A Functional Validation Technique: Biased-Random Simulation Guided by Observability-Based Coverage

Serdar Tasiran
Compaq Systems Research Center, Palo Alto, CA
Serdar.Tasiran@compaq.com

Farzan Fallah
Fujitsu Labs of America, Inc., Sunnyvale, CA
farzan@fla.fujitsu.com

David G. Chinnery, Scott J. Weber, Kurt Keutzer
University of California, Berkeley, CA
{chinnery,sjweber,keutzer}@ic.eecs.berkeley.edu

Abstract

We present a simulation-based semi-formal verification method for sequential circuits described at the register-transfer level. The method consists of an iterative loop where coverage analysis guides input pattern generation. An observability-based coverage metric is used to identify portions of the circuit not exercised by simulation. A heuristic algorithm then selects probability distributions for biased random input pattern generation that targets non-covered portions. This algorithm is based on an approximate analysis of the circuit modeled as a Markov chain at steady state. Node controllabilities and observabilities are estimated using a limited depth reconvergence analysis and an implicit algorithm for manipulating probability distributions and determining steady-state behavior. An optimization algorithm iteratively perturbs the probability distributions of the primary inputs in order to improve estimated coverage. The coverage enhancement achieved by our approach is demonstrated on benchmarks from the ISCAS89 and VIS suites.

1. Introduction

Functional validation continues to be the bottleneck during the design of a sequential circuit. The gap between the capacities of exhaustive verification methods and the requirements of current designs has led to the development of "semi-formal" methods: methods which trade some of the comprehensiveness of formal verification for computational efficiency. Many of these techniques add some degree of formal assurance to existing simulation-based validation frameworks. One key tool in making simulation more formal and directing input stimulus generation is coverage analysis. Validation coverage metrics act as heuristic measures that quantify the completeness of verification and guide input stimulus generation by identifying portions of designs that are not exercised adequately.

We present a semi-formal method based on "tag coverage" [4]. Tag coverage addresses a major weakness of code coverage metrics by augmenting them with an observability requirement. A code segment is considered covered during simulation only when it is exercised *and* affects an observed node of the circuit. Experiments indicate that tag coverage is a significantly better measure of validation quality than other code coverage metrics [5].

Our method uses tag coverage to guide biased random generation of input vectors (Figure 1) as part of an iterative simulation loop. Biased random simulation can explore deep into the state-space and potentially uncover errors that would be difficult to excite with other, more expensive methods. On an industrial design, typically, a large portion of the simulation time is spent running biased random simulations, and improvements to these simulation runs can translate to significantly improved quality of verification. But it is well-known that, even for some combinational circuits, unless the primary input biases are chosen carefully, random simulation is not able to exercise designs well [10, 11]. This provides the motivation for our work. We investigate algorithms for choosing primary input biases in order to achieve a given tag coverage goal.

Section 2 presents the preliminaries and introduces tag coverage. Section 3 describes our approach for coverage-guided biased random simulation. Experimental results are presented in Section 4.

2. Preliminaries

2.1. The Circuit Model

The sequential circuit under consideration, $\mathcal{C} = (\mathcal{I}, \mathcal{O}, \mathcal{L}, \mathcal{G}, \mathcal{N})$, is an interconnection of a set of latches $\mathcal{L}$ and gates $\mathcal{G}$. The primary inputs $\mathcal{I}$ and primary outputs $\mathcal{O}$, along with all of the latch and gate inputs and outputs comprise the set of nodes $\mathcal{N}$. Every circuit node $n \in \mathcal{N}$ takes on values from a finite domain $Dom(n)$. Each gate computes an output as a function of its inputs. A *state* s of $\mathcal{C}$ is an assignment to each latch. For $l \in \mathcal{L}$, $s(l)$ is the value of l at state s.

2.2. Tag Coverage

Tag coverage is defined on the register-transfer level description of a circuit in a hardware description language (HDL). It is an observability-based metric as it pays special attention to the case where a design error is stimulated but its effects never observed during simulation. Consider the following example:

```
i = j + k;
x = c * i;
if (a > 0)
      o = x;
else
      o = 0;
```

Suppose that the portion of the circuit that computes `i` has an error. If the inputs to the simulator are such that whenever `i` has an error, `c = 0` or `a ≤ 0`, and `o` is the only variable that gets compared with the reference model during simulation, the fact that `i` has an error will never be detected.

A design error is assumed to exhibit itself in the form of an erroneous assignment to a variable in the HDL code. Assignments are considered one at a time by "tagging" the variable on the left-hand side of the assignment with a $+\Delta$ (or $-\Delta$) which represents an assignment higher (lower) than intended to the variable. During simulation, a variable v can take on values of the form μ, $\mu+\Delta$, and $\mu-\Delta$, the latter two of which are called *tagged values*. A simulation calculus is defined in [4] to handle tagged values. A variable in the simulated circuit is called *observable* if its value is checked against a reference model or a specification. In this paper, we assume without loss of generality that the primary outputs are the observed variables. If an observed variable takes on a tagged value, the tag is said to be *covered*. In the example above, when a tag corresponding to the first line is being considered, `o` would take on a tagged value only if `c ≠ 0` and `a > 0`. Assignments in HDL correspond to nodes in our circuit description, thus, a tag τ is represented by a pair $\tau = (n, \Delta)$ or $\tau = (n, -\Delta)$, where $n \in \mathcal{N}$. While the error model underlying tag coverage may seem simplistic, it provides a fine grain measure of how adequately a design is exercised and has been found to be a superior measure of validation quality than other RTL code coverage metrics [5].

2.3. Modeling Biased-Random Simulation

During biased random simulation the primary inputs of the circuit $\mathcal{C}$ are driven by a sequence $\Pi = \pi_0, \pi_1, \pi_2, ...$, where π_j specifies the values of the primary inputs during the jth clock cycle and each π_j is referred to as an *input vector*. For a primary input $i \in \mathcal{I}$, $\pi_j(i) = \nu$ denotes the fact that the value of i at clock cycle j is given by ν. The selection of input vectors throughout one run of biased-random simulation is controlled by a probability distribution function $\mathcal{R}$, where $\mathcal{R}_i(\nu) = \alpha$ implies that input i is assigned to ν with probability α. $\mathcal{R}$ is called a *probability distribution* (PD). In this scheme, the probability of a transition from one circuit state to another is fixed, and therefore, the circuit driven by biased random inputs can be modeled by a Markov chain $M_{\mathcal{C},\mathcal{R}}$.

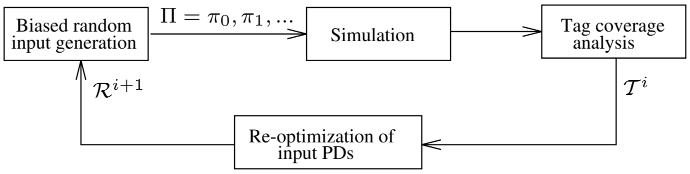

Figure 1. Biased-random simulation with coverage feedback

3. Our Method

Given a circuit $\mathcal{C} = (\mathcal{I}, \mathcal{O}, \mathcal{L}, \mathcal{G}, \mathcal{N})$ and a set of tags $\mathcal{T}$, our goal is to determine PDs (i.e., functions $\mathcal{R}^1, \mathcal{R}^2, ..., \mathcal{R}^k$ as described above) for the primary inputs $\mathcal{I}$ so that a biased random simulation run using $\mathcal{R}^1$ followed by a run that uses $\mathcal{R}^2$, and so on, covers as large a subset of $\mathcal{T}$ as possible. The high level structure of the algorithm is depicted in Figure 1. The PD optimization algorithm analyzes the set of tags to be covered, $\mathcal{T} = \mathcal{T}^0$, and the circuit structure to arrive at an initial PD $\mathcal{R}^1$ for the primary inputs. The simulator takes in a circuit described in Verilog and input vectors generated according to $\mathcal{R}^1$. A simulation run using $\mathcal{R}^1$ is performed until the tag coverage stops to improve. Then the PD optimization algorithm is run again to target the set of remaining tags, $\mathcal{T}^1$, to obtain a new PD, $\mathcal{R}^2$. This process is repeated until no more tags are covered. At the core of this method is the algorithm for optimizing $\mathcal{R}$ given $\mathcal{C}$ and $\mathcal{T}$, which is explained in the next section.

3.1. Optimizing Input Biases

We formalize the goal of optimizing primary input PDs using "merit functions" which, for a given $\mathcal{C}$, $\mathcal{T}$, and $\mathcal{R}$, estimate the number of tags that have a high likelihood of being detected during simulation. For each tag $\tau \in \mathcal{T}$ a figure of merit $\psi(\tau, \mathcal{R})$ that estimates this likelihood is computed. The merit function for $\mathcal{T}$ is given by $\psi(\mathcal{T}, \mathcal{R}) = \sum_{\tau \in \mathcal{T}} \psi(\tau, \mathcal{R})$. We then improve the merit function by gradually modifying $\mathcal{R}$.

The computation of $\psi(\tau, \mathcal{R})$ is done using a statistical analysis of the circuit behavior. We estimate the controllability (the probability that τ will be exercised) and the observability (the probability that τ will get propagated to an observable node) of τ. Since we run long simulations, we approximate the behavior of the circuit by the steady-state behavior of the Markov chain $M_{\mathcal{C},\mathcal{R}}$, which is explained in Section 3.3. Observability computation for tags is presented in Section 3.4. Section 3.5 defines various merit functions and describes our algorithm for optimizing them.

3.2. Data Structures and Subroutines

This section presents data structures and subroutines used in the probabilistic analysis of the circuit.

3.2.1. Multi-Valued Functions

A multi-valued function (MVF) is a data structure for representing a function $f(x_1, ..., x_n)$ with multi-valued inputs and a multi-valued output y. An MVF consists of a set of multi-valued decision diagrams (MDD's), one per element in the domain of $y = f(x_1, ..., x_n)$. f_p, the MDD corresponding to $p \in Dom(y)$ evaluates to 1 for input minterm $(a_1, ..., a_n)$ if and only if $f(a_1, ..., a_n) = p$. The primary use of an MVF package in our method is to express the value of a circuit node in terms of other variables in its transitive fan-in.

3.2.2. Computing the output PD for an MVF

Suppose that an MVF $f(x_1, ..., x_n)$ is given and we would like to compute the PD for the output variable y. This operation is performed by computing the conditional probability $P(f_p)$ for each $p \in Dom(y)$ as follows

$$P(f_p) = \sum_{\eta \in Dom(x_i)} P(x_i = \eta) \cdot P(f_p(x_i = \eta)) \qquad (1)$$

where $f_p(x_i = \eta)$ is the "cofactor" of f_p with respect to $\eta \in Dom(x_i)$. Iterating over the x_i's yields $P(f_p)$ in terms of the PDs for the x_i's. With proper memoization, this computation is linear in the size of the MDD for f_p. The PDs for the MVFs representing node functionalities are computed in this way using the PDs for the primary inputs and latches. The computation of the latch PDs is explained in the next section.

3.3. Determining Node PDs

3.3.1. Computing Latch PDs at Steady-State

Despite advances in Markovian analysis methods [6, 3], for most practical circuits, it is not feasible to explore or store information about every individual state in the state space. To avoid this, we employ an approximation based on [9]. Consider a state $s = (s_1, s_2, ..., s_{|\mathcal{L}|})$. We approximate σ_s, the steady state probability of being at state s, by $\prod_{i=1}^{|\mathcal{L}|} P(l_i = s_i)$, where $P(l_i = s_i)$ is the probability that l_i has value s_i at steady-state. Then we only need to determine the values of $P(l_i = \lambda)$ for each λ in the domain of l_i (called "line probabilities" in [9]). This amounts to computing PDs as if the PDs for the latches were independent. [9] reports that power estimates computed using this approximation are correct within 3% for a large number of sequential benchmark circuits. When it is important to preserve the correlation between the PDs of a group of latches, this can be handled in our framework by treating them as a multi-valued latch.

Let the next state of the circuit be described as a function of its present state and inputs as

$$\begin{aligned} ns_1 &= f_1(i_1, ..., i_{|\mathcal{I}|}, ps_1, ..., ps_{|\mathcal{L}|}) \\ &\vdots \\ ns_{|\mathcal{L}|} &= f_{|\mathcal{L}|}(i_1, ..., i_{|\mathcal{I}|}, ps_2, ..., ps_{|\mathcal{L}|}) \end{aligned} \qquad (2)$$

where i_j and ps_l, denote the values of the jth primary input and the output of the lth latch in the present state, ns_p denotes the value of latch p in the next state. The f_i denote the values computed for the next states by the circuit $\mathcal{C}$. The steady state of the circuit corresponds to a fixed point where for each latch l_i and for each $\nu \in Dom(l_i)$

$$P(ns_i = \nu) = P(f_i(i_1, ..., i_{|\mathcal{I}|}, ps_1, ..., ps_{|\mathcal{L}|}) = \nu) = P(ps_i = \nu)$$

To compute the fixed point, we start with an initial guess for the PDs of the latches and substitute these on the right-hand side of Eqn 2 along with PDs for primary inputs. Then we compute $P(f_j(i_1, ..., i_{|\mathcal{I}|}, ps_1, ..., ps_{|\mathcal{L}|}) = \nu)$ for each j and obtain a new PD for the latches. We re-substitute these and iterate until the rate of change in the PD of all latches from one iteration to the other falls below a threshold. Linear convergence of these iterations is guaranteed under minor restrictions on the circuit structure [9] assuming that the f's are computed exactly[1]. We refer to the computation of $P(f_j(i_1, ..., i_{|\mathcal{I}|}, ps_1, ..., ps_{|\mathcal{L}|}) = \nu)$ as the "propagation of the PDs across the circuit" as it requires the PDs of circuit nodes to be calculated using the PDs of nodes in its transitive fan-in. The next section explains how this is performed.

3.3.2. Propagating Probability Distributions

The PDs of circuit nodes are dependent on each other. To take these into account, MVFs representing node functionalities need to be constructed in terms of the primary inputs and latch outputs. However, for many circuits encountered in practice, node MVFs can grow prohibitively large. To reduce complexity, we take only a limited amount of node

[1] One sufficient condition is that each next state variable be a non-trivial Boolean function of at least two present state variables. This condition is satisfied by the benchmark circuits in Section 4.

correlations into account by placing a limit on the sizes of the node MVFs. This places a hard limit on the computation described in Eqn. 1. We build node MVFs in terms of primary inputs in topological order. If during the computation of the MVF for node n, the node limit is exceeded, n is declared a "vertex", a new multi-valued variable v_n corresponding to n is created, and the MVFs of the nodes in n's fan-out are built in terms of v_n rather than the primary inputs in the transitive fan-in of n. The PDs of vertex nodes are assumed to be independent from each other. Vertex nodes divide the circuit into overlapping regions called *clusters*. Within each cluster propagation of PDs is performed exactly. Once the vertex nodes are chosen and node MVFs built in terms of them, propagation of the PDs across the circuit is performed by applying at each vertex node the algorithm described in Section 3.2.2. This technique is based on the "frontier" heuristic in [8] and enables a trade-off between accuracy and efficiency through proper selection of the node limit.

The PD propagation algorithm is used to iterate Eqn. 2 in order to determine its fixed-point. Given the steady state PDs of latches and the PDs for the primary inputs, the PD propagation algorithm is run one more time to calculate the steady state behavior of the remaining nodes, i.e. nodes internal to clusters. Using the steady-state statistics determined in this way, we compute tag observabilities as described in the next section.

3.4. Observability Computation

Recall that a tag τ associated with a node n represents the fact that during some clock cycle n was assigned a value of q while the correct (intended) value of n was p, where $p \neq q$. We denote this discrepancy ("event") by $n|p \triangleright q$. The event $n|p \triangleright q$ is said to be *observed* if assigning a value of q to n instead of p causes a change in the value of some observed variable. The probability that $n|p \triangleright q$ is observed, $O(n|p \triangleright q)$, is called the *observability* of this event . $O_y(n|p \triangleright q)$ denotes the probability that $n|p \triangleright q$ is observable via a variable y. The probability that $n|p \triangleright q$ is observed at y by causing the event $y|k \triangleright l$ is written as $O_{y|k \triangleright l}(n|p \triangleright q)$.

The computation of observabilities is performed in reverse topological order. Events at directly observed nodes have an observability of 1. To limit computational complexity, we use the same decomposition of the network into clusters as in Section 3.3.2. Note that a circuit node $n \in \mathcal{N}$ may lie in many clusters and an event $n|p \triangleright q$ can be observed through an event propagating through the vertices $y_1, ..., y_m$ of any one of these clusters. The conditions under which $n|p \triangleright q$ propagates through $y_1, ..., y_m$ are logically dependent. To avoid analyzing these dependencies we underestimate $O(n|p \triangleright q)$ as

$$O(n|p \triangleright q) = \max_{j \in \{1,2,\dots,m\}} O_{y_j}(n|p \triangleright q) \qquad (3)$$

Eqn. 3 represents a greedy strategy for determining the most likely path that $n|p \triangleright q$ can propagate through. In the following, we formulate the computation of $O_{y_j}(n|p \triangleright q)$ from the observability information of y_j, which, along with Eqn. 3, forms the subroutine we use for backward propagation of observabilities.

3.4.1. Observabilities of Cluster Vertices

Consider a cluster with inputs $x_1, x_2, \dots, x_n$ and output $y_j = f(x_1, ..., x_n)$ as shown in Figure 2. Suppose that we are given the observabilities $O(y_j|r \triangleright s)$ for all $r \neq s$. We would like to compute the observabilities of the cluster inputs $O_{y_j}(x_i|p \triangleright q)$ in terms of the observability of the cluster vertex, $O(y_j|r \triangleright s)$. We start by computing $O_{y_j|r \triangleright s}(x_i|p \triangleright q)$, i.e., the probability that an event $x_i|p \triangleright q$ is observed at y_j by causing an event $y_j|r \triangleright s$. We achieve this by determining which minterms in the space $x_1, ..., x_{i-1}, x_{i+1}, ..., x_n$ would give rise to y_j switching from r to s when x_i switches from p to q. The minterms that would cause $y_j = r$ when $x_i = p$ are given by $f_r(x_i = p)$. Thus $f_r(x_i = p) \wedge f_s(x_i = q)$ is the set of minterms that cause $y_j|r \triangleright s$ in response to $x_i|p \triangleright q$. We conclude

$$O_{y_j|r \triangleright s}(x_i|p \triangleright q) = P(f_r(x_i = p) \wedge f_s(x_i = q)) \cdot O(y_j|r \triangleright s)$$

Considering all possible r and s yields

$$O_{y_j}(x_i|p \triangleright q) = \sum_{r \neq s} O_{y_j|r \triangleright s}(x_i|p \triangleright q) \qquad (4)$$

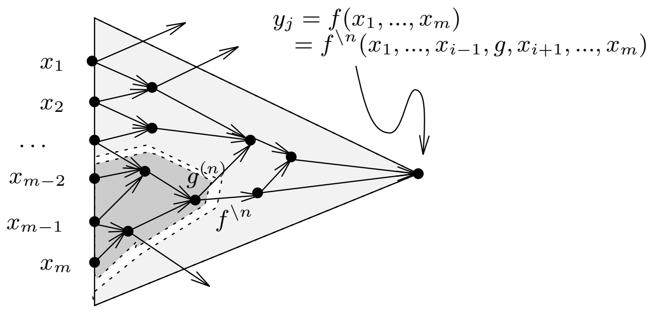

Figure 2. Cluster vertices and internal nodes.

3.4.2. Cluster Internal Nodes

Let n be an internal node of a cluster as in Fig. 2. Let $g^{(n)}$ be the MVF representing node n in terms of the cluster inputs. We define $f_{y_j}^{\setminus n}$ as the MVF representing y_j in terms of cluster inputs and node n. Given that node n has an intended value of p, the conditional probability that $n|p\triangleright q$ causes $y_j|r\triangleright s$ is

$$O_{y_j|r\triangleright s}(n|p\triangleright q) = \frac{P(f_r^{\setminus n}(n{=}p) \wedge f_s^{\setminus n}(n{=}q) \wedge g_p^{(n)})}{P(g_p^{(n)})}$$

Thus the probability that $x_i|p\triangleright q$ is observed at y_j is

$$O_{y_j}(n|p\triangleright q) = \sum_{r,s\in Y_j, r\neq s} O_{y_j|r\triangleright s}(n|p\triangleright q)\cdot O(y_j|r\triangleright s) \quad (5)$$

3.4.3. Observability Propagation

Some tags may take more than one clock cycle to propagate to an output. This is taken into account by performing several backward passes of observability propagation. After each pass, the observabilities at the latch inputs may change. If this is the case, observability propagation is repeated for all nodes in the transitive fan in of these latch inputs. Once the observabilities of all events at vertices are computed, the observabilities of events at internal nodes are determined using Eqn. 5.

The MDDs $O_{m|r\triangleright s}(v|p\triangleright q)$ are stored in a hash table using p, q, r and s as the key to avoid repeated computation. For each circuit node x_i and for each cluster vertex y_j in its fan-out, the MDD representing $O_{x_i|r\triangleright s}(y_j|p\triangleright q)$ needs to be computed for each $r, s \in Dom(x_i)$ and $p, q \in Dom(y_j)$. Therefore, observability computation has complexity $O(K \cdot |\mathcal{N}| \cdot \beta(\gamma)^4)$, where K is the length of the longest multi-cycle path considered, β is the maximum number of clusters that any circuit node lies in and $\gamma = max_{n\in\mathcal{N}}|Dom(n)|$. Since γ and β do not grow with the circuit size, observability computation is linear in the circuit size.

3.4.4. Detectability of a Tag

The detectability of tag $\tau = (n, +\Delta)$ is determined using its controllability and observability as

$$O(n, +\Delta) = \sum_{p,q\in Dom(n), p<q} P(n = p)\cdot O(n|p\triangleright q)$$

Detectabilities of $-\Delta$ tags are determined similarly.

3.5. Merit Functions and Optimization

A merit function is a heuristic means to quantify how good a given PD $\mathcal{R}$ is at detecting a given set of tags $\mathcal{T}$. The first category of merit functions we experimented with have the form $\psi(\mathcal{T}, \mathcal{R}) = \sum_{\tau\in\mathcal{T}} \psi(\tau, \mathcal{R})$, where ψ is a figure of merit related to the detection probability of τ. Choosing $\psi(\tau, \mathcal{R}) = O(\tau)$ gives the same weight to increasing the detectability of a tag from 0.8 to 0.9 as increasing it from 0 to 0.1. A tag with a high detectability such as 0.8 is very likely to be covered during a long simulation run. To bias the merit function so that it favors improving $O(\tau)$ for tags with very low detectabilities, we set $\psi(\tau, \mathcal{R}) = log(1 + O(\tau)/\varepsilon)$, where ε is an experimentally determined threshold.

The second category of merit functions is based on the intuition that it is desirable for every $\lambda \in Dom(l)$ of each latch $l \in \mathcal{L}$ to have a reasonable likelihood of being exercised, i.e. it should not be the case that at steady state $P(l = \lambda) << 1/|Dom(l)|$ for any λ. We have found that the combinational portions of the sequential benchmark circuits we considered were randomly testable when driven by uniformly random patterns. Therefore, if the latches of a circuit have PDs close to uniform distributions, most tags will have high detectabilities. To quantify this intuitive goal, we use a merit function that penalizes PDs for latches that deviate sharply from uniform distribution. Observe that computation of both categories of merit functions is linear in $|\mathcal{T}|$.

To maximize merit functions we use a simple local optimization algorithm: We start with an initial PD (usually a uniform distribution for all inputs), choose a primary input, perturb its PD some and determine if this results in an improvement in the merit function. If it does, we use the perturbed PD as the new starting point for further exploration. Otherwise we select another primary input. More elaborate optimization algorithms may improve results but are beyond the scope of this paper.

	FF	PI	PO	Tags	U	O	Cyc	Itn	Time	Mem
s1196	74	17	5	1122	1121	1121	200K	1	1	12
s1238	164	35	49	1080	1070	1070	200K	1	3.5	281
sbc	28	40	100	2158	1730	1864	100K	4	2*/4	53
s1423	74	17	5	1482	1112	1429	200K	4	7*/13	69
s5378	164	35	49	5043	4430	4943	48K	4	6*/12	100
8085	193	18	27	8276	2277	2864	150K	4	45*/75	53
s38584	1452	12	278	36100	24949	26090	900K	3	45*/104	205
dlx	62	25	6	676	489	496	100K	2	2.9	12
s13207	669	31	121	9450	6878	6890	450K	1	87*/93	270
s15850	597	14	87	13329	2070	2145	10K	1	28*/42	262

Table 1. Tag coverage results for sequential benchmarks.

4. Experimental Results

We implemented our algorithm for optimizing biases on the VIS[1] platform and integrated it with a custom-built RTL simulator and tag analysis tool. The input format to VIS as well as the simulator and coverage tool is BLIF-MV[2]. Circuits described in other formats are translated to BLIF-MV, and each tag corresponds to the output of a

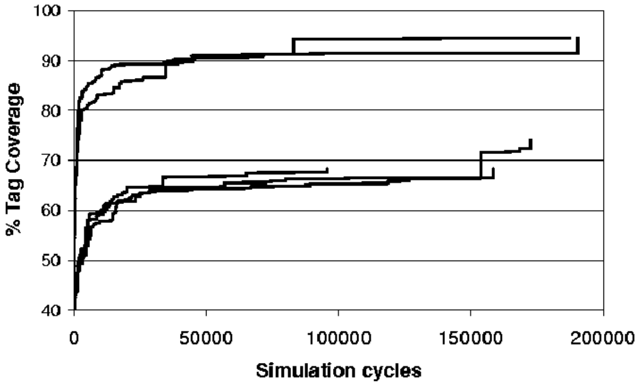

Figure 3. Tag coverage for s1423. The upper set of curves correspond to coverage obtained by optimized PDs. The lower set of curves are obtained using uniform PDs.

"table" or a latch in BLIF-MV. We tested our method on a set of sequential circuits from the ISCAS89 and VIS [1] benchmark suites and models of the DLX and Intel 8085 processors. On each circuit, we ran several very long (millions of cycles) simulations with uniform PDs to determine how much coverage can ultimately be attained in this way[2]. The results reported in Table 1 in column (U) are the best of several such long runs[3].

We compared these results with those obtained by our method. For the PD propagation and observability computation algorithms, we used an MDD node limit of 100 nodes. Experiments with each of our subroutines showed that estimates obtained with the 100 node limit were accurate within 2 % of those computed using 10,000 nodes. The convergence threshold for the iterations of Eqn 2 was set to 0.01. Further accuracy was not justified as the rest of the computations involve more coarse approximations. $\varepsilon = 5 \cdot 10^{-5}$ worked well as the threshold parameter for the first category of merit functions. On each circuit, we ran PD optimization to obtain a distribution $\mathcal{R}$. We ran several simulations using $\mathcal{R}$ until further runs did not improve coverage. At that point we re-applied the PD optimization algorithm to the set of remaining tags. We continued each simulation run until the coverage level reached saturation.

Although our framework is designed to handle multi-valued variables, we tested our method on binary benchmarks since, for these circuits, stuck-at fault coverage and tag coverage are equivalent and, for benchmarks from the testing literature, we were able to evaluate our coverage results taking into account the faults that are known to be impossible to cover[4]. The PD optimization algorithm performed 10 passes over primary inputs and, during each pass, the PD for each primary input was perturbed by 0.1. No primary inputs were grouped together into a multi-valued variable as we did not have high-level information about the circuits that would have guided this grouping[5]. The experimental results are presented in Table 1. The columns correspond to the benchmark, the number of latches, primary inputs and outputs, best known estimate for the number of tags that can be covered, tags covered by simulation with uniform PDs (U) and our iterative approach (O), the number of simulation cycles (Cyc) and loop iterations (Itn) for (O) , the CPU seconds per pass of merit function computation (Time) and the total memory consumption in MB (Mem). The run times are reported in this fashion to separate the effects of the optimization and estimation algorithms. The run times marked with a "*" correspond to latch PD based merit function computation. All experiments were run on a Compaq AlphaServer 8400 5/625 with 2GB of RAM running at 625 MHz. The experimental results fell into three categories:

(i) Simulation with uniform and optimized PDs both yield almost complete coverage.

(ii) Our approach achieves better tag coverage than uniform PDs in much fewer simulation cycles.

(iii) Uniform PDs give poor coverage and our approach does not improve coverage considerably.

Circuits in categories (i),(ii), and (iii) correspond to the first, second, and third groups of results in Table 1.

For circuits in category (ii), iterative simulations with optimized PDs gave higher coverage. Note that, while the coverage improvement is moderate, it did not require any manual effort and the same results could not simply be obtained by running longer uniform random simulations. Moreover, for any given number of simulation cycles, the coverage obtained by our approach was always better than that achieved by uniform PDs as demonstrated in the examples in Figures 3 and 4. It is also important to observe that many of these circuits are beyond the capacities of exhaustive state-space traversal methods.

Circuits in category (iii) are the circuits for which our method provided no significant improvement over simulation with uniform PDs, despite the fact that uniform PDs achieved relatively low coverage. For s13207, s15850, and DLX, we ran extensive experiments picking randomly selected input PDs and running long simulations with them. We found that even the union of all tags covered during

[2] We fixed "reset", "enable" and similar inputs to the values required for proper operation.

[3] The union of the sets of tags covered in these runs was only marginally larger than the set of tags covered by the best one.

[4] However, observe that much work on sequential circuit testing focuses on scan designs and is not directly comparable with our work. Since we are using tags as a proxy for errors in sequential behavior, we do not modify the structure of the circuit.

[5] Experiments run with randomly grouped primary inputs did not yield improved coverage but run-time and memory consumption was increased.

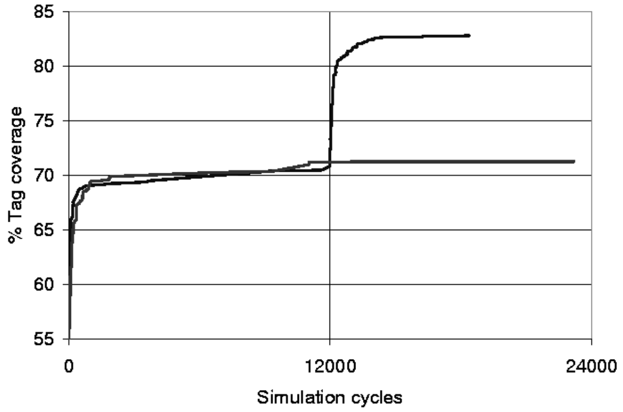

Figure 4. Tag coverage for s5378. The upper curve corresponds to simulation with optimized PDs. The sharp rise in coverage after 12 Kcycles is a result of the PD optimization algorithm being rerun. The lower curve is obtained by simulation with uniform PDs.

those runs is not bigger than the set of tags covered by uniform PDs. For these circuits, no selection of PDs seems to work better than uniform PDs. This finding suggests that, for some circuits, biased random simulation alone may not be able to control a simulation run through the state space as required. For instance, a closer examination of the simulation runs for s15850 showed that most latches had fixed values throughout the runs. Many sequential circuits have initialization sequences that are hundreds of clock cycles long and require that their inputs satisfy certain sequential and combinational constraints[6]. Such constraints were not available for the benchmarks in category (iii). It is difficult to ensure that input constraints are satisfied or particular input sequences are applied by only selecting input PDs. Thus, in general, biased random simulation seems more suited to exploring certain well-connected portions of the state space and needs to be complemented with more directed, deterministic methods. An example of such a combined approach is presented in [7]. Our method provides a way to improve the performance of biased random simulation as part of such a framework.

5. Conclusions and Future Work

We demonstrated an iterative, coverage-guided semi-formal validation method that improves the validation quality obtained by biased random simulation. The next step is to perform experiments on high-level, multi-valued circuits where we have a good understanding of the circuit structure and the input constraints. This will aid in eliminating the concerns described in the previous section and enable us to pinpoint the strengths and limitations of biased random simulation. A by-product of this work will be possible ways our work can be used to complement other vector generation methods.

Allowing input PDs to depend on state variables improves the control biased random inputs have on the simulation run. We intend to implement this capability. We also plan to investigate methods that handle the datapath and the datapath-control interaction. Since the datapath involves variables with much larger ranges, new, specialized methods are required to estimate and improve tag coverage. Our longer term research goals include the exploration of PD optimization algorithms for other coverage metrics.

[6]For instance, an input may be a delayed version of another, or it may be illegal to assert an acknowledgement before a request is sent.

Acknowledgment

We thank Aarti Gupta for the DLX example.

References

[1] VIS `http://www-cad.eecs.berkeley.edu/~vis`.

[2] Y. Kukimoto BLIF-MV `http://www-cad.eecs.berkeley.edu/~vis/doc/blifmv/blifmv.html`. 1996.

[3] M. Bozga and O. Maler. On the representation of probabilities over structured domains. In N. Halbwachs and D. Peled, editors, *CAV 99: Computer Aided Verification*, Lecture Notes in Computer Science 1633, pages 261–273. Springer-Verlag, 1999.

[4] S. Devadas, A. Ghosh, and K. Keutzer. An observability-based code coverage metric for functional simulation. In *Proceedings of the* 33^{rd} *Design Automation Conference*, pages 418–425, June 1996.

[5] F. Fallah, S. Devadas, and K. Keutzer. OCCOM: Efficient Computation of Observability-Based Code Coverage Metrics for Functional Simulation. In *Proceedings of the* 35^{th} *Design Automation Conference*, pages 152–157, June 1998.

[6] G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Markovian analysis of large finite state machines. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(12):1479–1493, 1996.

[7] P.-H. Ho, T. R. Shiple, K. Harer, J. H. Kukula, R. Damiano, V. Bertacco, J. Taylor, and J. Long. Smart simulation using collaborative formal and simulation engines. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 120–126, Nov. 2000.

[8] R. Ranjan, A. Aziz, B. Plessier, C. Pixley, R. K. Brayton, and F. Schneider. Efficient BDD Algorithms for FSM Synthesis and Verification. In *Proceedings of IEEE/ACM International Workshop on Logic Synthesis*, 1995.

[9] C.-Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. Despain, and B. Lin. Power estimation methods for sequential logic circuits. *IEEE Trans. on VLSI Systems*, 3(3):404–416, 1995.

[10] H.-J. Wunderlich. PROTEST: a tool for probabilistic testability analysis. In *Proceedings of the* 22^{nd} *Design Automation Conference*, pages 204–211, June 1985.

[11] H.-J. Wunderlich. Multiple distributions for biased random test patterns. In *Proc. of the Int'l. Test Conference*, pages 154–163, 1988.