Toggle: A Coverage-guided Random Stimulus Generator

Stephen M. Plaza, Igor L. Markov, Valeria Bertacco
EECS Department, University of Michigan, Ann Arbor, MI 48109-2121
{splaza, imarkov, valeria}@umich.edu

ABSTRACT

Despite the increasing research effort in formal verification, constraint-based random simulation remains an integral part of design validation, especially for large design components where formal techniques do not scale. However, effective simulation often requires the construction of complex constraints to stimulate important aspects of the design.

In this paper we present Toggle, a novel solution which automatically identifies those regions of the design which are not sufficiently exercised in random simulation. Toggle then generates legal random stimulus of the primary inputs of the design to improve coverage over those regions, by augmenting the toggling activity of the signals internal to the region. In addition, Toggle can also be used to toggle a set of user-specified signals anywhere in the design. Experimental results indicate that Toggle can stimulate regions of a design in much fewer simulation cycles than random simulation, leading to simulation runs which can potentially expose bugs sooner.

1. INTRODUCTION

Verification costs greatly affect time-to-market, and increasingly complex hierarchical designs exacerbate the inherent intractability of common verification algorithms. Although exhaustive simulation is infeasible, simulation allows for partial validation of large designs whose error states are logically deep. This observation was exploited in [7] to guide formal verification algorithms and thereby increase the scalability and coverage of their hybrid approach.

Maximizing verification coverage through simulation or hybrid strategies motivates the development of fast cycle-based and event-based simulators. In [2], a parallel simulation approach is proposed that uses circuit partitioning to reduce interprocessor communication. However, fast random simulation must be coupled with constrained simulation because random simulation will not necessarily stimulate important parts of the design.

Constrained random simulation typically involves restricting inputs to certain ranges of values or asserting that certain conditions are true. Modeling more complex constraints is more prohibitive for two reasons: 1) it necessitates detailed knowledge of the design and complex specification languages by the engineer and 2) it requires an efficient mechanism for generating random stimuli that satisfy these constraints. The latter concern was partially addressed in [14], where constraints are modeled as BDDs and simulation vectors can be obtained by randomly traversing the BDD. However, this approach still requires a constraint specification and is limited by the size of the BDD.

1.1 Contributions

We present Toggle as a solution for quantifying and improving simulation coverage while decreasing engineering efforts. Toggle enables efficient identification of regions in a design that experience low activity and provides an efficient strategy for stimulating these regions while satisfying the input constraints of the design.

Consider Figure 1. First, we perform low-effort synthesis on a design so that we can exploit efficient gate-level tools available. Then, we monitor the toggle activity of each signal. We use this information to intelligently organize the signals into several partitions. The partitioning allows us to target regions with low activity for simulation. To this end, we develop a coverage metric that captures signal correlations and distributions of multi-bit combinations of internal wires. The coverage analysis suggests which partitions should be stimulated more. Our flow requires minimal user input with the goal of evenly distributing activity throughout the design.

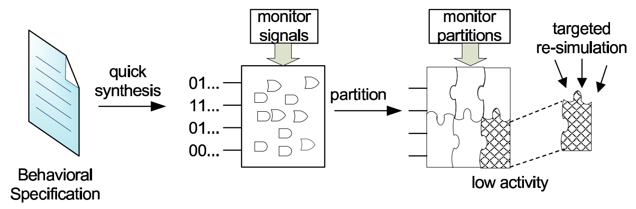

Figure 1: Toggle framework which monitors and evenly distributes stimulation in the design. First, individual signals are monitored and then groups of signals are monitored. Signal groups that experience low activity are then stimulated while satisfying the input constraints.

To perform guided re-simulation, we implement a novel SAT-based technique to toggle certain signals in a designs while satisfying the design's input constraints. In the case where the engineer specifies certain properties that need to be asserted, we can specifically toggle these checkers through our targeted simulation strategy. We show that generating even distributions of simulation vectors with SAT can be orders of magnitudes faster than random simulation.

In Section 2, we review previous work in simulation and refinement and constrained random simulation. In Section 3, we introduce Toggle as well as our partitioning and re-simulation methodologies. In Section 4, we describe in detail how we monitor activity in a design. Finally, in section 5, we describe how we re-simulate areas of a circuit to increase its toggling activity. Results are given in Section 6 that show considerable improvements over random simulation on several benchmarks.

2. PREVIOUS WORK

Simulation is an integral part of verification methodology. It can efficiently evaluate different input sequences in large designs which can be specified by designers much more easily than complex formal properties. Several techniques exist for improving random simulation and are outlined in the following paragraphs.

Constrained random simulation has been developed where desired properties are checked by asserting and constraining certain values in a design or by using formal methods like BDDs in [14]. There has also been work in using SAT to solve constraints [8, 4]. For the Boolean formula $F(V)$, the satisfiability problem involves finding an assignment to the set of variables V so that $F(V) = 1$ or discovering that no such assignment exists. A combinational circuit can be transformed into a SAT instance where a solution to that SAT instance gives the behavior of the circuit for a certain input stimuli. Both BDD or SAT-based strategies often require the addition of engineer-specified constraints. Also, the size and complexity of the constraints can make such strategies infeasible. Toggle addresses some of these limitations by performing automatic re-simulation using a novel extension to SAT.

At the instruction-level, Markov models can be used [13] to modify instruction sequences to effectively stimulate certain parts of the design. Simulation can also be refined at the gate level as in [10] where counter-examples derived from SAT create dynamic simulation patterns that help check equivalence of two circuits. Although automated, this simulation is very application-specific and is primarily useful for deriving input patterns that distinguish two designs.

Finally, simulation is used extensively with formal methods to form hybrid verification strategies that can find bugs in larger blocks of a design [7, 11]. However, formal methods still suffer from limited scalability, while their effective deployment requires specialized training and a relatively rare skill-set.

3. TOGGLE

As shown in Figure 1, the input to our framework is a behavioral design description which can be transformed to a gate-level representation so that signals can be monitored at a finer granularity. However, we note that our strategies can be abstracted to work directly at the RTL level. Toggle has two main components: coverage analysis and guided simulation.

We first monitor the toggle activity of each signal in the design. This information can be used to guide a min-cut partitioning algorithm which divides a circuit into several interacting components. This technique can extract a hierarchy from a flattened design. For a large component with no obvious logic hierarchy, this technique can discover a hierarchy that reflects the switching activity. We can then examine the inputs to each partition and monitor its activity. This provides a more powerful metric than simple toggle coverage because we are correlating several signals together and examining their collective toggling.

Using this coverage analysis, we can evenly distribute activity in a design through a SAT-based simulation strategy. Since random simulation does not usually achieve equal coverage of different design partitions, we seek to stimulate partitions with low activity, since this is more likely to expose hard-to-find bugs. Our SAT-based simulation targets these low-activity partitions while satisfying the design's input constraints. In particular, we can constrain the primary inputs to certain instructions to randomly generate streams of instructions. The SAT-based simulation strategy described can also be used to generate input patterns satisfying given constraints or properties. Also, unlike the strategies in [8, 4], our approach can use state-of-the-art SAT solvers [6] while minimizing unwanted simulation bias.

In Section 4, we introduce our coverage analysis techniques and describe our mechanism for design partitioning. Then in Section 5 we will discuss the theoretical underpinnings of our SAT-based simulation along with how we perform re-simulation specifically in Toggle.

4. FINDING INACTIVE PARTS OF A CIRCUIT

In this section, we describe how to automatically find regions of a design that are inadequately stimulated. Specifically, We use toggle activity to develop a more descriptive means of coverage analysis.

4.1 Toggle Activity of a Signal

Among the many coverage metrics used in verification, toggle coverage specifies the activity of a particular signal in a circuit. To determine the activity of a given signal s in a circuit C, we use **Shannon entropy** which estimates the amount of information associated with a signal. The entropy is calculated using the following formula:

$$E_s = -\frac{nOnes}{K}\log_2(\frac{nOnes}{K}) - \frac{nZeroes}{K}\log_2(\frac{nZeroes}{K}) \quad (1)$$

where E_s is the entropy of s, $nOnes$ is the number of simulations where $s = 1$, and K is the number of simulation vectors examined. The formula gives values that range from 0 to 1. Notice that higher entropy indicates more activity and would occur when the number of ones and zeroes are evenly distributed. The primary inputs under random simulation will have the highest entropy. However, random simulation does not stimulate all parts of the circuit equally.

For signals with low entropy, we can derive simulation that increases the activity of a given signal. For instance, if $s = 1$ for most simulation vectors, one can assert $s = 0$ and derive several solution via a SAT solver. The solutions derived will not necessarily be well-distributed but the entropy of s would increase.

As a practical example, consider the effect of random simulation on an 8-bit bidirectional counter as shown in Figure 2. Notice that increasing the number of random simulation vectors still does not generate activity at the most significant bit. Therefore, random simulation inadequately sensitizes the circuit. We show that by guiding the simulation using signal entropy, we can stimulate the counter in a way that produces more uniform coverage across all of the bits. First, we identify the bit with the smallest entropy, and then we derive a sequence of counter increments and decrements that targets this bit. Figure 3 shows that after 300 guided simulation vectors we can achieve a relatively even distribution of entropy across the 8 bits.

This analysis has certain limitations. By considering one signal at a time, one cannot tell if two signals with uniform activity are equivalent (perfectly correlated) or independent. A better coverage analysis would identify small groups of signals and consider entire combinations in terms of coverage and frequency bias (entropy).

4.2 Toggle Activity of a Partition

For designs that are broken down hierarchically, the task of selecting signals for activity analysis is straightforward. Namely, analyze the input activity of each component to assess whether adequate coverage is obtained. However, even in hierarchical designs, several internal points in a component may be inadequately stimulated. Furthermore, even if one knows at which level of granularity to perform coverage analysis, *how* to perform this analysis can be challenging in the presence of complex constraints. We now discuss a circuit partitioning strategy that works well on a variety of designs for locating where to analyze in the circuit. We utilize signal entropy discussed previously to identify areas of the design most amenable to coverage analysis. We then describe an entropy-based coverage estimation that can be used to identify circuit partitions that experience low activity. In the next section, we propose a more powerful SAT-based strategy that can analyze biasing in the presence of constraints. We use this strategy to refine the results given by the entropy-based estimation.

Automatic circuit partitioning. Circuit partitioning has been explored considerably in physical placement applications where net-cut minimization leads to smaller wirelength. The Fiduccia-Mattheyses min-cut partitioning algorithm [5] is commonly used

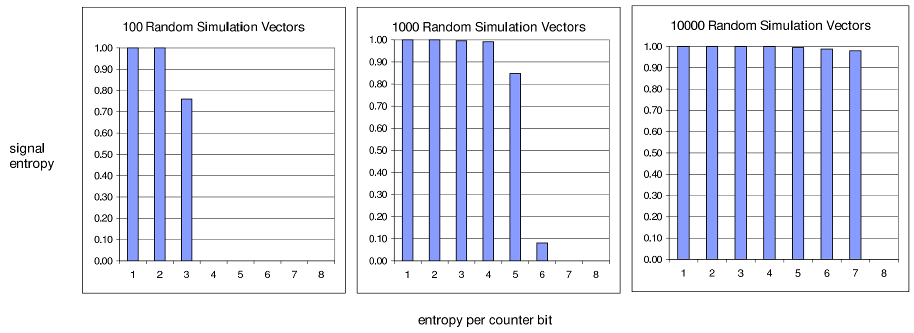

Figure 2: Random simulation for an 8-bit bidirectional counter. The entropy is given for each bit. Notice the bias that exists for smaller values even with 10000 **simulation vectors.**

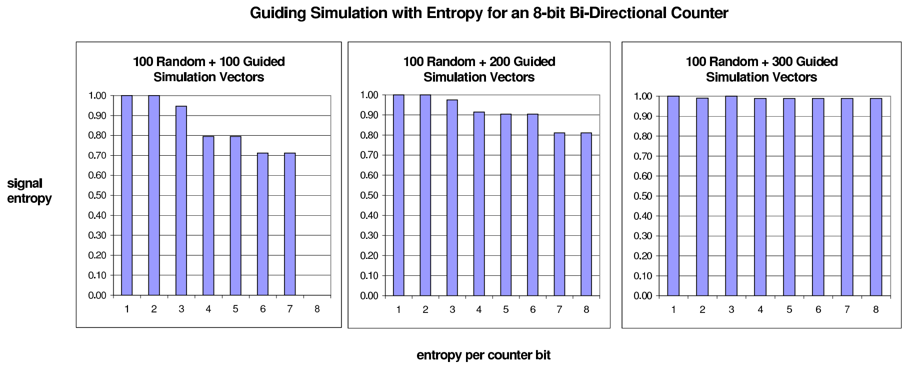

Figure 3: Guiding simulation with entropy for an 8-bit bidirectional counter. The counter is first simulated with 100 **initial random simulation vectors. Notice that the entropy is almost evenly distributed after only** 300 **guided simulation vectors.**

and runs in linear time per pass. Furthermore, multi-level extensions of this algorithm scale near-linearly to very large designs. In our work, we only perform partitioning once on a design and hence the runtime is easily amortized by verification costs, and its runtime overhead is negligible in the overall simulation flow.

We perform recursive-bisection, i.e., make multiple FM-based partitions, until we partition a circuit to a desired granularity either specified by the user or dynamically derived. Our partitioning objective is to minimize the total number of incident edges to all partitions while ensuring balance between the size of each partition. Poor partitioning results in an enormous search space for the component whereas a good partition can allow one to fully examine a larger region with a much smaller component solution space. As described before, the solution space of a component could be significantly smaller in practice with respect to the whole circuit because of the correlations between the inputs. As a result, minimizing the number of inputs to a component mitigates the effect of this correlation.

To realize our objective, we construct a *hypergraph* where each wire in the circuit represents one hyperedge that connects multiple vertices/gates. We weight the hyperedges by the signal entropy derived in the previous section. Namely, we can derive a weight as $1/E_s$ to weight wires with low-entropy higher to encourage cutting low-entropy wires. In this way, the inputs to our partitions should contain signals with lower entropy than not weighting the wires. Recursive partitioning is then performed as implemented in [9] to minimize the cut wires under this weight constraint. We specify that each hyperedge is reconstituted after each partitioning. This means that an edge that is cut is not removed from the graph but is rather redrawn around remaining nodes that are in the same partition. This is done to minimize the introduction of additional inputs for new partitions.

EXAMPLE 1. *Consider a simple circuit with gates* $A = AND(C,D)$ *and* $B = AND(C,E)$*. Assume the following partition is created* $\{A,B\}\{C,D,E\}$*. In this case The partition* $\{A,B\}$ *has 3 inputs,* C,D,E*. We reconstitute an edge around* A,B *to penalize an additional partition like* $\{A\}$ *and* $\{B\}$ *where each would have* 2 *inputs resulting in* 4 *total inputs for all partitions.*

Estimating cut activity and biasing through entropy. The cuts can be analyzed for activity to assess the amount of coverage in each partition. Consider the following simple metric for cut activity:

$$A^c_f = num\_diff\_vecs(< f_i, \cdots, f_m >) \quad (2)$$

where `num_diff_vecs` is the number of different simulation patterns on partition f's input cut. Notice though that this formula does not factor in the frequency of certain simulation vectors; it only provides the number of different simulation vectors and is therefore of limited value.

To improve upon this, we can measure the amount of information associated with the signals along the partition cut by using a full expression for Shannon entropy. This measure accounts for multiple *seen* simulation vectors along the cut which indicates simulation bias. We compute the entropy of f with m cut inputs as:

$$E_f = -\sum_{vec: freq(vec) \neq 0}^{2^m - 1} \frac{freq(vec)}{2^m} \log_2(\frac{freq(vec)}{2^m}) \qquad (3)$$

where `freq` is the frequency of a particular vector `vec` represented by an integer value. For f, we need to consider the frequency of simulation for each of the 2^m input combinations. For input combinations that do not occur, the frequency is 0 and there is no increase in entropy. For frequent input combinations, the entropy will decrease because of bias. In this calculation, the entropy increases when there are several different simulation vectors and when the vector frequencies are very low. Because the number of *legal* input combinations to the cut, i.e., those combinations that satisfy the input constraints of the design, can be less much less than 2^m, our entropy calculation is an approximation. The approximation can be improved by randomly sampling this input cut and estimating the percentage of input combinations that are legal.

We observe that with small input cuts we can perform our approximate entropy computation very efficiently with complexity linear to the number of vectors simulated. We now outline our algorithm:

```
double entropy_calculation(Partition part, Activity_count freq){
  double entropy = 0;
  max_vecs = 2 $\ll$ part.num_inputs;
  while(vec++ $\neq$ max_vecs){
    if(freq(vec) > 0){
      double prob = freq(vec)/max_vecs;
      entropy -= prob * $\log_2$(prob);
    }
  }
  return entropy;
}
```

Figure 4: Calculating entropy of a cut.

The function `entropy_calculation` takes the partition and the frequency of each possible simulation vector as input. The frequency can be computed by just traversing each simulation vector and counting the number of times each vector occurs which can be trivially accomplished during a calculation of Equation 2. Notice that `max_vecs` determines the complexity of this procedure if it is greater than the number of simulation vectors. If the number of inputs is small, each vector can be stored efficiently as sequence of successive integers from 0 to *max_vecs* which allows for quick $O(1)$-time lookup of frequencies. When the cuts are large, we can alternatively employ hash techniques to quickly compute frequency.

Often, the number of input combinations possible over a partition's inputs is much greater than the number of simulation vectors applied. Also, there is often a great difference in the number of inputs to each partition resulting in entropy that is dependent more on the number of inputs to each partition than to the input's toggle activity. As a result, we modify Equation 3 by calculating entropy with respect to the number of simulation vectors K which we define as:

$$E_f^K = -\sum_{vec: freq(vec) \neq 0}^{2^m - 1} \frac{freq(vec)}{K} \log_2(\frac{freq(vec)}{K}) \qquad (4)$$

This approximation is useful in our environment because most of the partition cuts that we examine have solution spaces much larger than number of simulation vectors applied.

In the next section, we leverage these coverage metrics to guide a fast, SAT-based simulation approach that produces even coverage in the presence of complex constraints and design hierarchies.

5. TARGETED RE-SIMULATION

In this section, we describe how we can perform guided simulation using SAT. First, we discuss the theoretical underpinnings of our SAT-based simulation approach and show that we can achieve random simulation via SAT using any commercially available SAT solver. Then, we explain how properties can be stimulated using our simulation approach. Finally, we explain how to use this guided simulation to target partitions with low entropy in Toggle to reduce simulation bias.

5.1 Random Simulation with SAT

For the Boolean formula $F(V)$, the satisfiability problem involves finding an assignment to the set of variables V so that $F(V) = 1$ or discovering that no such assignment exists. SAT problems are typically expressed in CNF, and combinational circuits can be converted to a CNF with complexity linear to the size of the circuit. For a circuit with n inputs, there would logically be 2^n satisfying assignments possible for the corresponding SAT instance. We now discuss how to derive these satisfying assignments randomly.

Theoretical background. It has been shown in [12] that the number of solutions to a SAT problem does not change the inherent difficulty of solving the problem. More specifically, [12] shows that a randomized polynomial-time reduction of an arbitrary Boolean formula can generate a set of corresponding formulas where one of them has only one solution, i.e., unique SAT (U-SAT) with probability $\geq \frac{1}{2}$. In the following paragraphs, we discuss only an aspect of this result that is relevant to our work. Specifically, we explain that random constraints or Boolean clauses can be added to a SAT problem that partitions the solution space in roughly equal halves with high probability.

Assume a SAT instance f with variables $x_1, x_2, ..., x_n$ that has solutions $v \in \{0,1\}^n$. To partition this space, we randomly pick an assignment $w \in \{0,1\}^n$ and add the following constraint to f: $v \bullet w = 0$ in base-2 arithmetic.

This can be expressed as follows:

$$f \wedge (x_{i_1} \oplus x_{i_2} \oplus \cdots \oplus x_{i_j} \oplus 1) \qquad (5)$$

where i_j represents the indices of x_i where w is 1. In other words, this adds an XOR constraint whereby an even polarity of x_{i_j} determined by w need to be assigned to 1.

Alternatively, a CNF representation can be given as:

$$f \wedge (y_1 \Leftrightarrow x_{i_1} \oplus x_{i_2}) \wedge (y_2 \Leftrightarrow y_1 \oplus x_{i_3}) \wedge \cdots \\ \wedge (y_{j-1} \Leftrightarrow y_{j-2} \oplus x_{i_j}) \wedge (y_{j-1} \oplus 1) \qquad (6)$$

EXAMPLE 2. *Consider the SAT formula* $(a+b+c')(b+d)$. *This solution space can be partitioned by generating an XOR clause*

#sims	Random			SAT-based		
	#diff_sims	entropy	(s)	#diff_sims	entropy	(s)
64	64	1.00	0	64	1.00	2
128	128	1.00	0	128	1.00	5
256	253	1.00	0	256	1.00	11
512	499	0.99	0	506	1.00	30
1024	991	0.99	0	1003	1.00	98

Table 1: The quality of simulation for random and SAT-based random simulation on circuit *alu*4. Notice that SAT-based simulation is comparable to random simulation at evenly stimulating a design because the entropy of the primary inputs is similar for both approaches. 1 is the highest entropy possible. *alu*4 has 2403 gates and 14 inputs.

for the randomly generated w : $a = 1, b = 1, c = 0, d = 0$. *The resulting CNF would be* $(a+b+c')(b+d)(y \Leftrightarrow a \oplus b)(y \oplus 1)$.

As mentioned, [12] shows that this roughly partitions a search space evenly with high probability. However, very uneven partitions are possible with low probability. If S_f is the set of all solutions of f, then the addition of constraints from k random w vectors reduces the solution space to roughly $2^{-k}|S_f|$.

Random simulation with SAT. Partitioning a solution space via Valiant-Vazirani can be adapted to generate an even distribution of simulation vectors. According to this formulation, each randomly generated constraint randomly reduces the solution space. Therefore, ignoring the case where an all 0 solution exists, any particular solution can be generated through XOR-based reductions to U-SAT.

In the simple case, where random simulation is desired over the primary inputs of a circuit with no constraints, we can perform SAT-based simulation in a straightforward manner. For a circuit C with n inputs, we can approximate that the addition of n XOR-constraints will produce a randomized U-SAT. Because we are modifying the SAT instance, any SAT solver can be used to derive the solution. Our approximation might result in UNSAT instances or instances with many solutions; however, we expect a decent distribution of simulation.

To demonstrate this even distribution, we show the entropy and number of different simulation vectors (`diff_sims`) derived using SAT-based simulation on the circuit *alu*4 in Table 1. We also show the entropy for random simulation. The results indicate that the SAT-based strategy achieves competitively high entropy to traditional random simulation.

Notice, however, that the runtimes can become large using the SAT-based approach. This is due, in part, to the large number of XOR-based clauses added (in this case 14). In other words, approximately finding U-SAT solutions can be computationally expensive because several of the reductions will be UNSAT. For instance, 399 UNSAT instances were derived in the process of solving for 1024 different vectors for the circuit in Table 1. The non-linear increase in runtime can be attributed to the use of the SAT solver incrementally where the cost of disabled old constraints in the SAT solver can negatively impact future SAT runs and sometimes outweigh the benefit of incrementality.

If only 64 evenly distributed input vectors are desired for circuit C where $2^n \gg 64$, a more efficient procedure can be used. In this case, 6 XOR constraints can be added to approximately partition the solution space into $1/64$ of the original size with high probability. By adding different random sets of 6 XOR constraints 64 times, we can achieve a distribution of values for the circuit. We define the addition of more XOR constraints as increasing the **resolution** of the solution space. More formally, we say that resolution is equal to $2^{num(XOR)}$ where the inverse of this is the probabilistic size of the partitioned solution space.

#sims	#XORs	#diff_sims	entropy	(s)
64	6	60	0.98	0
128	7	120	0.98	1
256	8	243	0.99	1
512	9	466	0.98	5
1024	10	960	0.99	25

Table 2: The distribution of entropy using SAT-based random simulation using fewer XOR constraints than in Table 1 . Notice that we improve the runtime considerably while still retaining high entropy.

In Table 2, we show the entropy and runtimes for deriving simulation vectors where $log_2(\#sims)$ XOR constraints are added. The results indicate a considerable improvement in runtime with nominal impact to the entropy/quality of the simulation.

5.2 Stimulating Specified Properties

The previous section demonstrated that SAT randomization via the addition of random XOR constraints can be used to randomly stimulate a design's inputs. However, our technique is capable of handling any type of engineer-specified constraint whereas random simulation might not scale. For simple constraints such as input biasing, random simulation is clearly sufficient. However, the designer may wish to find simulation that asserts certain specified properties in the middle of design C. Random simulation may rarely assert this value. Also, solving for multiple solutions using SAT with no randomization could produce biased simulation.

With our SAT-based strategy, achieving simulation uniformity while asserting a property is a trivial extension of the previous scenario. The main difference is that the constrained C^* will contain an often unknown fewer solutions than C, i.e., $|S_{C^*}| < |S_C|$. If $|S_{C^*}|$ is similar in size to the number of desired simulation vectors, one can just exhaustively enumerate all solutions in the SAT instance. Otherwise, we can add XOR constraints to achieve the desired resolution. We can reduce SAT solving runtimes by minimizing the number of added XOR constraints with similar effect as in Table 2.

The trade-off between stimulating a design with random simulation versus SAT-based simulation is illustrated in Figure 5a. In this example, we show the original solution space of C as a superset of the reduced solution space of C^* which is sparse due to adding engineer-specified constraints. By randomly picking solutions in C, one rarely achieves a solution in C^*. However, if C^* is randomized by adding XOR constraints, we can derive legal inputs that satisfy these constraints. By adjusting the number of XOR constraints added, we can maximize the chance that solutions to C^* are found while still achieving a distribution of simulation.

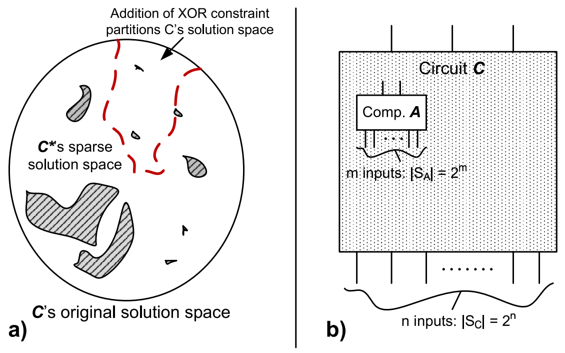
Addition of XOR constraint partitions C's solution space
C*'s sparse solution space
C's original solution space
a)
Circuit C
Comp. A
m inputs: |S_A| = 2^m
n inputs: |S_C| = 2^n
b)

Figure 5: a) A sparse solution space from adding constraints to C. b) Simulating component A within circuit C.

5.3 Partition Targeted Simulation

In Toggle, we strive to improve the activity of partitions that experience low entropy with the goal of exploiting new behavior in the circuit. We have shown that SAT can be used to generate even distributions of simulation in the presence of constraints. However, we now desire to automatically stimulate internal partitions of the design while satisfying the design's input constraints.

Simulating a component within a design. From Figure 5b., we show the problem of stimulating component A from the primary inputs of circuit C. Notice that component A with m inputs is positioned far away from the n primary inputs of C where simulation is performed. It is possible that A is not adequately stimulated in this example. One possible solution would be to randomly find a solution to A and check whether this satisfies the input constraints to C. However, this procedure would tend to be costly because several input combinations for A would be impossible due to the interaction with the rest of C. In other words, the number of solutions of A with respect to C or $|S_{A/C}|$ is much smaller than 2^m. A sparse solution space would require prohibitive amounts of SAT calls.

To mitigate this problem, we derive an extension to the SAT-based simulation theory presented previously. A can be considered a sub-space of C. The CNF of C can be defined as a $CNF(C - A) \wedge CNF(A)$. Therefore a solution to C implies a solution to A. By randomly generating w_A to create XOR constraints specific to A, we can partition A's solution space and conjoin the result with $CNF(C-A)$. Any derived solution to C will produce a solution to A with the resolution determined by the number of XOR constraints added. This approach could also produce $UNSAT$ instances due to very sparse or uneven solution spaces. However, the resolution can be dynamically decreased to reduce these occurrences but still ensure a distribution of solutions.

Algorithm. Function `partition_simulate` takes a partition where even stimulation is needed along with the number of simulation vectors desired and the circuit as arguments. The resolution gives the number of *bins* a solution space can be divided into and is derived from the number of desired simulation vectors. In other words, if we desire 16 simulation vectors, a resolution of 4 is required and 2 XOR constraints are needed. We then construct the SAT instance of the circuit with `construct_cnf` and add additional constraints. Although we do not do this here, one could assert several conditions about the solution space that should be additionally checked. Also, one could add constraints reflecting the correlations of the inputs to the partition to better model the partition's solution space with respect to the whole design.

```
void partition_simulate(Partition part, Circuit C, int num_sims){
 num_xor = $log_2$(num_sims);
 CNF = construct_cnf(C);
 add_additional_constrs(CNF);
 while(num_sims–){
   add_xor_constrs(num_xor, part, CNF);
   if(Solve(CNF, solution)) {
     add_solution(solution);
     add_blocking_clause(solution, CNF);
   }
 }
}
```

Figure 6: Evenly simulating a partition.

We then *derive* `num_sims` solutions that satisfy the constraints of the circuit along with the resolution of *part*. Different constraints are added for each pass of the while loop by function `add_xor_constrs`. If a satisfying solution is found, `add_solution` adds the derived solution. `add_blocking_clause` adds a constraint to the CNF called a **blocking clause** that assures that the solution cannot be rederived. Implicit in this algorithm is the fact that we are incrementally calling a SAT solver to utilize information learned that is consistent across many calls to `Solve`.

circuit	#gates	#parts	avg. entropy	lowest entropy
spi	3010	30	0.95	0.63
systemcdes	3196	31	0.92	0.57
tv80	6847	68	0.89	0.17
systemcaes	7453	74	0.98	0.52
ac97_ctrl	10284	102	1.00	0.95
usb_funct	11889	118	1.00	0.86
aes_core	20277	202	0.75	0.41
wb_conmax	28409	284	0.91	0.67
ethernet	37634	376	0.99	0.54
des_perf	94002	940	0.91	0.50

Table 3: Entropy analysis for circuits partitioned so that the average partition size is approximately 100**. The maximum entropy for each circuit is** 1.0

Dynamically constraining a partition. It is possible that the SAT solver will not find solutions to the constrained space. We propose as future work a technique to automatically mitigate this occurrence. When a call to a SAT solver returns UNSAT, we can analyze the conflicts that occurred during the SAT solving algorithm. In particular, we can examine the input cut of the partition and check when an assignment of a subset of the inputs necessarily implies a conflict outside of the partition. This would indicate a situation where inputs are correlated in a way that would make solutions of the partition's solution space inconsistent with the rest of the circuit. We could add a constraint to model this correlation to prune the false solution space of the partition. The constraint added with this type of analysis would be independent of the XOR-constraints and be used for future analysis of the partition.

6. EXPERIMENTS

We have already shown some results indicating that SAT-based simulation can evenly stimulate a design. In this section, we will demonstrate the effectiveness of Toggle. We will show that a design can be stimulated unevenly when using unguided random simulation and that our guided simulation strategy reduces these biases. Furthermore, we will show the effectiveness of stimulating a partition using SAT versus random simulation.

The SAT algorithms that we developed were built on MiniSAT [6] and required only a few lines of code. We used hMetis [9] to perform the recursive bisections needed for our circuit partitioning. We simulated the circuit using bit-parallel simulation and considered only the combinational portions of the design. Our circuits are from the IWLS 2005 suite [15].

6.1 Assessing Simulation Bias in a Circuit

In this section, we show that the entropy varies between each design and that the worst partition in a design has relatively low entropy making it a good candidate for re-simulation. A partition with low entropy suggests that a part of the design is not properly stimulated under random simulation. By focusing on these areas for re-simulation, we hope to expose new circuit behavior that can expose bugs more quickly.

In Table 3, we show a set of circuits ordered by their size. We partitioned each design using signal entropy so that each partition would be approximately 100 gates in size. The number of partitions that were used is given by `#parts`. The average entropy of all of the partitions is given as `avg. entropy`. The maximum possible entropy is 1.00. The entropy for the partition with the most inactivity is given by `lowest entropy`.

circuit	our SAT-based sim		random sim		entropy
	#SAT calls	(s)	#SAT calls	(s)	(s)
spi	32	1	11168	6	0
systemcdes	43	2	-	time-out	0
tv80	38	6	-	time-out	1
systemcaes	44	9	12163	10	1
ac97_ctrl	32	17	402	14	1
usb_funct	32	19	32	17	0
aes_core	67	15	662655	209	1
wb_conmax	32	170	969723	200	1
ethernet	33	375	141043	1429	3
des_perf	216	1582	14669	1578	2

Table 4: Comparing SAT-based re-simulation with random re-simulation over a partition for generating 32 vectors. The runtime is also shown for the entropy analysis performed. Time-out is 10000 seconds.

Notice that for most of the circuits, the average entropy is pretty close to 1.00; however, there is usually at least one partition that is considerably worse as in `tv80` and `aes_core`. We have also observed that there are usually a small number of partitions that show very low activity. We can exploit this by performing re-simulation mainly over this small number of poorly covered partitions.

6.2 Random vs. SAT-guided Simulation

We show that evenly simulating a partition via random simulation at the partition's inputs is often much slower than using SAT-guided simulation. In Table 4, we perform re-simulation of a partition using 32 vectors. The partition with the worst entropy is chosen.

We show the number of calls to a SAT engine and the runtime required for the SAT-based simulation strategy to generate 32 simulation vectors under the `SAT-based sim` columns. Notice that close to a minimal number of SAT calls are required for most of the circuits to generate 32 simulation vectors. The only significant exception is the circuit `des_perf`. By performing conflict analysis on failed SAT attempts in future work, we can mitigate the number of extra SAT calls. In the next set of columns, `random_sim`, we show that random simulation over a partition's input rarely satisfies the constraints of the entire design resulting in many failed SAT calls. Notice in several cases, we achieve orders of magnitude runtime improvement, or random simulation times out as in `systemcdes` and `tv80`.

We also show the runtime for the entropy calculation given by `entropy(s)` in the last column. The runtime for the coverage analysis is very fast in general and scales well for the larger designs.

7. CONCLUSION

We proposed Toggle to automatically stimulate areas of a design that experience low coverage. To achieve this goal, 1) we developed a strategy for analyzing the toggling of several signals using entropy and 2) we developed a novel simulation framework using SAT that allows for an even distribution of simulation in the presence of complex constraints.

We show results that indicate that our SAT-based simulation can be useful for improving the activity of parts of the design that are inadequately stimulated. We also show that adding only some XOR constraints is often sufficient for evenly stimulating a design thus mitigating the complexity of our original SAT formulation. Our results also indicate that guided simulation is useful for removing biases in random simulation and outperforms random simulation considerably in several benchmarks.

Our efforts to improve simulation quality could also be instrumental in improving hybrid verification strategies guided by simulation and synthesis approaches that require high-quality simulation to guide their optimizations.

8. REFERENCES

[1] F. Aloul, B. Sierawski, and K. Sakallah, "Satometer: how much have we searched?", *TCAD*, pp. 995-1004, 2003.

[2] K.-H. Chang, H. Wang, Y. Yeh, and S. Kuo, "Automatic partitioner for distributed parallel logic simulation", *IASTED*, 2004.

[3] A. Das, P. Basu, A. Banerjee, P. Dasgupta, P. Chakrabarti, C. Mohan, L. Fix, and R. Armon, "Formal verification coverage: computing the coverage gap between temporal specifications", *ICCAD*, pp. 198-203, 2004.

[4] R. Dechter, K. Kask, E. Bin, and R. Emek, "Generating random solutions for constraint satisfaction problems", *AAAI*, pp. 15-21, 2002.

[5] C. Fiducia and R. Mattheyses, "A linear-time heuristic for improving network partitions", *DAC*, pp. 175-181, 1982.

[6] N. Een and N. Sorensson, "An extensible SAT-solver", *SAT '03*, `http://www.cs.chalmers.se/∼een/Satzoo`.

[7] P.-H. Ho, T. Shiple, K. Harer, J. Kukula, R. Damiano, V. Bertacco, J. Taylor, and J. Long, "Smart simulation using collaborative formal and simulation engines", *ICCAD*, pp. 120-126, 2000.

[8] W. Jordan, "Towards efficient sampling: exploiting random walk strategies", *AAAI*, pp. 670-676, 2004.

[9] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: applications in VLSI domain", *TVLSI*, pp. 69-79, 1999.

[10] A. Mishchenko, S. Chatterjee, R. Jiang, and R. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification", *ERL Technical Report*, Berkeley. `http://www.eecs.berkeley.edu/∼alanmi/publications/`.

[11] S. Shyam and V. Bertacco. "Distance-guided hybrid verification with GUIDO", *DATE.*, pp. 1211-1216, 2006.

[12] L. Valiant and V. Vazirani. "NP is as easy as detecting unique solutions", *Theor. Comput. Sci.*, pp. 85-93, 1986.

[13] I. Wagner, V. Bertacco, T. Austin, "StressTest: an automatic approach to test generation via activity monitors". *DAC*, pp. 783-788, 2005.

[14] J. Yuan, A. Aziz, C. Pixley, and K. Albin, "Simplifying Boolean constraint solving for random simulation-vector generation", *TCAD*, pp. 412-420, 2004.

[15] `http://www.opencores.com/`