Tackling Random Blind Spots with Strategy-Driven Stimulus Generation

Matthew Ballance
Mentor Graphics Corporation
Design Verification Technology Division
Wilsonville, Oregon
matt_ballance@mentor.com

***Abstract*— Hardware verification typically uses two primary types of stimulus generation: engineer-directed stimulus generation and open-loop random generation. This paper proposes a third approach for generating stimulus that automatically identifies and produces high-value test patterns from a constraint-based stimulus model. A trial implementation built on top of an intelligent testbench automation tool is described.**

Keywords—functional verification; automated test generation; intelligent testbench automation; graph-based stimulus

I. INTRODUCTION

In today's hardware-verification space, two categories of tests are being created: engineer-directed tests and open-loop random tests. Directed and coverage-driven testing are examples of engineer-directed activities. Based on a test plan, the engineer lays out a goal of cases to test and either proceeds to create a test to exercise those cases (directed test) or defines functional coverage to ensure that random stimulus hits the identified test goal. Engineer-directed testing has many benefits, including ensuring that key functionality is verified in a documented and repeatable manner. The primary drawback is the "imagination gap": the very human inability to imagine key but obscure combinations of functionality that must be verified. In contrast, the main benefit of open-loop random simulation is that it closes this gap. Taking the engineer out the imagination loop enables automation to create legal but obscure cases and helps to find bugs.

Both engineer-directed and open-loop random testing are valuable techniques. Consequently, a typical verification cycle starts with engineer-directed tests to verify basic functionality, transitions to a 'random simulations' stage to help find bugs, and concludes with closing coverage on the engineer-defined functional coverage goals. There are, of course, challenges in each phase of this cycle. This paper focuses on the random-regressions phase of the verification cycle. It proposes a technique to augment the generation of pure-random stimulus with high-value stimulus identified using common patterns and information extracted from a constraint model.

II. CHALLENGES OF PURE-RANDOM GENERATION

In order to understand why it would be desirable to augment pure-random generation during the random regressions phase of the verification cycle, it is necessary to explore some of the downsides of pure-random generation.

In our example three-phase verification cycle, both the bring-up phase and the coverage-closure phase provide good metrics on what is being tested relative to defined goals. By contrast, in the random regressions phase, the only meaningful metric of verification progress is bugs found. When a bug is found, the stimulus generated during that simulation is identified by the random seed. This is a good thing in the sense that it allows the stimulus set to be reproduced such that the bug can be examined and corrected. The drawback is that changes to the design and testbench environment – perhaps to resolve the just-discovered defect – change the meaning of the seed and make it impossible to reproduce the same stimulus pattern that uncovered the issue. In addition, when a defect is discovered, ideally it would be desirable to generate new, similar stimulus to the one that uncovered the defect. The random seed provides no information to enable this to be done.

Our verification goal during the random regressions phase of the verification cycle is to get to as many corners of the stimulus state space as possible. Random-resistant corner cases are an unfortunate artifact of even the best constraint solver and present an obstacle to achieving this verification goal. Random-resistant cases result from the fact that random stimulus generation is all about probability. A well-implemented random constraint solver will generate an even distribution of values across the domain of random variables in the absence of constraints.

Take, for example, the simple SystemVerilog class with two random variables shown in Figure 1.

```
class unconstrained;

  rand bit[3:0]          A;
  rand bit[3:0]          B;

endclass
```

Figure 1 - Unconstrained random-stimulus class

Both variables have a reachable domain of 0 through 15. Running 1,000 randomizations and counting the number of times each variable takes on each value might result in a plot like the one in Figure 2.

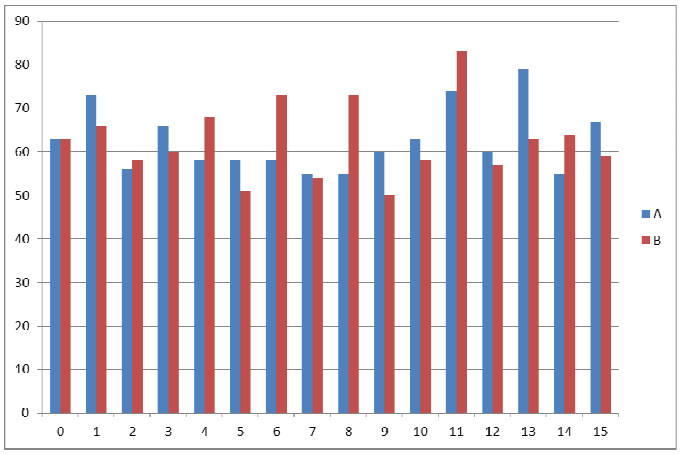

Figure 2 - Unconstrained-random value distribution

Note that the distribution of values is very even across the reachable domain and between the A and B variables. From a verification perspective, this is good since it means that we're testing different cases most of the time.

However, the presence of constraints changes this situation substantially. Constraints make random selection of some values or value combinations substantially less probable. For example, a constraint might be added between A and B such that for most values of A, B is forced to 0.

```
class unconstrained;

  rand bit[3:0]          A;
  rand bit[3:0]          B;

  constraint c {
    if (A > 1) {
      B == 0;
    }
  }

endclass
```

Figure 3 - Constrained-random stimulus class

This constraint has no impact on the reachable domain of A and B individually. However, this relationship significantly alters the results of running another 1000 randomizations, as shown in Figure 4.

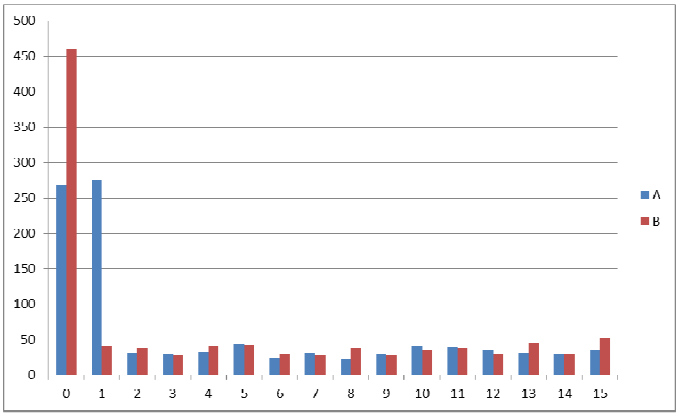

Figure 4 - Constrained-random value distribution

Note that the effect of this simple constraint is to skew the value distribution for both variables, making values 0 and 1 occur much more frequently for A and value 0 occur much more frequently for B. From a verification perspective, this is bad, since we're mostly testing the same thing over and over again.

In the simple examples above, the total stimulus space is quite small (256 total), so all individual values of A and B are selected. The total stimulus space in a real verification environment is enormous, which can result in large portions of the stimulus space remaining unexercised during random regressions. During the coverage closure phase of our verification cycle, these random-resistant corner cases are visible – at least where they intersect defined functional coverage goals. There are many techniques for dealing with random-resistant corner cases with respect to coverage closure, including the use of intelligent testbench tools, providing more direction to the random solver in the form of solve-order directives or random distributions, or simply creating directed tests. However, during the random regressions phase, these random corner cases aren't even visible.

III. Strategy-Driven Stimulus Generation Overview

This paper proposes a stimulus-generation technique to help address the challenges outlined above. In seeking to design techniques that address the challenges described above, it helps to revisit the goals of random stimulus generation. Fundamentally, the goals of random stimulus generation are:

- maximize the verification benefits realized from covering a subset of the complete stimulus space
- generate stimulus across the reachable stimulus space
- generate cases that the verification engineer is unlikely to think of

Strategy-driven stimulus generation seeks to address these goals by generating stimulus according to a set of user-controllable strategies. For example, one strategy might be to subdivide the reachable domain of each stimulus variable into at most 64 ranges and ensure that a value in each range of each variable is produced. A generation strategy like this helps to address all three goals above by:

- defining a subset of the complete stimulus space
- ensuring that stimulus is generated across the reachable domain of each variable
- in the process of generating values across the domain of each variable, generating values and value combinations that the verification engineer is unlikely to think of

In addition to addressing the same fundamental goals as random generation, the strategy-driven approach adds an important piece of information when a bug is discovered. It not only allows for reproducing a bug by running the same generation strategy with the same random seed, but also provides more information about the type of stimulus being generated when the bug occurred. If, for example, a bug was uncovered when targeting at most 64 value ranges across the domain of each stimulus variable, it would certainly make sense to continue running that generation strategy after the bug was corrected to guard against regressions. It could also be valuable to run a strategy targeting at most 128 or 256 value ranges across the domain of each stimulus variable. In other words, strategy-driven stimulus generation adds new information to that verification process that can guide what is done after a bug is discovered and corrected.

IV. DEVISING STIMULUS-GENERATION STRATEGIES

The work described in this paper focuses on strategies around two fundamental aspects of stimulus generation strategy: techniques for selecting variable target values and value ranges, and techniques for selecting variable target combinations. In both cases, the selection techniques will leverage information present in the constraint system as well as common patterns that are quite independent of the constraint system.

The simple constraint system in Figure 5 will be used as an example to show how various stimulus-generation strategies can be applied to a real world case.

```
class ethmac_tx_seq_item;
  rand frame_fmt_e      frame_fmt;
  rand bit              pad;
  rand bit              crc;
  rand bit              has_tag;
  rand bit[15:0]        len;
  rand bit[15:0]        payload_len;

  constraint c {
    len inside {[4:4096]};

    if (len < 46) { pad == 0; }

    if (frame_fmt == FRAME_FMT_ETH) {
      crc == 1;
      if (has_tag) {
        payload_len inside {[42:1500]};
        len == (payload_len + 6+6+2+4);
      } else {
        payload_len inside {[46:1500]};
        len == (payload_len + 6+6+2);
      }
    } else {
       len == payload_len;
    }
  }
endclass
```

Figure 5 - Ethernet-frame transaction class

This abbreviated UVM sequence item is used to specify transmit operations in a UVM testbench created for the Ethernet MAC design from the opencores.org website [1].

V. TARGET-VALUE SELECTION STRATEGIES

Target-value strategies can be devised from information captured in the constraint system as well as recipes that are quite independent of a specific constraint system. Several target-value selection strategies are described below.

Prior to applying a target-value selection strategy, it always makes sense to apply reachability analysis to determine the true domain of each stimulus variable. After determining the actual reachable domain of each variable, a value-range selection strategy can be applied on top of the reachable domain.

The constraint system shown in Figure 5 has an apparent value space of 131,081 and a total stimulus space of 103,079,215,104 combinations prior to analyzing the reachable domain of each variable, as shown in Table 2.

Field	Apparent Domain	Domain Size
frame_fmt	0..2	3
pad	0,1	2
crc	0,1	2
has_tag	0,1	2
len	0..65535	65536
payload_len	0..65535	65536
		131081

Table 1 - Apparent value space size

After reachability analysis, total value space is shown to be 8,186 and the total stimulus space is shown to be 402,063,576.

Field	Actual Domain	Domain Size
frame_fmt	0..2	3
Pad	0,1	2
Crc	0,1	2
Has_tag	0,1	2
Len	4..4096	4093
Payload_len	4..4096	4093
		8186

Table 2 - Reachable value space size

A. *Uniform Target-Range Strategy*

The simplest scheme for selecting target-value ranges is to evenly divide the reachable domain of each variable into at most N ranges. This strategy is similar to the functional coverage strategy of selecting an auto-bin-max setting and applying a coverpoint with automatic bins to each stimulus variable.

In the simple example shown in Figure 5, most of the stimulus fields are small. However, the len and payload_len fields have a larger reachable domain, and thus partitioning the domain of these fields is sensible.

Field	Target Value Ranges
frame_fmt	3
pad	2
crc	2
has_tag	2
len	256
payload_len	256
	521

Table 3 - Target value ranges

B. *Edge Target-Range Strategy*

Corner cases are often significant from the perspective of verification. For example, in the case above, frame padding will be enabled if the total frame length is less than 46 bytes. The Edge target-range selection strategy makes the assumption that producing several individual values at the minimum and maximum of each large variable's reachable domain will provoke more-interesting activity in the design. The Edge target-range selection strategy is an extension of the uniform target-range selection strategy. With this strategy, a maximum of N total range will be selected. In addition, M individual-value bins will be selected at the minimum and maximum of the reachable domain. In the case of the 'len' field in the example sequence item, specifying N=16 and M=4 results in four individual-value ranges selected at the minimum of the reachable domain of 'len', four individual-value ranges selected at the maximum of the reachable domain of 'len' and eight uniform-sized ranges selected in the middle of the reachable domain of 'len', as illustrated in Figure 6.

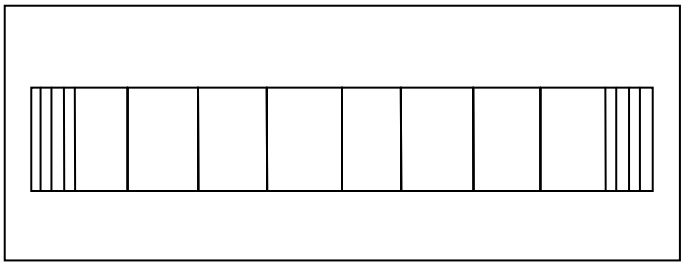

Figure 6 - Edges target-range selection example

C. *Constraint-Guided Target-Range Strategy*

The constraints used to specify the bounds of legal stimulus also provide a wealth of information about what cases may be most interesting to create. For the purposes of selecting target values and value ranges, equality and inside operators involving constant quantities are most actionable. Target values can be inferred from these operators when used either as a condition or in the body of a condition. Below are several examples showing how using constraint-guided target-value selection can help to ensure that the stimulus space is traversed more comprehensively.

```
rand bit[15:0]                  A;
rand bit[15:0]                  B;

constraint c {
  if (A inside {[1:5]}) {
    B < 10;
  }
}
```

Figure 7 - Condition-inferred target values

In the example shown in Figure 7, an inside operator is used within a condition. Given the large domain of A, targeting the range 1…5 will ensure that the constraint B < 10 is activated.

```
rand bit[15:0]                A;
rand bit[15:0]                B;

constraint c {
  if (A < 2) {
    B == 20;
  }
}
```

Figure 8 - Body constraint-inferred target values

In the example shown in Figure 8, an equality constraint is used as the body of a condition. Targeting the value B=20 will ensure that the condition A<2 occurs.

In the real world, constraint-guided target-range selection can be an extremely valuable technique for constraint systems with complex conditional logic. The target values selected by use of this technique help to ensure that all conditional-constraint branches are entered.

VI. VARIABLE-COMBINATION SELECTION STRATEGIES

Selecting target variables to exercise in combination leads to production of even more varied and complex stimulus.

A. No Combinations

The simplest approach to selecting target variable combinations is to ignore combinations entirely and allow the variable-value combinations to occur randomly. Since targeting variable combinations – especially more than two variables – greatly increases the target stimulus space, not targeting any combinations enables greater focus on variable values.

For the example used in this paper, only targeting individual variables enables targeting large numbers of value ranges. For example, when targeting a maximum of 256 value ranges, a no-combinations strategy means that a total of 521 values will be targeted. If just variable pairs were targeted using the same maximum of 256 value ranges, a total of 70,174 value-range pairs would be targeted – likely not a feasible number to verify in simulation.

B. All-Pairs Target Variable Selection

All-pairs testing, also known as pairwise testing, has been explored extensively for testing software [2]. From several case studies on software systems, empirical data suggests that 70% of defects are triggered by the combination of two or fewer input parameters and no defects were triggered by a combination of more than six parameters [3]. This data suggests that the simplistic combination-selection strategy of not selecting any pairs is actually a pretty good one. It also suggests that targeting pairs of fields, triples of fields, etc. is a good graduated strategy for selecting a subset of the full stimulus space.

As mentioned in the previous section, one downside to all-pairs target-variable selection is that the total combinations targeted expands quickly as the targeted variable combinations increase from pairs to triples to quads. For example, targeting pairwise variable combinations for the example shown above results in 926 reachable value combinations to target. If triples of variable combinations are targeted, 2,368 reachable value combinations are targeted. This rapid expansion of the number of targeted variable-value combinations has the practical effect of limiting the number of target value ranges that can be targeted as the pairwise degree increases. Nevertheless, the all-pairs strategy provides a valuable automated way to select variable combinations that produce more-complex and varied stimulus combinations.

VII. IMPLEMENTATION

As the saying goes, "The proof of the pudding is in the eating." It is critical to actually implement strategy-driven stimulus generation tool to assess whether the strategies described above have a beneficial impact on the diversity of the generated stimulus and the comprehensiveness of verification.

Among the most important requirements for the technology infrastructure used to implement a strategy-driven stimulus generator:

- extract information from existing SystemVerilog transaction/sequence item classes
- provide ways to analyze the stimulus model extracted from the SystemVerilog class to, for example, determine the reachable domain of each variable
- provide a high degree of control over the generated stimulus and provide a mechanism to 'close the loop' for stimulus-generation targets; in other words, a way to know when all goals have been met

The proof of concept implementation described in this paper was built on top of the inFact intelligent testbench automation tool. Internally, the inFact tool uses a rule-based stimulus description, but it is also able to import random variables and constraints directly from SystemVerilog transaction classes. The tool provides features that enable reachability analysis. It accepts directives to specify stimulus-generation targets and provides APIs to monitor the completion status of those targets. inFact can integrate into multiple verification environments, though this paper only focuses on integration as a UVM sequence. The reason for focusing on UVM environments is that strategy-driven stimulus generation emphasizes applying multiple sets of stimulus based on different generation strategies. Because the number of automatically-generated test cases is relatively high, changing the stimulus-generation mechanism within the testbench must be a low-overhead operation. UVM provides several features, that make swapping between multiple stimulus-generation

mechanisms extremely low-overhead, including the factory and the modularity and encapsulation provided by sequences.

In the proof-of-concept implementation described in this paper, the process for generating stimulus according to a strategy is outlined in the process diagram shown in Figure 9.

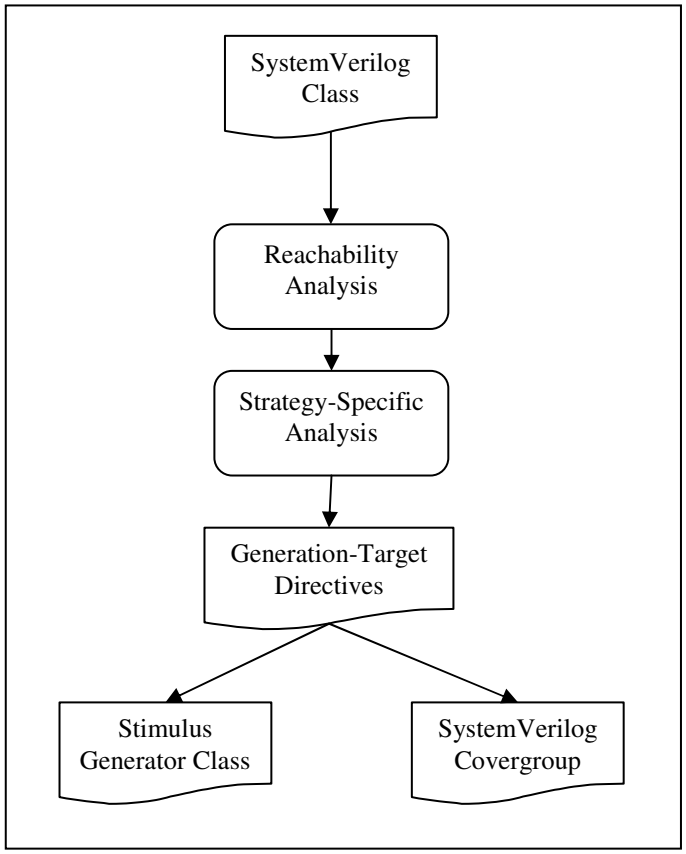

Figure 9 - Strategy-driven generator creation process

For each targeted stimulus sequence item or transaction, the constraints and random variables are imported from SystemVerilog into the inFact tool. Next, reachability analysis is applied to each of the stimulus variables to determine the reachable domain.

After determining the reachable domain of each variable, the user-specified combination of target-value generation strategy and variable-combination variable strategy is applied to the variable and constraint system. This results in a set of generation-target directives, specified in terms of target values and value ranges on variables and variable combinations to target.

The combination of the imported variable/constraint system and the generation-target directives produce a stimulus-generator class (a UVM sequence for the purposes of this paper) that will generate stimulus that is legal according to the variables and constraints and focuses on the generation targets identified during the strategy-specific analysis. A SystemVerilog covergroup is also created that contains functional coverage goals that correspond to the generation-target directives. This covergroup can be used as an independent monitor to ensure that the generated stimulus is applied to the design as intended. It can also be used as a means of grading the verification value of the strategy-generated stimulus relative to stimulus generated via open-loop random generation.

VIII. RESULTS

Grading the effectiveness of the results from strategy-driven stimulus generation is important from two perspectives. First, it is useful to understand whether the technique offers general benefits for verification beyond that offered by open-loop random stimulus. Secondly, grading the effectiveness of strategy-driven stimulus generation helps to select the highest-value strategies to apply. For example, sufficient coverage of the no-combinations strategy may be achieved during the course of regular open-loop random regressions. However, the open-loop random regressions may not achieve sufficient coverage of the pairwise strategy. In this case, applying the pairwise strategy would add verification benefit.

For the purposes of this paper, results will be analyzed both quantitatively and qualitatively using the example shown earlier in Figure 5.

A quantitative comparison of results can be done using the covergroup created based on the generation-target directives. The methodology for comparison was to identify the number of tx sequence items generated by the strategy-driven sequence to achieve the strategy goal, as monitored by the generated covergroup. Then, run 20 simulations, each with a unique seed, in which the same number of tx sequence items were generated randomly. Comparing the coverage achieved by the randomly-generated sequence items against the strategy-based coverage goal enables a comparison of whether the randomly-generated stimulus reaches the same corners of the stimulus space.

Take a stimulus-generation strategy targeting a maximum of 64 value ranges on each class variable. This strategy required 68 tx sequence items to achieve the generation goal. By contrast, generating transactions purely randomly results in 99.2% after 5 simulation runs of 68 tx transactions (340 total transactions), and no further apparent progress across 15 additional simulations. A plot comparing progress towards the strategy goal of the strategy-driven sequence and the pure-random sequence is shown in Figure 10.

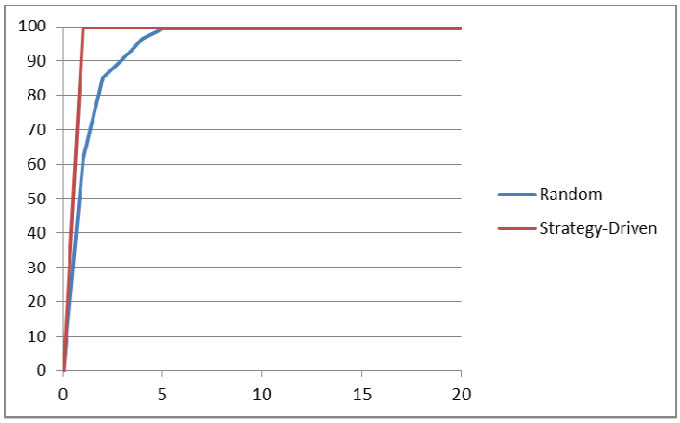

Figure 10 – Results comparison with max 64 even bins

Although pure-random generation did not entirely achieve the generation goals, this is a case where it appears that the generation targets implied by this generation strategy will be eventually achieved as a side effect of the normal open-loop random generation.

Next, consider a pairwise generation strategy again with a maximum of 64 target ranges on each variable. Results are shown in Figure 11. In this case, 180 sequence items generated by the strategy-driven generator are required to meet the generation goals. By contrast, randomly-generated tx sequence items achieve 85.5% of the pairwise goal after 12 simulations (2160 transactions), and appear not to make further progress.

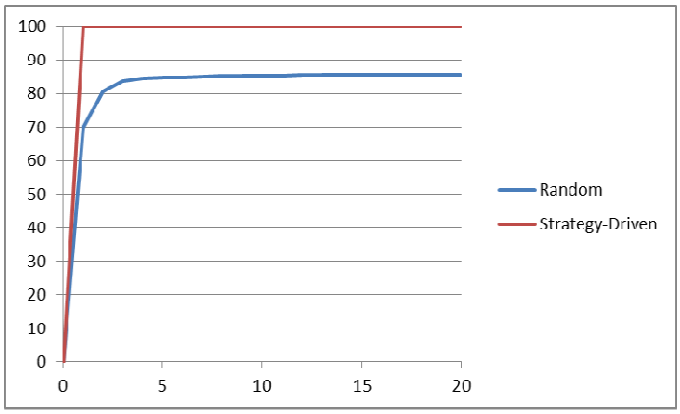

Figure 11 - Results comparison with a pairwise strategy

This is a case in which strategy-driven generation shows clear benefits in terms of efficiently hitting a random blind spot that would likely not be noticed until the coverage-closure phase of verification (if at all).

Another way to examine the results is to look at a qualitative comparison with open-loop random regressions. The UVM testbench for the OpenCores Ethernet MAC design was developed as an example, and ran through a typical set of random regressions and systematic coverage closure on verification goals derived from the functional specification. Surprisingly, though, application of a pairwise generation strategy with the Edges strategy for target-value selection uncovered a previously-undiscovered state machine lock-up involving very small packets, CRC generation, and small-packet padding.

IX. CONCLUSION

Strategy-driven stimulus generation, as described in this paper, leverages automation and existing variable/constraint stimulus models to select generation targets and systematically stimulus that satisfies generate those generation targets in order to more-comprehensively exercise random-resistant portions of the stimulus space. Several generation strategies were described. To evaluate the effectiveness of the technique, the result of strategy-driven generation was compared against open-loop random generation for the same constraint model. By adding strategy-driven stimulus generation to the verification toolkit, more metrics are available during the random-regressions phase of the verification cycle and bugs that normally would lurk in random blind spots are uncovered in a more-predictable manner.

REFERENCES

[1] I. Mohor, “Ethernet MAC 10/100 Mbps” [Online]. Available: http://opencores.org/project,ethmac

[2] D. Richard Kuhn, “Practical Combinatorial Testing” [Online]. Available: http://csrc.nist.gov/groups/SNS/acts/documents/SP800-142-101006.pdf

[3] D. Richard Kuhn, “Software Fault Interactions and Implications for Software Testing” [Online]. Available: http://csrc.nist.gov/groups/SNS/acts/documents/TSE-0172-1003-1.pdf