A Methodology for Automated Design of Embedded Bit-flips Detectors in Post-Silicon Validation

Pouya Taatizadeh and Nicola Nicolici
Department of Electrical and Computer Engineering
McMaster University
Hamilton, Ontario L8S 4K1, Canada
Email: taatizp@mcmaster.ca and nicola@ece.mcmaster.ca

***Abstract*—Post-silicon validation is concerned with detecting design errors that escape to silicon prototypes and need to be fixed before committing to high-volume manufacturing. Electrical errors are particularly difficult to catch during the pre-silicon phase because of the insufficient accuracy of device models, which is often traded-off against simulation time. This challenge is further aggravated by the rising number of voltage domains, especially if subtle errors are excited in unique electrical states. Since these electrically-induced subtle errors most commonly manifest in the logic domain as bit-flips, to the best of our knowledge there are no systematic methods to design embedded hardware monitors for generic logic blocks that can detect bit-flips with low detection latency. Toward this goal, we propose a methodology that relies on design assertions that are ranked based on their potential to detect bit-flips and subsequently mapped into user-constrained embedded hardware monitors with the aim to increase bit-flip coverage estimate.**

I. Introduction

Pre-silicon verification offers full controllability and observability, nonetheless it is known to be very slow when compared to the real-time execution; for example, register-transfer level (RTL) simulation is approx 5-6 orders of magnitude slower than the silicon prototypes [1]. Considering the growing size and complexity of modern designs, functional verification might require tens and even hundreds of person-years and the computing power of thousands of workstations [2]. When combining the above constraints with tight project timelines, it is common practice that designs are taped-out when the verification confidence is deemed sufficient. Nonetheless, before committing to high-volume manufacturing, some verification steps continue on the silicon prototypes, a task commonly referred to as post-silicon validation [3].

Unlike manufacturing test, where the primary focus is to detect fabrication defects, the purpose of post-silicon validation is to check the design correctness rather than the manufacturing process. A large number of validation tests, ranging from random input sequences to end-user applications, such as operating systems, computer games or scientific applications, are applied to the silicon prototype and the behaviour is monitored at runtime for unexpected events, such as system crashes or incorrect results [4]. There are two main classes of design errors (or bugs) that escape to silicon: *functional* and *electrical*. Functional errors occur when the implementation deviates from the specification, e.g. the condition for a state transition was incorrectly described in the RTL code. Electrical errors, on the other hand, are caused by subtle interactions between the design and the electrical state of the system, such as power supply noise or thermal effects. Due to the approximations used in circuit-level modelling, as well as the computational requirements needed to account in sufficient detail for the device physics, it is difficult to discover all the electrical errors before tape-out. Therefore, they are the dominant type of design errors that escape to silicon. They emerge at apparently random times under unique and difficult to reproduce conditions; for instance, a sub-block of a circuit might exhibit faulty behaviour only at a certain working temperature due the variations in power supply. The most common manifestation in the logic domain of these subtle electrically-induced errors is in the form of *bit-flips* in flip-flops [5]. The very purpose of post-silicon validation is to detect and localize such errors in order to enable their fix during the subsequent re-spin, rather than to compensate for the erroneous behaviour through fault tolerant mechanisms. In order to achieve this goal, it is critical to detect bit-flips within a low number of clock cycles after their occurrence, since the traces that are acquired on-chip are constrained in depth.

To reduce the *error detection latency*, which is the amount of time it takes for an error, e.g., a bit-flip, to cause an observable failure, system-level methods have been thoroughly investigated [6]. The core idea from [6] is to detect errors rapidly using time-redundant execution, which is diversified and combined with fine-grained checking. The transformation of validation tests is done in a systematic manner, nonetheless the method has been architected for, and hence restricted to, microprocessor-based designs. Another direction of research for reducing detection latency, which is applicable to any circuit block, is to rely on embedded hardware monitors for run-time property checking. Assertions are extensively used to detect and localize design bugs during pre-silicon verification and, using hardware mapping [7], they have shown promise for in-system validation by monitoring the synthesized properties at run-time. In order to expand the number of assertions that can be checked at run-time, time-multiplexing can be employed [8]. Although using assertions in hardware enables the real-time checking for any circuit block, as it is the case during the pre-silicon phase, crafting these hardware assertions based on design functionality, rather than its structure, makes it difficult to assess how many of the potential bit-flips were monitored during a validation session.

In this paper, we propose a **methodology** that identifies assertions that are most suitable for improving bit-flip coverage estimate in post-silicon validation. At the core of the proposed methodology is a new algorithm that ranks a large pool of pre-silicon assertions, based on their potential to detect bit-flips in real-time, before committing them to hardware.

978-3-9815370-4-8/DATE15/©2015 EDAA

II. Methodology

Before elaborating on the main steps in our methodology, we first motivate the usage of assertions for bit-flip detection.

Due to the inherent lack of real-time observability in circuit blocks that are deeply embedded into the design under validation, depending on the workload, most bit-flips will not manifest themselves at an observable output, despite the fact that their presence proves an underlying problem with the design. For example, some of our exploratory simulation experiments have shown that for the s38417 circuit (from the ISCAS89 benchmark set [9]) only one in ten of the injected bit-flips were observable at the primary outputs. Moreover, even if intermittent errors do propagate to outputs, unlike simulation where signal values can be compared against a pre-computed golden response, this is not feasible for post-silicon validation. This is because of the huge volume of clock cycles that are applied to silicon prototypes, which makes pre-computation of golden responses impractical. Another concern that arises when bit-flips are not detected soon after they occur is because failing experiments, which are caused by bit-flips, are not easily reproducible due to the electrical phenomena that cause them (e.g., unique temperature and power supply noise). Hence, on-chip embedded memories that collect a set of trace signals [10–12] will record only a short recent history that lead to the system failure. This recorded information is critical during root-causing [13] and, to ensure that meaningful information is analyzed, the error detection latency must not exceed the depth of the trace buffers. Consequently, considering the goal of low error detection latency, using assertions during post-silicon validation is motivated by the following:

- Assertions can perform property checking without needing a golden response;
- Techniques, such as [7], have been proposed for mapping assertions to hardware, thus making them suitable candidates for post-silicon validation;
- Traditionally, assertions have been carefully crafted by verification engineers before being deployed. Recent researches, such as [14], have explored automatic assertion generation.

Although the original goal of automated assertion generation is to aid pre-silicon verification, e.g., when the design implementation changes iteratively or design blocks are reused in different environments, we recognize the potential of these discovered assertions for post-silicon validation. Therefore we build our methodology (illustrated in Figure 1) by leveraging the recent advancements in both assertion discovery [14] and their hardware mapping [7]. A key observation is that bit-flips, unlike functional errors, are related to the design netlist, which facilitates both a quantification of the error space to be covered, as well as a method that does not rely on design functionality but rather on its structure. This, in turn, facilitates automation in a manner that resembles common tasks in the electronic design flow, such as, for example, logic synthesis, place and route, or automatic test pattern generation. Nonetheless, to make the methodology practically feasible, one has to account for unique hardware constraints. Therefore, the huge number of assertions that are mined during the pre-silicon phase need to be consciously selected before mapping them to hardware, which is an important novel aspect of our work.

A. Automatic Assertion Generation

As illustrated in Figure 1, the first step of our methodology is to find assertions for a given design. Although one can develop assertions manually, in order to enable an automated methodology, it is necessary to rely on tools that can generate non-obvious assertions automatically. In other words, assertions must be extracted from the given design (either netlist or RTL code) irrespective of circuit's functionality. There are two commonly used languages for writing assertions: Property Specification Language (PSL) and System Verilog Assertion (SVA). An example of an SVA assertion is shown below:

```
assr1: ((x == 1) && (y == 0) |=> ##2 (a == 0))
```

where a is the destination signal and x and y can be flip-flops, primary inputs or internal nets.

Most of the available tools for automatic assertion generation are customizable, thus meaning that the user can choose the destination flip-flops. Since the objective is to detect bit-flips that occur in flip-flops, assertions that target flip-flops as destination are deemed to be the ones important to be added to the discovered assertion pool. For instance, in the assertion statement above, if a is a flip-flop and a bit-flip occurs in the circuit changing its value from 0 to 1, and all other conditions hold, then this assertion would be violated. If the status of the assertions is monitored during circuit's operations, then this bit-flip can be detected as soon as the assertion is fired. We call this flip-flop as a *potentially covered* flip-flop if its affected by a bit-flip. We will come back to the topic of coverage in the following subsections.

B. Preparation Experiments

Our exploratory experiments (on the ISCAS89 circuits [9]) indicate that a design block with one thousand flip-flops can easily have more than twenty thousand assertions. Mapping all these assertions to hardware is obviously impractical due to both *area* and *wiring* constraints. For this reason, assertions need to be weighted and a subset of them must be selected as candidate assertions for hardware mapping.

Since our objective is to improve the bit-flip coverage, assertions that are more likely to violate after bit-flip occurrence within a time window are preferred over the other ones. For example, if, during a bit-flip injection experiment, *assr i* detects 12 different bit-flips and *assr j* detects 5 different bit-flips, but 4 of these bit-flips that are detected by *assr j* are also detected by *assr i*, then it would be logical to select *assr i* as the candidate assertion and dismiss *assr j*. There are many other factors, other than the violation count, that need to be taken into account and they will be elaborated in section III.

After discovering assertions, *preparation experiments* are carried out in order to determine the violation count for each assertion when bit-flips are randomly, but uniformly, injected in all the flip-flops. As shown in Figure 2, the following steps are needed to perform the preparation experiments:

1) As it is common for bit-flip injection experiments, flip-flops are instrumented with a 2-to-1 mux and an inverter. The select signal of the mux determines when and where the bit-flip should occur. Also, all discovered assertions are added to the design.
2) For each simulation, we load the circuit in a random state. We wait for a user-defined time (e.g., 10 clock

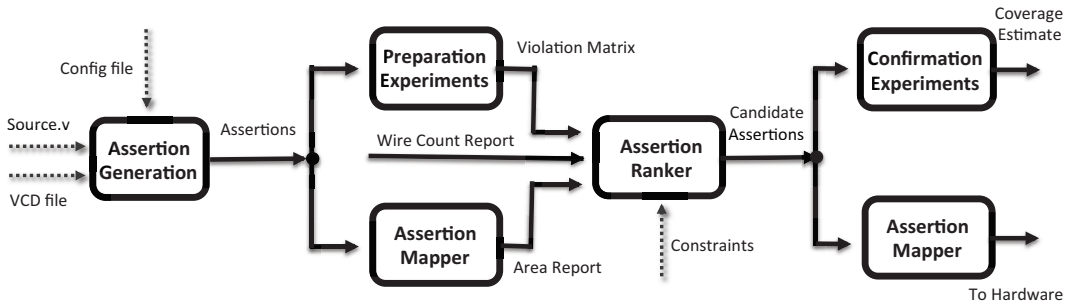


Fig. 1. Tool flow for finding the most suitable assertions to embed on-chip under wire constraints with the aim of maximizing bit-flip coverage.

cycles), during which we monitor assertions to make sure no violation happens due to the unlikely possibility that the initial state is an unreachable state.

3) If there is no violation after that user-defined period, we will target one flip-flop at a time and simulate the design using random input stimuli. In each simulation, bit-flips are injected at n random times and the circuit is simulated for a user-defined time (e.g., 256 clock cycles) after error injection; during this time assertions are monitored and their violation is recorded, as illustrated in Figure 2. As a result, if the circuit has m flip-flops, the total number of simulations will be $m \times n$.

4) Finally, by combining the violation reports for each simulation, a $m \times n$ matrix is created. Each entry in this *Violation Matrix* represents the total violation count of an assertion for a specific flip-flop in all simulations. For instance, entry (1,2) in Figure 2 means that *assr1* has been violated 12 times for all the errors injected in *flip-flop 1*.

It is important to note that the violation matrix captures the error space of bit-flips that can be detected by the assertions that were assessed during the preparation experiments. Due to the randomness in the preparation experiments, which is needed to account for the random occurrence of bit-flips on silicon prototypes, an assertion that is fired for one bit-flip injection might not be violated for another bit-flip injection

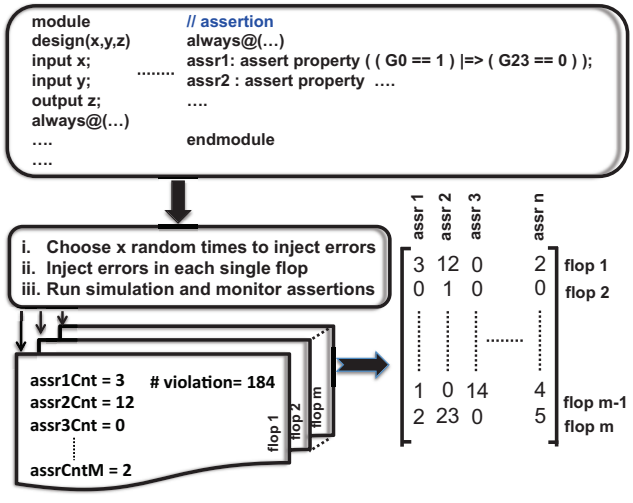


Fig. 2. Preparation Experiments illustrating steps towards creation of the Violation Matrix. Each entry in this violation matrix shows the total number of violations of the assertion (from the corresponding column) when the bit-flip was injected in the respective flip-flop (identified by the row).

in the same flip-flop; this is due to the circuit being in a different state and also the effect of the bit-flip might not propagate to the assertion checker due to a different input sequence. This is the reason for referring to the flip-flops from the violation matrix, for which at least one assertion has fired during the preparation experiments, as potentially covered. It is up to the user of the methodology to exclude or include flip-flops of interest for the bit-flip injection experiments. Hence, the number of rows in the violation matrix is upper-bounded by the number of flip-flops in the design and the number of columns, i.e., the number of assertions to be considered by the preparation experiments, can be bounded based on, for example, the available computational resources. It should also be noted that the quality of the violation matrix is central to the accuracy of the assertion ranking algorithm. The more simulations we run during the preparation experiments, the more relevant is the information in the violation matrix.

C. Mapping Assertions to Hardware

Assertions have been developed for verification and are composed of logic and temporal operators and regular expressions. These statements can be added to the source code in pre-silicon verification to monitor errors using functional simulators. However, for using them in post-silicon validation, they must be mapped into hardware in order to do on-line property checking. Both PSL and SVA assertions are not synthesizable by default. However, as mentioned before, there are tools such as [7] that can accomplish assertion synthesis. Once assertions are discovered, assertion mapping can be done simultaneously with the preparation experiments. This will provide accurate area estimates for each assertion, which are needed by the ranking algorithm, as it can be seen in Figure 1.

D. Assertion Ranking

Due to area and wiring constraints, adding all assertions to hardware is not feasible. Therefore, the large pool of available assertions must be assessed and only a subset of them are chosen and marked as candidate assertions to be embedded into hardware. In this work, by having established a relationship between assertions and bit-flips that they might detect, we focus on maximizing the number of flip-flops that are potentially covered, as defined above, under user-provided constraints. The ranking algorithm, which will be detailed in section III, uses the violation matrix, area estimates, the wire count report and user-specified constraints. Steps needed to produce the violation matrix were explained in section II-B; likewise, area estimates for assertions are obtained as explained in section II-C. The wire count report can be directly extracted from the assertions pool by counting the distinct number of wires that comprise each assertion statement. Finally, the

constraints are provided by the user and, in our current implementation, we provide the wire count as the constraint.

E. Confirmation Experiments

The last step in our methodology is to run confirmation experiments, which are carried out using only the subset of assertions selected by the ranking algorithm. The circuit is simulated using random stimuli during which one error is injected at a time (injections will be uniformly distributed across all the flip-flops throughout the entire duration of the confirmation experiments) and the assertion violations are recorded. In the ideal case, whenever a bit-flip is injected into an arbitrary flip-flop, a violation will be detected if at least one of the selected assertions was identified during the preparation experiments to potentially cover the respective flip-flop. Considering the random occurrence of bit-flips, it can happen that some bit-flip injections will not cause a violation despite the fact that more than one of the selected assertions was expected to fire. Further, though it is less likely, it might also happen that some of the selected assertions, that were not identified during the preparation experiments to be related to a particular flip-flop, will fire during the confirmation experiments when a bit-flip is injected in the respective flip-flop.

The ratio between the number of bit-flips detected by the confirmation experiments and the total number of bit-flips injected during the confirmation experiments is defined as the bit-flip *coverage estimate* and it can provide important feedback to different steps of the proposed methodology. For example, the original pool of assertions might need to be expanded if the confirmation experiments indicate that, for some flip-flops, no bit-flips were detected; it is also possible that an insufficient number of preparation experiments have been carried out, and the confirmation experiments will indicate that the quality of the violation matrix needs to be improved. In addition, this *coverage estimate* can also serve as a proof of due diligence when the subset of the selected assertions was used on the silicon prototype and, after extensively long post-silicon validation experiments, no failures were recorded; in this case, the confidence to commit to high-volume of manufacturing is substantiated by the value captured by the coverage estimate.

III. Ranking Algorithm

As emphasized in the previous section, mapping all the assertions from the violation matrix to hardware for the purpose of low-latency bit-flip detection is impractical. Therefore, as summarized in section II-D, we present a novel algorithm to select a subset of assertions by accounting for a wire count constraint. The objective is to maximize the potential coverage of bit-flips for each flip-flop, while at the same time the area should be contained. The flow of the ranking algorithm is illustrated in Figure 3. Specific details of each step are given next:

- In order to prioritize one assertion over another, a metric is defined for the purpose of one-to-one comparison. We define the *importance metric* (IM) as:

$$Assr(i)_{IM} = \frac{BFCov \times TotalViolation}{(\alpha \times WireCnt) + (\beta \times Area)} \times \frac{1}{\sigma}$$

  where `BFCov` is the total number of bit-flips that are covered by *assr(i)*, `TotalViolation` is the total number of times that *assr(i)* has been violated in all simulations during preparation experiments, `WireCnt` is the number of wires used in *assr(i)*, `Area` is the area estimate obtained through mapping assertions to hardware, and α and β are empirically determined parameters used to bring the wire and area units on the same scale. The standard deviation σ of the violation count for each assertion is important because of the following: if two assertions have a similar BFCov, area, wire count and TotalViolation, the selected assertion should not have a significant discrepancy in violation counts for different flip-flops, since it will likely have a higher potential to detect bit-flips for all the flip-flops that it is related to.

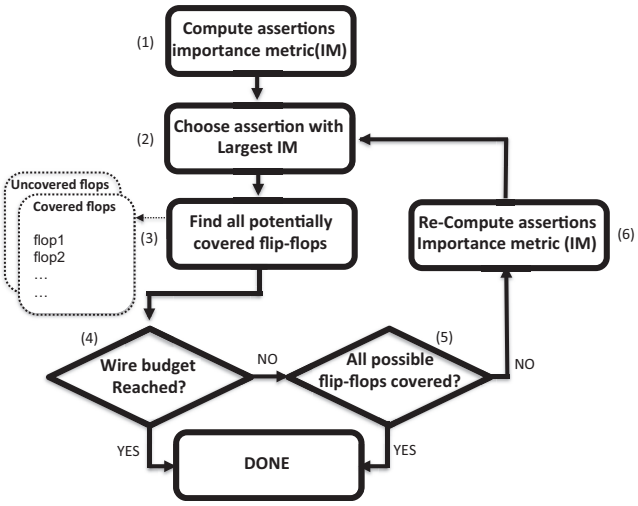


Fig. 3. Flow for the ranking algorithm.

- Once the *importance metric* is computed for all the assertions, the next step would be to select the one with highest IM. In Figure 4, assuming that all assertions have equal wire count of 4, area size of 6 and unit size coefficients, the following IMs will be computed:

$$assr(1)_{IM} = \frac{3 \times 19}{(1 \times 4) + (1 \times 6)} \times \frac{1}{7.26} = 0.392$$
$$assr(2)_{IM} = \frac{3 \times 19}{(1 \times 4) + (1 \times 6)} \times \frac{1}{13.7} = 0.208$$
$$assr(3)_{IM} = \frac{3 \times 8}{(1 \times 4) + (1 \times 6)} \times \frac{1}{3.63} = 0.33$$
$$assr(4)_{IM} = \frac{3 \times 13}{(1 \times 4) + (1 \times 6)} \times \frac{1}{8.43} = 0.231$$

  Based on the above, *assr1* has the largest IM so it is selected first (Steps 1 and 2 in Figure 3). It is important to note that both *assr1* and *assr2* have similar BFCov and TotalViolation. However, TotalViolation for *assr2* is dominated by the high violation count for *flip-flop 1*, which leads to its standard deviation to be larger than the one for *assr1*; therefore, it is not chosen to cover *flip-flops 1-3*. For the same total violation count, it is of interest to choose an assertion that is equally likely to cover all the flip-flops it is related to, rather than choose an assertion whose total violation

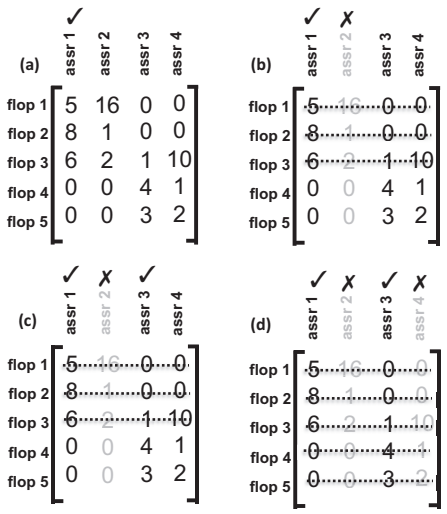


Fig. 4. Example of ranking assertions based on the violation matrix. Note that after a particular assertion is selected, the flip-flops that are covered by it are added to the covered list and are not taken into further consideration.

count is dominated by the violation count for one flip-flop only. Once an assertion is selected, all the flip-flops that are covered by that assertion are marked (dotted line in Figure 4 and Step 3 in Figure 3), so that other assertions are chosen based on the covering requirements for the rest of the flip-flops.

- After each selection, it is mandatory to check if we have reached the available wire budget. Likewise, there is no point to select more assertions if the set of assertions selected so far covers all the flip-flops in the violation matrix (Steps 4 and 5 in Figure 3).
- If none of the aforementioned conditions hold, then the next step is to re-compute the *Importance Metric* for all the unselected assertions. This is a critical step, since BFCov and TotalViolation must take into account only the values for bit-flips in flip-flops that have not been covered so far. For example, *assr4* has an original TotalViolation larger than *assr3*, however it is not chosen because, once only uncovered flip-flops are taken into account, its TotalViolation will be less than that of *assr3* (Step 6 in Figure 3).

IV. Experimental Results

The software for all the steps for our methodology, including the proposed ranking algorithm, have been implemented on an Intel Core i7 machine with 32GB of RAM using GCC 4.8.2. For assertion discovery, we have used GoldMine, an automatic assertion generation tool that uses data mining and formal verification. GoldMine has several modes of operation and the detailed discussion on its different modes and configurations are out of the scope of this paper; the interested reader is referred to [14]. Results from a combination of mining engines of GoldMine have been used for gathering assertions in our test cases; we should note that decision forest and coverage mode engines have been used more often than the other engines. With regard to the input stimuli, if Value Change Dump (VCD) file is provided to GoldMine, it will be used as input for assertion discovery; otherwise, it will automatically generate random stimuli to be used for assertion discovery. We have used both random stimuli and deterministically generated VCD files obtained using the Validation Vector Generator tool from Virginia Tech [15]. MBAC [7] has been used for mapping assertions to hardware. All assertions have been added to the original source code and passed to MBAC. Once the synthesizable Verilog model for all the assertions is produced by [7], the area estimate for each is determined using Synopsys Design Compiler (based on generic implementation libraries).

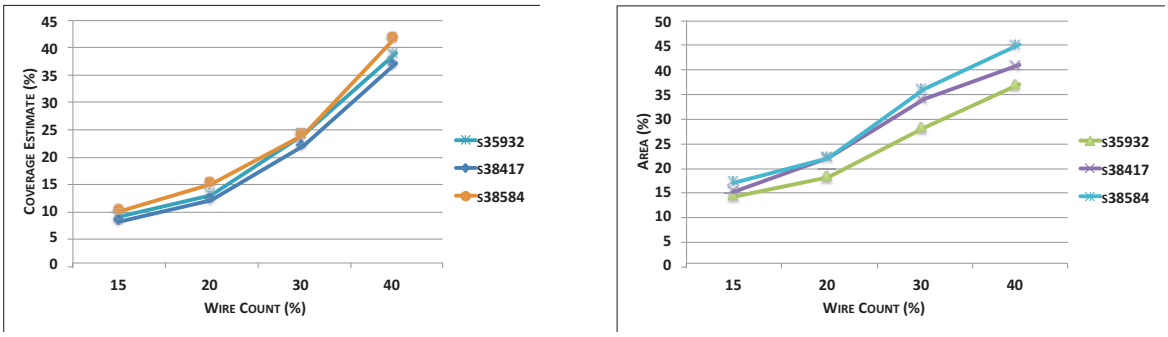
After creating the *violation matrix* (the output of preparation experiments explained in section II-B) and extracting area estimates and wire counts of assertions, the ranking algorithm will select a subset of assertions. Although a user can choose constraints specific to his hardware environment, in our current implementation we have used the number of wires as the constraint. The objective is to maximize bit-flip coverage estimate within the wire count budget, as discussed in section III. Once the assertions are selected, confirmation experiments are carried out to obtain the *coverage estimate*, as introduced in section II-E. In this work, in order to reflect the random occurrence of bit-flips in post-silicon environments, all the preparation and confirmation experiments are performed using random stimuli. The total run-time for our current implementation is mostly dominated by the assertion discovery part. For the benchmark circuits studied in this paper, the task of finding assertions varies in the range of three to ten days. Once this step is complete, the run-time for subsequent steps depends on the constraints and the number of simulations, though their contributions to the total run-time are insignificant compared to assertion generation. It should be noted that the preparation and confirmation experiments can be migrated to emulation-based environments that will further improve both the run-time of experiments and the accuracy of the violation matrix, which indirectly will impact the quality of the results.

Figure 5a illustrates how the coverage estimate changes as the number of wires is varied for the three largest ISCAS89 benchmark circuits [9]. Since there exists an overlap between the nets that are used by different assertions, and also the fact that some assertions which have been used for this work cover multiple bit-flips, we can notice that, in general, as the wire count is increased, the slope of change in coverage gets steeper. In addition, Figure 5b shows how much area is needed by the hardware assertion checkers with respect to the total area of the circuit when the wire count is varied from 15% to 40% of the *total number of flip-flops* (and not the total number of nets) in the design.

As motivated in section II, the key reason to use assertion checking for bit-flip detection is to minimize the error detection latency. Table I shows the number of errors that fall into different error detection latency windows. As it can be seen in this table, the vast majority of the bit-flips are detected in less than 10 clock cycles after their occurrence. An interesting

TABLE I. Evaluation of the error detection latency for the detected errors

Circuit	<5 clock cycles	<10 clock cycles	>10 clock cycles
s35932	98.1%	99.4%	0.6%
s38417	96.3%	99.1%	0.9%
s38584	98.4%	99.4%	0.5%



(a) Coverage estimate when varying the wire count

(b) Area evaluation when varying the wire count

Fig. 5. Analysis of coverage estimate and area based on varying the number of wires, running the assertion ranker and carrying out the confirmation experiments.

topic to explore in future work is how one can trade-off the error detection latency, which is acceptable to be in the range of tens to hundreds of clock cycles in practice, for a lower area investment for the bit-flip assertion checkers.

V. Concluding Remarks

In this paper we have presented a methodology for automatically assessing the quality of assertions based on their potential toward maximizing the bit-flip coverage estimate in post-silicon validation. As shown by our experimental results, a coverage of approx 40% is attainable when using a number of wires equal to 40% of the number of flip-flops in the design. In other words, by injecting bit-flips, we have estimated that at least 40% of them will be detected by a subset of selected assertions that are mapped onto hardware. It should also be noted that this coverage estimate has been achieved by embedding, on average, only 4% of the total number of assertions discovered during pre-silicon verification.

It is noteworthy to mention that the reported coverage estimates are based on random bit-flip injections that are distributed uniformly across *all* the flip-flops in the assessed designs. However, it has been investigated in [16] that electrically-induced errors are more likely to affect the values of timing-critical flip-flops. Therefore, if the experiments would be done only for a subset of timing-critical flip-flops, the coverage estimates for these flip-flops are expected to be significantly higher when the same wire constraint would be used (and also a similar amount of on-chip hardware would be allocated). Running this type customized experiments is fully supported by the methodology presented in this paper.

Finally, since post-silicon validation experiments run for extensively long times, it is possible to have multiple debug sessions during which assertions for a subset of flip-flops are configured into an embedded programmable event-detector, as proposed in [8]. In this paper we did not assume that event-detectors will be programmable and hence all of the assertions have dedicated hardware circuitry and monitor concurrently. Nonetheless, if embedded programmable logic cores are used for implementing the assertion checkers, then, through time-multiplexing, the coverage estimate can be further improved using the same area invested for event detection.

Acknowledgment

The authors thank McGill University for MBAC [7] (assertion synthesis), UIUC for GoldMine [14] (automatic assertion generation) and Virginia Tech for the Validation Vector Generator [15] (used for deterministic stimuli for Goldmine).

References

[1] J. Goodenough and R. Aitken, "Post-silicon is too late avoiding the $50 million paperweight starts with validated designs," in *ACM/IEEE Design Automation Conference (DAC)*, June 2010, pp. 8–11.

[2] A. Adir, A. Nahir, G. Shurek, A. Ziv, C. Meissner, and J. Schumann, "Leveraging pre-silicon verification resources for the post-silicon validation of the IBM POWER7 processor," in *ACM/IEEE Design Automation Conference (DAC)*, June 2011, pp. 569–574.

[3] A. Nahir, A. Ziv, M. Abramovici, A. Camilleri, R. Galivanche, B. Bentley, H. Foster, A. Hu, V. Bertacco, and S. Kapoor, "Bridging pre-silicon verification and post-silicon validation," in *ACM/IEEE Design Automation Conference (DAC)*, June 2010, pp. 94–95.

[4] Intel Corp, "Intel platform and component validation," 2003, http://download.intel.com/design/chipsets/labtour/PVPT_WhitePaper.pdf.

[5] S. Mitra, S. Seshia, and N. Nicolici, "Post-silicon validation opportunities, challenges and recent advances," in *ACM/IEEE Design Automation Conference (DAC)*, June 2010, pp. 12–17.

[6] T. Hong, Y. Li, S.-B. Park, D. Mui, D. Lin, Z. Kaleq, N. Hakim, H. Naeimi, D. Gardner, and S. Mitra, "QED: Quick error detection tests for effective post-silicon validation," in *IEEE International Test Conference (ITC)*, Nov 2010, pp. 1–10.

[7] M. Boule, J.-S. Chenard, and Z. Zilic, "Assertion checkers in verification, silicon debug and in-field diagnosis," in *IEEE International Symposium on Quality Electronic Design*, March 2007, pp. 613–620.

[8] M. Gao and K.-T. Cheng, "A case study of time-multiplexed assertion checking for post-silicon debugging," in *IEEE International High Level Design Validation and Test Workshop (HLDVT)*, June 2010, pp. 90–96.

[9] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 1989, pp. 1929–1934 vol.3.

[10] D. Chatterjee, C. McCarter, and V. Bertacco, "Simulation-based signal selection for state restoration in silicon debug," in *ACM/IEEE Int. Conf. on Computer-Aided Design (ICCAD)*, Nov 2011, pp. 595–601.

[11] K. Rahmani, P. Mishra, and S. Ray, "Efficient trace signal selection using augmentation and ILP techniques," in *IEEE International Symposium on Quality Electronic Design (ISQED)*, March 2014, pp. 148–155.

[12] H. Ko and N. Nicolici, "Automated trace signals identification and state restoration for improving observability in post-silicon validation," in *ACM/IEEE Design, Automation and Test in Europe (DATE)*, March 2008, pp. 1298–1303.

[13] Y.-S. Yang, N. Nicolici, and A. Veneris, "Automated data analysis solutions to silicon debug," in *ACM/IEEE Design, Automation Test in Europe Conference Exhibition (DATE)*, April 2009, pp. 982–987.

[14] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy, and D. Johnson, "Goldmine: Automatic assertion generation using data mining and static analysis," in *ACM/IEEE Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2010, pp. 626–629.

[15] A. Parikh, W. Wu, and M. Hsiao, "Mining-guided state justification with partitioned navigation tracks," in *IEEE International Test Conference (ITC)*, Oct 2007, pp. 1–10.

[16] M. Gao, P. Lisherness, and K.-T. Cheng, "Post-silicon bug detection for variation induced electrical bugs," in *ACM/IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2011, pp. 273–278.