected by the merge operation Fig. 4 (E).

The iterative partitioning algorithm based on the merge and update procedure continues until we create the required number of clusters. The final result of applying the iterative partitioning algorithm in our sample CM graph is shown in Fig. 4 (G). As this figure illustrates, eventually, we have there clusters of assertions and the hardware-assertion inside two of them share the maximum number storage elements. After clustering hardware-assertions inside the CUD into the specified number of clusters and taking into account the limitation on the number of elements in each cluster, we have to equip our post silicon debugging infrastructure by insertion of clusters of hardware-assertions in the debug infrastructure.

IV. Insertion of Hardware-assertions Clusters in the Debug Infrastucture

Having partitioned hardware-assertions based on their input fan-in cone, we have to insert each cluster inside the debugging infrastructure. In fact, the related infrastructure should be equipped to control each cluster of hardware-assertion plus supplied by a mechanism to capture the violated signals of each cluster. To control a specific cluster of hardware-assertions, there must be a way to selectively enable

or disable it. Resorting to TAP interface [5], the technique used to control the regular break-point modules in scan-based run-stop debug infrastructure, each cluster of hardware-assertions has been equipped with a control debug register (EN). Fig. 5 shows how TAP controller can access each cluster of assertions through the chain of debug registers; thereby it provides the required flexibility to enable or disable the particular cluster of Hardware-assertion.

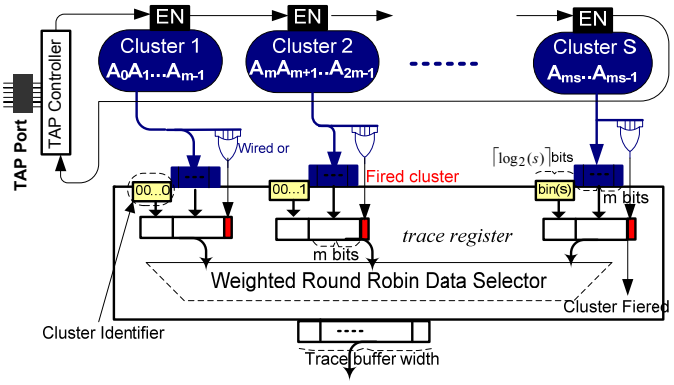

Figure 5. Insertion of the Cluster inside the debug infrastructure

V. EXPERIMENTAL RESULTS

To verify the effectiveness of our method, we have applied our proposed hardware-assertion partitioning algorithm to the AMBA 3 AXI Bus protocol from ARM [12][12]. For the experiments, we have assumed that the bit-width of the debug trace buffer is 16 bits. Since we have 85 assertions in the AMBA 3 AXI Bus protocol, based on the inequality (1) and the number of available clusters in the debugging infrastructure, we can find the set of all arrangements *(c, m, s)* in which we can configure hardware-assertion in CUD. In these arrangements c is number of the trace-register that can be embedded into trace data, m is the number of hardware-assertion inside each cluster and s is the number of the clusters. For example, in our test case, AMBA 3 AXI Bus protocol all these configuration are valid: (2,3,28), (1,4,22), (1,5,17), (1,6,15) , (1,8,11),(1,9,10), (1,14,9),(1,15,7).

We performed the proposed four-step partitioning algorithm on all of these configurations. We first extracted the netlist using Leonardo synthesis tool without considering assertions. Related directed graph for the net-list of the design and the fan-in cone of each primary outputs has been extracted respectively. Then, using MBAC [8] and its rewrite rules [8], we generated Verilog RTL for assertions and added them to the design. Debug enhancements [13], SystemC path [14] and FPGA integration [15] are also available for integration.

Afterward, the CM graph of all hardware-assertions has been defined. Then the partitioning algorithm has been carried out based on the given CM graph and all valid configurations have been determined. Consequently, we integrated all checkers using the presented method in part V.

We used 65nm CMOS library to synthesize our design. During the synthesis our test-case contains the proposed debug infrastructures and all clusters of hardware-assertion. We have synthesis our experiments on all valid configurations. TABLE I shows that the configuration (1,7,13) is the best configuration for resource usage. For energy consumption, configuration (1,8,11) is the best.

TABLE I.

Configuration (c, m, s)	Hardware Cost (MG)	Total Power(W)
(2,3,28)	28.70	0.21
(1,4,22)	28.50	0.21
(1,5,17)	28.42	0.20
(1,6,15)	28.11	0.20
(1,7,13)	28.02	0.18
(1,8,11)	28.20	0.17
(1,9,10)	28.32	0.20
(1,14,9)	28.40	0.21
(1,15,7)	28.70	0.22
No clustering	29.01	0.32

REFERENCES

[1] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller, "A reconfigurable design-for-debug infrastructure for SoC," *Proc. 43rd Design Automation Conference* (43rd DAC), 2006, pp. 7-12

[2] J. Geuzebroek and B. Vermeulen, "Integration of Hardware Assertions in Systems-on-Chip," *Proc. IEEE Intl. Test Conference (ITC)*, 2008, 10 pages.

[3] M-W. Chiang, Z. Zilic, K. Radecka and J-S. Chenard, Architectures of increased availability wireless sensor network nodes, *Proceedings of the International Test Conference*, pp. 1232-1241, 2004.

[4] B. Vermeulen and S. K. Goel, "Design for debug: catching design errors in digital chips," *IEEE Design & Test of Computers*, vol. 19, pp. 35-43, 2002.

[5] ARM limited. Coresight on-chip debug and trace technology. http://www.arm.com/products/solutions/CoreSight.html.

[6] G. J. Van Rootselaar and B. Vermeulen, "Silicon debug: scan chains alone are not enough," *Proc. IEEE Intl. Test Conference ITC*, 1999, pp. 892-902.

[7] M. Boule and Z. Zilic, *Generating Hardware Assertion Checkers: For Hardware Verification, Emulation, Post-Fabrication Debugging and On-Line Monitoring,* Springer, 2005.

[8] M. Boule and Z. Zilic. "Automata-based assertion checker synthesis of PSL properties," *ACM Transactions on Design Automation of Electronic Systems (TODAES),* Vol. 13, No. 1, 20 pages, Jan. 2008.

[9] O. Caty, P. Dahlgren, and I. Bayraktaroglu, "Microprocessor silicon debug based on failure propagation tracing," *Intl. Test Conf.*, 2005, pp. 755-763.

[10] J. Gao, Y. Han, and X. Li, "A new post-silicon debug based on suspect window," in *Proceedings IEEE VLSI Test Symposium.*, 2009, pp. 85-90.

[11] H. F. Ko, Adam B. Kinsman and Nicola Nicolici, "Distributed Embedded Logic Analysis for Post-Silicon Validation of SoC," *Proc. IEEE International Test Conference (ITC)*, 2008, 10 pages.

[12] ARM AMBA 3 specification and assertions. http://www.arm.com/products/solutions/axi_spec.html

[13] M. Boule, J.S. Chenard and Z. Zilic, Adding debug enhancements to assertion checkers for hardware emulation and silicon debug, *International Conference on Computer Design*, pp. 294-299, 2006.

[14] C. Cote and Z. Zilic, Automated SystemC to VHDL translation in hardware/software codesign, *Proceeding of 9th International Conference on Electronics, Circuits and Systems, ICECS*, pp. 717 - 720 , 2002.

[15] Z. Zilic, G. Lemieux, S. Brown and Z. Vranesic, Designing for High Speed-Performance in CPLDs and FPGAs, *The 3rd Canadian Workshop on Field-Programmable Devices (FPD'95)*, pp. 108-113, 1995.