Efficient Verification of Behavioral Models Using the Sequential Sampling Technique

Tom Chen[1]
Isabelle Munn[2]
Anneliese von Mayrhauser[2]
Amjad Hajjar[1]
[1] *Department of Electrical and Computer Engineering*
[2] *Department of Computer Science*
Colorado State University, Fort Collins, CO 80523

Abstract HDL model validation can involve billions of cycles of simulations. To improve validation efficiency we propose a stopping rule to determine when a validation phase using a specific type of patterns has reached a point of diminishing return.

Keywords: Behavioral Model Verification, Statistical Stopping Rules, VHDL.

1. INTRODUCTION

With rapid advances in VLSI technology, the complexity of VLSI chips has reached millions of gates per chip. Modeling of these complex chips often resulted in complex behavioral models with thousands of lines of HDL (Hardware Description Language) code. Verification of HDL models has, therefore, become a critical and a time consuming task. The conventional approach to verifying HDL models is through extensive functional simulations. Design engineers who wrote the HDL models are often responsible for generating test cases to verify the models. This approach works reasonably well for small to medium size HDL models. For large and complex HDL models, this approach is less effective because of the exponentially increasing input space that has to be explored during verification. Large and complex designs that employed the conventional approach often resulted in a huge amount of test patterns applied to the models. For example, in verifying the PowerPC-601 chip's behavioral

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35498-9_57
L. M. Silveira et al. (eds.), *VLSI: Systems on a Chip*

model, the design team at IBM generated more than a billion instruction-level test cases [13]. Yet often, at the end of the verification phase, the designers do not know the extent and quality of their verification efforts.

Applying verification coverage to HDL models is a relatively new concept that is rapidly gaining popularity in the VLSI design community. Using a set of coverage metrics to guide the verification process can help designers monitor simulation quality and, therefore, reduce time to market by providing a quantitative measure of simulation completeness. There are many commercial tools available to assist HDL code coverage measurement during simulation. The widely used coverage metrics include conventional coverages such as statement coverage and branch (decision) coverage, as well as state machine coverages related to state visitation and state transition. Although these tools provide instant feedback about the quality of a simulation, they do not guide designers to achieve the highest coverage in the shortest amount of time, i.e. design and verification productivity. For example, in the process of verifying UltraSPARC I, various coverage measures were taken including statement coverage, finitestate-machine trans ition coverage, and gate-level nodal toggle count coverage, and yet, 5 billion instruction simulation cycles with over 1 billion random patterns were ran before tape-out [14]. The question is whether 5 billion patterns are all necessary to achieve the final quality goal. The test suite used for verifying UltraSPARC I was composed of several different techniques including random patterns and functional patterns. Each of these different techniques are useful during a particular phase of the verification process. When verifying a complex design such as UltraSPARC I with billions of patterns, how do we determine a point where continuing with one test technique will likely result in diminishing return for code coverage, therefore, require a change in test technique? Furthermore, are 5 billion patterns enough for UltraSPARC I? Why not 10 billion patterns? How do we determine when to stop verification (simulation)?

This paper is our attempt to answer some of these questions by presenting a statistical stopping rule and its application to HDL model verification. Section 2 gives the background about the proposed stopping rule and its relationship to stopping rules used in software testing. Section 3 present our stopping rule using the sequential sampling approach. Section 4 summarizes the experimental results. Concluding remarks are given in Section 5.

2. BACKGROUND

Determining when to stop simulation with a given set of patterns (generated according to some strategy) can be done in a variety of ways. Most of the techniques come from software testing. Howden [3] uses a simple binomial distribution to determine probability of finding another coverage element and

an associated confidence interval. It is assumed that software test runs are independent of each other. Other approaches similar to [3] that are based on the binomial distribution include [9, 2].

Various statistical techniques have been used to determine the number of software test inputs needed to achieve a particular software test objective (in our case coverage related) [10]. The authors apply their statistical testing formula to a variety of white box software testing techniques. Stopping rules have also been used to attain a given reliability criterion [5, 12]. Poore et al. use statistical testing based on a usage model [6, 11]. This method models usage process and the software testing process as Markov Chains, assuming that the current state (usage as well as failure behavior) completely determines the next state. The model is more accurate than others, because it explicitly models states in the software. On the other hand it assumes memoryless behavior which may not always be the case, thus limiting the model's usability.

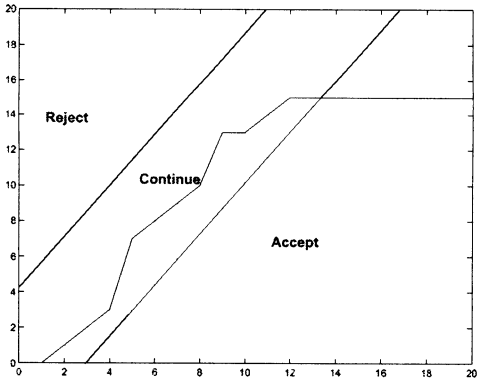
Through appropriate reinterpretation of failure events in the models as '*new coverage*' events, one could apply software reliability growth models [19]. However, one has to be careful to consider that while software failures may happen one at a time, coverage elements usually do not; e.g. one input pattern applied to a VHDL model may cause many branches to be covered. Thus, coverage shows a certain clumping effect [7]. This limits the applicability of some of the reliability growth models. An example of a stopping rule that considers clumping is the Compound Poisson Software Reliability Model by Sahinoglu et al., both the time-failure [7] and the clustered (interval) failure models [8]. Dalal et al. [1] also allow general distributions. However, these models are not as successful when the expectations of increase in coverage are low, as may be the case when the coverage is already rather high and further increases in coverage are difficult to achieve. This led us to consider two versions of a sequential sampling rule, because it does not make distributional assumptions nor assumptions about having new coverage when first using a new strategy for inputs (as for example when switching from random inputs with 7 clock cycles to random inputs with 4 clock cycles).

3. A SEQUENTIAL SAMPLING RATE

The sequential sampling approach to determining whether continued simulation of a HDL hardware design is likely to increase coverage or not is based on a related technique in software reliability analysis [4]. This technique, also called (software) reliability demonstration, determines whether the software failure intensity is met with high confidence or not. Key evaluation factors are the *discrimination ratio* and the *supplier and consumer risk*. The discrimination ratio, abbreviated γ, represents the maximum number of input patterns accepted that generate no new coverage. The supplier risk, α, represents false

positives. Whereas the consumer risk, β, represents false negatives. When assessing software reliability, the sequential sampling approach considers three decisions: accept, reject, and continue software testing. This results in three regions on the sampling chart: accept, continue software testing, and reject the software.

Figure 1 Original stopping rule plotted against software defect data



When using the same approach for determining whether continued simulation of a HDL model is likely to yield higher coverage, there are only two regions on the chart: continue, and stop. This requires a modified method. The discrimination ratio is now the ratio of the smallest acceptable branch coverage intensity versus the branch coverage intensity objective. This can also be expressed as the number of inputs one is willing to simulate that are not increasing coverage versus the number of uncovered branches that should still be covered with the given simulation strategy. The consumer and supplier risk have the same meanings. The equation below specifies the boundary between the stop and continue regions on our chart. The x-axis is the number of input patterns applied. The y-axis is the cumulative number of coverage items.

$$f(x) = \frac{B - x * ln(\gamma)}{1 - \gamma} \tag{1.1}$$

where x is the number of input patterns applied, $f(x)$ is the cumulative branch coverage after x inputs have been applied, γ is the discrimination ratio, and B is given by

$$B = ln\left(\frac{1-\beta}{\alpha}\right) \tag{1.2}$$

α is the supplier risk, β is the consumer risk.

The slope of the boundary line $f(x)$ is determined in large part by the discrimination ratio and to a lesser degree by α and β. The discrimination ratio

thus ought to be determined by the expectations of coverage yield. This may be fixed, no matter what strategy is used, or it may be variable. If it is variable, it should be determined both as a function of coverage increase expectations and branches left to cover. We present our sequential sampling technique both ways. When a variable discrimination ratio is used, we assume that a series of strategies are used to increase coverage and that the discrimination ratio is determined for each strategy as follows:

$$\gamma_n = \gamma_{n-1} * ln(\Delta_{n-1}) \quad n > 1 \tag{1.3}$$

where γ_{n-1} is the previously used discrimination ratio, γ_n is the new discrimination ratio to be applied to next data series, Δ_{n-1} is the number of new branches found in the previous technique.

The rationale for this continued growth in the discrimination ratio is based on the following observations: (1) It will get harder to increase coverage as cumulative coverage increases. Thus we must reduce our coverage expectations for successive strategies (the discrimination ratio must increase). (2) The rate of growth in the discrimination ratio is related to how many branches were found in the previous strategy. We chose a logarithmic relationship. Further, if the number of new branches found in strategy n is less than or equal to e, we multiply by a factor of 1. This forces a larger discrimination ratio. This makes the variable discrimination ratio more conservative later in the verification process when yield is likely to be lower.

Sequential sampling with variable discrimination ratio is derived from sequential sampling with a fixed discrimination ratio. Depending on the coverage data, it can behave similar to the fixed model. If, for instance, we have an initially slow rate of finding new branches, the discrimination ratio will not increase by very much, if at all. The more initial coverage generated, the more conservative the next discrimination ratio will be. The less coverage generated, the more closely this statistical sampling method will resemble the fixed sampling method.

4. COMPARATIVE STUDY

We experimented with a VHDL model that implements an image processing algorithm. The model contains 3785 lines of code and 591 branches. The system is organized with several systolic arrays as the processing elements and a global controller. The use of systolic array increases sequential depth, thus, making validation of the model more difficult.

When verifying VHDL models, hardware designers commonly start with a limited number of functional simulations that represent common or typical usages of the design's capabilities. This is then followed by large numbers of random inputs with varying clock cycles. The number of clock cycles accompanying each pattern is determined by sequential depth. Our goal of

simulating the model is to have most of the branches covered with a minimum amount of patterns.

We subjected the model to the following five verification strategies: (1) A functional simulation is applied first. It consists of 283 patterns. (2) 5000 random patterns are applied, each was held for 7 clock cycles. (3) 1000 random patterns are applied, each was held for 4 clock cycles. (4) 5000 random patterns are applied, each was held for 2 clock cycles. (5) 5000 random patterns are applied, each was held for 1 clock cycle.

Table 1 Test Coverage Results without Stopping

Test Case	# of *patterns*	*DCR Branch coverage%*	*covered*	*DR Branch coverage%*	*covered*
Functional	283	88.66	524	88.66	524
Random H=7cc	5000	93.57	553	96.45	570
Random H=4cc	1000	93.91	555	96.79	572
Random H=2cc	5000	95.26	563	96.79	572
Random H=1cc	5000	96.11	568	97.12	574

We generated random input patterns in two different ways: the first experiment only randomized data signals (marked DR in the results), while the second randomized both data and control signals (marked DCR in the results).

Table 1 shows the number of test patterns a pplied and the cumulative branch coverage at the end of applying each strategy for both the DR and DCR experiment.

Table 2 DCR/DR data using sequential sampling stopping rule $\alpha = .5$, $\beta = .01$, $\gamma = 2\ 50$

Testing Strategy	# of patterns	Cumulative coverage	New branches	Patterns saved	Branches missed
Functional	283	524	524	...	.
Random cc7	78/245	536/562	12/38	4922/4755	17/8
Random cc4	250/73	536/563	0/1	750/927	19/9
Random cc2	125/125	538/565	2/2	4875/4875	25/9
Random cc1	125/125	538/565	0/0	4875/4750	30
Total	861/976	538/565		15422/15307	30/9

The first experiment uses a fixed discrimination ratio $\gamma = 250$. This reflects the following yield expectations: we are willing to apply 5,000 patterns to increase coverage by 20 branches. Determination of the discrimination ratio depends on the amount of current coverage (thus, how much more difficult

it will be to increase coverage further) and the ability of a strategy (and the patterns it produces) to cover new branches.

Supplier risk is assumed to be medium ($\alpha = .5$; we are not particularly concerned about simulating a little longer than our coverage expectations provide for), while consumer risk must be extremely small ($\beta = .01$). This represents a very conservative approach to stopping.

The results in Table 2 show that for both the D C R and the DR case, the sequential stopping rule saves large amounts of simulation steps while missing few branches. The results of the DCR dataset show that simulating with only 5.3% of the input patterns only causes a loss of 5% in branch coverage. Similarly, the results of the DR dataset show simulating with only 6% of the original input patterns only loses 1.5% of the coverage. Thus, the sequential stopping rule reduces the amount of simulation required by about 95% without large sacrifices in coverage achieved. More importantly, it is able to indicate at which point the chosen strategies for simulation have become inefficient. In this case, simulating with random patterns quickly loses its effectiveness in covering new branches and should be stopped. To illustrate the stopping rule graphically, the Figures in 2 show visually the application of the stopping rule for the random-cc7 and random-cc2 strategies, respectively. The y-axis records the new branches (cf. column 5 in Table 2). The x-axis r epresents patterns applied multiplied by how many clock cycles they were applied. In both cases the stopping rule determined that saturation occured. The probability of new branch coverage is lower than what we established by setting values for γ, α and β.

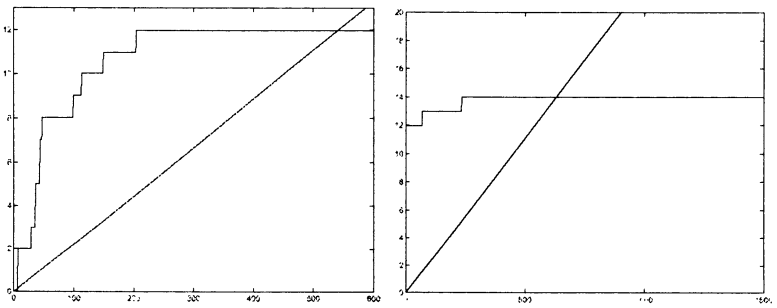
In the previous section we argued that a variable discrimination ratio might be helpful, indeed, that the rate at which new branches are found should determine the choice of the discrimination ratio. In the VHDL model, SYS7, the rate of finding new branches was very high in the beginning (during the functional simulation). The functional simulation alone covered 88.66 percent of all branches. The relatively high coverage and the low yield expectations for the random inputs lead us to expect that the probability of finding new branches will decrease rapidly with each new test phase.

To adjust for this decreased probability, we multiply γ (the current discrimination ratio) by a logarithmic factor of the number of branches found in the previous phase. Increasing γ forces a shallower slope. It means that we are willing to accept more inputs that generate no new coverage.

The results show that simulating with 24% of the original patterns loses 3.6% of the original coverage for the DCR dataset. For the DR dataset, simulating with 39% of the original patterns loses 1.7% of the original coverage. In this case, there is not a lot of benefit to a more conservative stopping rule, once again emphasizing the limited utility of simulating with random data as a means of increasing coverage.

Table 3 DCR/DR using variable sequential sampling stopping rule $\alpha = .5$, $\beta = .01$, $\gamma = var$.

γ	Testing strategy	# of patterns	Cumulative coverage	Patterns saved	Branches missed
100/100	Functional	283	524	–	–
626/626	Random cc7	195/541	538/563	4805/4459	15/7
1652/2293	Random cc4	305/932	539/563	695/68	16/9
1652/2293	Random cc2	826/2293	543/563	4174/2707	20/11
2290/2293	Random cc1	2290/2293	547/564	2710/2707	21/10
	Total	3899/6342	547/564	12384/9941	21/10



(a) (b)

Figure 2 The stopping rule for random-cc7 and random-cc2 strategies

5. CONCLUSION

This paper presented a stopping rule using the sequential sampling method. Unlike some of the previously proposed stopping rules used mainly for software testing, the advantage of the proposed stopping rule is that it allows the assumption that bug, or code coverage in our case, distribution is not well understood. Furthermore, it takes into account the possibility of very slowly increasing code coverage during the course of validation. Our results show that, using the proposed stopping rule, only 5.3% of the original patterns can cover 95% of the branches and only 6% of original patterns can cover 98.5% of the branches for DCR and DR methods, respectively. We would like to stress here that we are not suggesting that the entire validation process should be stopped altogether after the proposed stopping rule has been met. Rather, the proposed stopping rule is useful in guiding the validation process to determine the point when a different validation phase using a different strategy is warranted. Different validation strategies include applying patterns generated using different methods such as functional, random, and antirandom [15] pattern generation, so that the code coverage can be maximized using the minimum amount of

effort. Our future work includes applying the proposed stopping rule to other HDL models to better understand its effectiveness.

6. ACKNOWLEDGMENTS

This research was partially supported by a National Science Foundation Award MIP-9628770.

7. REFERENCES

[1] S. Dalal, C. Mallows, "When should one stop testing software", J. Am. Stat. Assn., v 83, 1988.

[2] D. Hamlet, J. Voas, "Faults on its sleeve: amplifying software reliability testing", Int'l Sym. Software Testing and Analysis, 1993, 89-98.

[3] W. Howden, "Systems testing and statistical test data coverage", COMPSAC 97, 1997, 500-505.

[4] J. Musa, *et. al.*, Software reliability: measurement, prediction, application, McGraw-Hill, 1987.

[5] B. Littlewood, D. Wright, "Some conservative stopping rules for the operational testing of safety-critical software", IEEE-TSE, v 23, n 11, 1997.

[6] J. Poore, *et. al.*, "Statistical testing based on a software usage model", Software Practice and Experience, 1995.

[7] P. Randolph, M. Sahinoglu, "A stopping rule for a compound poisson random variable", Applied Stochastic Models & Data Ana., v 11, 1995.

[8] M. Sahinoglu, U. Can, "Alternative parameter estimation methods for the compound poisson software reliability model with clustered failure data", Software Testing Ver. and Rel., v 7, n 1, 1997.

[9] A. van Schouwen, *et. al.*, "Evaluation of safety-critical software", Comm. of the ACM, v 33, n 6, 1990.

[10] P. Thevenod-Fosse, H. Weselynck, "An investigation of statistical software testing", J. Software Testing Ver., and Rel., v 1, n 2, 1991.

[11] J. Whittaker, M. Thomason, "A markov chain model for statistical software testing", IEEE Trans. Software Eng., v 20, n 10, 1994

[12] M. Yang, "Comparisons of stopping rules and reliability estimation in software testing", SERC-TR-58-F, Purdue Univ., 1992.

[13] P. Bose, "Architectural timing verification and test for super scalar processors", 24th Int. Symp. on Fault-Tolerant Computing, 1994, 256-265.

[14] J. Gateley, "Verifying a million gate processor", Integrated System Design 1997, 19-24.

[15] A. von Mayrhauser, *et. al.*, "Fast antirandom (FAR) test generation", 3rd IEEE High-Assurance Systems Eng. Sym., 1998, 13-14.