INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

NOTE TO USERS

This reproduction is the best copy available.

UMI®

DISSERTATION

BAYESIAN BASED STOPPING RULES FOR BEHAVIORAL VHDL VERIFICATION

Submitted by

Amjad Fuad A. Hajjar

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

for the degree of Doctorate of Philosophy

Colorado State University

Fort Collins, Colorado

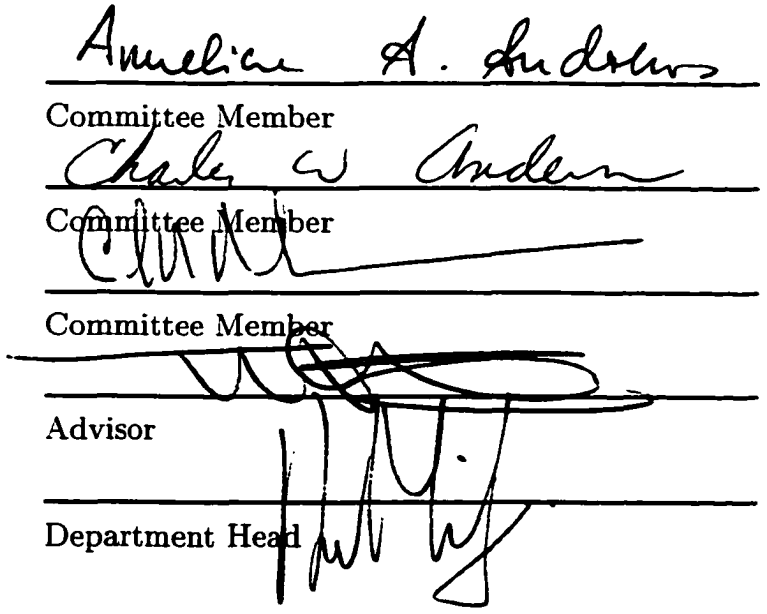
Fall 2001

COLORADO STATE UNIVERSITY

December 6, 2000

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY AMJAD FUAD A. HAJJAR ENTITLED BAYESIAN BASED STOPPING RULES FOR BEHAVIORAL VHDL VERIFICATION BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTORATE OF PHILOSOPHY.

Committee on Graduate Work



Committee Member

Committee Member

Committee Member

Advisor

Department Head

ABSTRACT OF DISSERTATION

BAYESIAN BASED STOPPING RULES FOR BEHAVIORAL VHDL VERIFICATION

Verification of complex behavioral models has become a critical and time-consuming process in hardware design. During behavioral model verification, it is important to determine the stopping point for the current test strategy and for moving to a different test strategy. It has been shown that the location of the stopping point is highly dependent on the statistical model one chooses to describe the coverage behavior during the verification process. This research presents Bayesian based statistical stopping rules for behavioral VHDL model verification. Unlike other existing approaches, the statistical assumptions of the proposed stopping rules are based on experimental evaluation of probability distribution functions and correlation functions extracted from simulation results of a suite of HDL models. The resulting assumptions are then employed with proper testing criteria in developing the proposed stopping rules.

Fourteen behavioral VHDL models were experimented with to determine the efficiency of the proposed stopping rules over the existing stopping rules. Two metrics for measuring efficiency have been developed to objectively compare between different stopping rules. Results show that the efficiency of using the proposed stopping rules are up to 3 times better than that of using the best existing stopping rule and up to two orders of magnitude better than that without using stopping rules.

We also propose a statistical forecasting model that predicts the future coverage during the verification process. The proposed model has high prediction accuracies in determining the probability of having coverage in the future as well as the expected waiting time to a new coverage item within the same prediction window. The estimated prediction error of having coverage was found to be 2% in the worst case when predicting the next 1000 simulation cycles. When extending the prediction window size to 10,000 simulation cycles, the maximum expected errors in predicting the interruption probability is 13%.

We believe that the proposed stopping rules and the statistical forecasting model will transform the existing brute-force approaches in behavioral model verification to more intelligent and statistical approaches to meet the tight time-to-market requirement of complex electronic systems without sacrificing the quality of the design.

Amjad Fuad A. Hajjar
Department of Electrical and Computer Engineering
Colorado State University
Fort Collins, Colorado 80523
Fall 2001

ACKNOWLEDGEMENTS

I would like to thank my advisor, *Dr. Tom Chen*, for not only his academic guidance but also his support provided all through my graduate study at Colorado State University. I would also like to thank the members of my committee, *Dr. Anneliese Andrews, Dr. Charles Anderson, and Dr. Carl Wilmsen.*

The *VLSI Lab* in the *Department of Electrical and Computer Engineering* has provided me a good facility to carry out the research. I am very grateful to everyone at the lab for all their help.

I sincerely thank my parents *Fuad Hajjar and Nawal Abuljoud* for their encouragement and support. Special thanks to my wife *Wafaa Kurdi.* I am highly indebted to her; without her care, patience, and understanding, this dissertation could not have been successfully completed.

DEDICATION

To my parents *Fuad Hajjar and Nawal Abuljoud,*

my wife *Wafaa Kudri,*

and

my daughters *Sawsan and Sana*

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Hardware design involves a series of mapping steps, i.e. synthesis steps, from one design representation to another. As shown in Figure 1.1, a typical chip design starts from its specification and the behavioral level representation and finishes at its layout representation and is ready to be manufactured. Upon completing each mapping step, more details are added to the design. With increasing complexity of VLSI chips, verifying that a design functions *exactly* according to its specification is becoming more and more difficult and consumes an ever greater portion of the overall design effort.

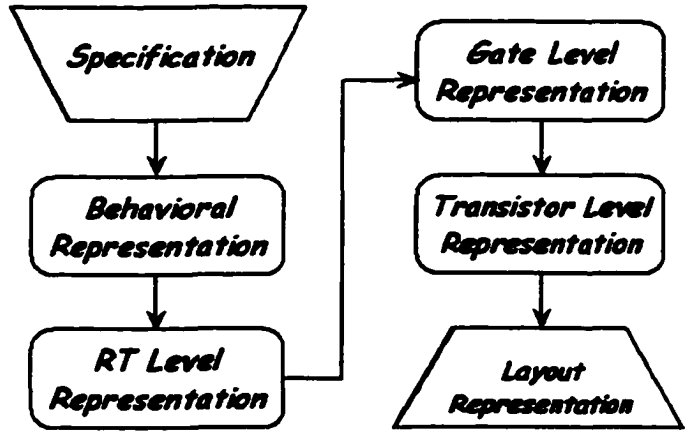


Figure 1.1: Phases of Hardware Design

Different algorithms and models have been developed to validate the mapping tasks. Formal verification techniques such as formal specification, symbolic model checking, and theorem proving have been developed to prove identical synthesis to

the specified behavior given that the behavioral model is correct by itself [6, 7, 8, 9, 10, 11, 12]. In general, these techniques put constraints and rules on the designs and mathematically prove equivalence to the synthesized (mapped) output.

In ensuring fault-free VLSI chips, fault models have been developed to check for manufacturing defects that can be introduced into the final product during fabrication. These fault models include stuck-at fault models, bridging fault models, and μ-operation models [13, 14, 15, 16, 17, 18, 19, 20, 21]. Test pattern generation techniques using these fault models include heuristic testing, mutation testing, D-algorisms, and artificial intelligence methods.

Most of the existing research in the area of hardware verification has concentrated on proving the equivalency between two design representations before and after a mapping step is applied. However, not enough research has been done to verify that a given design in its very first representation at the behavioral level is accurate against its specification. In the 80s, the idea of building testbenches was introduced with the development of HDLs [22]. A testbench is a model that compares two different designs and gives an error signal when the outputs from the two are not matched for the same input pattern. Designs in the behavioral level in VHDL, for example, were tested against their equivalent functional designs. Random tests are then generated and fed to the testbench for long periods of time to ensure that equivalence. The oracle in this case is the simple modeling of the function of the design where timing, delays, power consumption, and skewing are not the issue [22]. However, one might consider the designs described at the behavioral level as a piece of software program that performs certain functions. Thus, the problem of ensuring design correctness is becoming more like code correctness in the software engineering field. Quality measures of a given software can be evaluated in terms of code coverage.

Code coverage is analogous to fault coverage used in detecting manufacturing faults in the structural level modeled designs [24, 25, 26, 27]. As in software engineering, coverage results indicate what portions of the design are being tested; hence, the quality of the verification process is measured using some set of code coverage metrics [24, 27]. In HDLs, similarly, many different types of coverage criteria have been defined including block coverage, path coverage, expression coverage, state coverage, statement coverage, signal coverage, and branch coverage [24, 23, 25, 26]. A typical verification process in hardware design involves applying a large amount of test patterns to a design model described in a behavioral level language. Similar to that in software engineering, the quality of the testing[1] process can be measured using coverage metrics.

With a dramatic increase in design complexity, achieving higher coverage has become more and more difficult. Current methods of achieving desired quality use brute force approaches, where billions of test cases (patterns) were applied without knowing the effectiveness of the techniques used to generate these test cases (patterns) [27]. In verifying the PowerPC-601 chip's behavioral model [28, 29], the design team at IBM generated more than a billion instruction-level test cases to ensure a fault-free chip. Likewise, in verifying UltraSPARC I, 5 billion instruction simulation cycles were run before tape out [30]. The question to be asked is whether or not these billions of test patterns applied are all necessary in achieving the required quality of the final product. An even more relevant question to ask in light of ever tightening time-to-market pressure on hardware product design cycle is: At what point is a certain testing strategy less likely to result in a diminishing return for code coverage therefore

---

[1]In software engineering environment, testing means verifying that the software is correct. In hardware design, however, the testing is often referred to methods that deal with defects and errors inside the finished product. We use verification and testing interchangeably in this research to refer to the process of ensuring correct designs at the behavioral level

switching to another strategy? Our research is focused on developing solutions to answer these questions.

For a given product, the desired set of testing techniques is often known. Time spent on verifying a behavioral model using a given testing technique should be fruitful, i.e. time spent with diminishing return should be avoided. Therefore, techniques to determine the optimal stopping points for switching from one testing technique to another are needed. The problem of determining an optimal stopping point for a testing process is not new. Statistical models used to determine the proper stopping points for software testing [31, 32, 33, 34, 35, 36, 37] have been developed in the past. These models assume certain statistical behavior with regard to coverage based on the historical data collected in software engineering. As we will discuss later, the statistical behavior of coverage for hardware models at the behavioral level is different from that in software engineering. A new statistical method based on the correct statistical behavior of coverage for hardware behavioral models is needed.

This research proposes a statistical methodology to forecast the future yield of a certain testing strategy for a given behavioral model. Based on the history of testing, the proposed method expects near future coverage for a given testing strategy to be of a high level of accuracy and suggests accordingly the time to switch or stop testing. In Chapter 2, we discuss our strategy of choosing appropriate coverage metrics for this research. A discussion on many existing stopping rules used in software engineering is given in Chapter 3. Chapters 4 and 5 present the mathematical derivation of the proposed statistical method to determine the optimal stopping points. Experimental results based on a suite of VHDL benchmark circuits are presented in Chapter 7, and finally, concluding remarks are given in Chapter 8.

Chapter 2

TESTING CRITERIA

As mentioned earlier, the goal of testing or verifying a behavioral design is to check the effectiveness of the simulation test cases against faults in the behavioral code and to reduce the redundant portions of the design. To achieve this goal, quantitative measures are needed to help direct development efforts; these measures are called *metrics.*

"A popular and precisely defined metric used in the software world is *code coverage*" [27]. Later, the code coverage idea from the software field has been used in verifying hardware designs at the behavioral level to ensure correct designs at early stages of development [24, 26]. Good software tools that measure code coverage include many coverage metrics such as *statement, branch, condition, path,* and many other features [23].

The first and most obvious feature that comes to our minds when looking for a metric assisting in covering all portions of a design is statement coverage. Every assignment statement of a VHDL code should be executed. Otherwise, there might be a problem or a bug in that portion of the design, which then should be highlighted for the testers to debug.

Assignment statements in the behavioral modeled designs using VHDL are of three types: concurrent assignments, sequential assignments within process blocks, and conditional assignments. Figure 2.1 shows an example of a VHDL code including

the three different assignment statements in lines 9, 10, and (15, 17, or 19), respectively. By default, all concurrent assignments in the model are executed, leaving the question of whether the other two types are executed or not. One could consider conditional assignment statements as sequential assignments within dummy process blocks that are sensitive to their condition's signals. Thus, the question left is what statements in the process blocks, or dummy process blocks, are executed. One could consider the process blocks in VHDL as regular software codes where statements are executed sequentially, line by line. Thus, all statements within the process blocks will be executed unless they are controlled by branch conditions. That gives the *branch coverage* metric a greater importance over the statement coverage in VHDL. In fact, statement coverage in VHDL is a subset of branch coverage [26], where if all the branches in a VHDL design were covered, all the design statements would be covered too. For example, if the signals *ck* and *b* in the VHDL example of Figure 2.1 are set so that the 2 branches of conditional statement in line 10 and the 4 branches of the IF statement at line 14 are all visited, then all statements of the architecture will be covered. In addition, not all branches in the design code contain just statements—a branch could be empty, could contain other nested branch blocks, or could be missing, as in the dummy ELSE clauses. This means that the branch coverage metric gives more information to the tester about what portions of the design are executed [26].

Branch subcondition coverage, abbreviated as *condition coverage*, is another feature supporting behavioral design verification. It requires that all combinations of subconditions of all branch conditions be triggered. The motivation for this metric is that sometimes faults occur at one of the subconditions of a branch, forcing it to be always covered or always uncovered. The condition coverage is obviously more informative than the branch coverage; however, that requires more simulation time to report coverage. It has been estimated that different code-coverage-tool vendors have

```
1    ENTITY example IS
2    PORT ( a, b, ck: IN BIT;
3           s, z, x : OUT BIT
4         ) ;
5    END example;
6
7    ARCHITECTURE behavior OF example IS
8    BEGIN
9        s <= a XOR b XOR c;
10       z <= ( a AND b ) WHEN ( ck = '1') ELSE ( a OR b );
11
12       p: PROCESS (a, b)
13       BEGIN
14           IF ck'event AND ck = '1' THEN
15               x <= ( a AND b);
16           ELSIF ck = '0' AND b = '0' THEN
17               x <= '1';
18           ELSE
19               x <= '0';
20           ENDIF;
21       END p;
22   END behavior;
```

Figure 2.1: A Behavioral VHDL Example

different simulator-time overhead, ranging from 5% (the lowest possible for relatively clean code and for basic statement and branch coverage) to as high as 100% [23]. Overhead times for reporting coverage are definitely costly, especially when coverage is used in deciding when to stop the testing process.

A similar argument could be raised in regard to other coverage metrics in VHDL such as *path coverage*, where all possible combinations of paths through branches of different constructs are to be covered. Path coverage helps us know how many possible outcomes occurred during simulation. Thus, it requires more effective simulation of test cases in order to exercise all possible sequential paths through branches. Path coverage gives more information than branch coverage. However, since VHDL is a complex concurrent language, the overhead time needed to extract path coverage is much larger.

Two other coverage metrics that deal with signals have also been used for behavioral model verification. They are *toggling coverage* and *triggering coverage*. Toggling

coverage requires that all the signals' bits in the design have changed states or made transitions between states in both directions. Signals are assigned either in process blocks or in concurrent statements, which consequently affect the state transition of the model. Hence, a coverage that requires all possible transitions of all signals would help identify faulty portions of the model. The triggering coverage, however, requires that every signal in the sensitivity lists of all process blocks change states in order to activate their blocks (signals *a* and *b* in the sensitivity list of Figure 2.1 at line 12). This coverage aims to eliminate unnecessary signals in the sensitivity lists; it is also a subset of the toggling coverage, where all other signals have to be activated.

Other coverage metrics developed in the hardware field include *state coverage, arc coverage, expression coverage,* and *sequence coverage.* These coverage metrics focus on measuring the activities in the state machine of the given VHDL model. State coverage shows whether or not a valid state of the circuit has been reached. Arc coverage additionally requires transitions between the states in the state machine. The decisions of all state transitions are made through the expression on arcs. Thus, expression coverage shows whether these controlling expressions are tested or not. Finally, similar to path coverage, sequence coverage requires that all possible state sequences be covered. Figure 2.2 shows a finite state machine example, which consists of 4 states, 8 arcs, 6 expressions, and 256 different state sequences. Table 2.1 lists most of the commonly used coverage metrics in hardware testing.

Recent research has studied the effectiveness of two coverage criteria for behavioral VHDL models in the verification process [38]. The study used a set of evaluating properties developed in software engineering that objectively shows whether the given test criterion is reasonable, sound, and well-designed to assist verification or not. The seven properties of the Parrish and Zweben's reduced set [39] used in this study were shown to be consistent and independent. All the properties of the set hold for branch

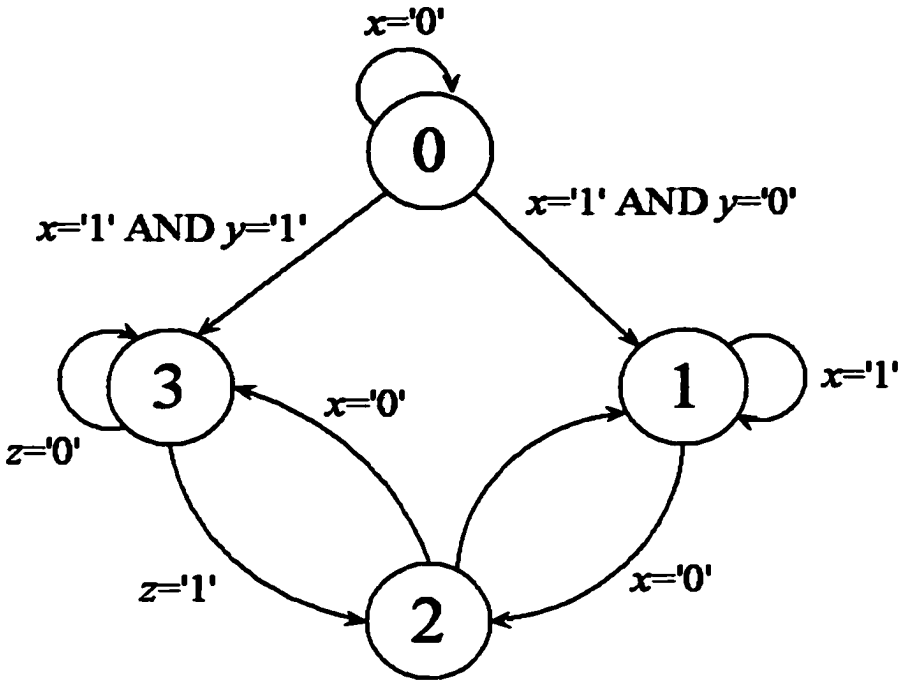


Figure 2.2: Finite State Machine Example

Metric	Description
Statement	All statements should be executed
Branch	All branches should be visited
Condition	All branch conditions should be exercised
Path	How many sequences through branches are executed
Toggle	All signals' bits should change states
Trigger	All signals in the processes' sensitivity lists are activated
State	All possible states in the state machine are visited
Arc	How many transitions have been made between states
Expression	State transition controlling expressions are tested
Sequence	How many sequences of states are executed

Table 2.1: Coverage Metrics in Hardware World

and bit toggling criteria except for the anticomposition property, which deals with testing branches in multiple units of the same behavior. An empirical evaluation of the criteria in having an error-free product was also provided in [38]. Out of 26 design errors reported in developing a 5-stage pipelined microprocessor, only one fault could

not have been found using branch and bit toggling coverage; 20 were guaranteed to be found through test stimuli that fulfill the criterion; the remaining 5 faults are detectable unless stimuli were selected with certain relationships between them.

In this research, therefore, we chose branch coverage as a metric to verify hardware designs described in the behavioral level with VHDL for the following reasons:

- Branch coverage is a superset of statement coverage, which examines all portions of the design.
- Branches in the given design are the key elements in the path coverage; thus more emphasis on branches is needed.
- The simulation-time overhead resulting from reporting branch coverage is minimal [23].
- It is shown that branch coverage criterion has a high potential value with respect to the software evaluation properties well known in the software area [38]. It is also proven that branch coverage assists practically in ensuring error-free designs at the behavioral level.

Once this choice is made, a statistical model is needed to allow testers to make the best decision about when to stop certain testing strategies that are not expected to yield a worthy branch coverage and switch to a different strategy or even stop the whole testing process.

Chapter 3

EXISTING MODELS FOR STOPPING RULES

Two areas contain information critical to developing mathematical models for forecasting coverage of a VHDL circuit: software testing and statistical forecasting. Researchers in the software area have developed many models to improve the estimation of software reliability. Similarly, statisticians have studied forecasting techniques in several applications such as weather, population, and accidents. This preview chapter of existing work done in both areas gives a clear background that helps develop the proposed model to forecast the future of branch coverage in testing a VHDL model.

3.1 Software Reliability Models

Almost all of the existing statistical models used to determine stopping points stem from software engineering research. During the past 30 years, many models have been proposed assessing the reliability measurements of software systems [31, 32]. These models help the designers evaluate, predict, and improve the quality of their systems. The meaning of the word "reliability" has also been discussed since the 80s. Some researchers prefer to use the binary concept of reliability for certain software programs; the program is either correct or faulty. Others, however, view the reliability of software as its probability of having faults that do not cause failures in future operations [31]. Reliability in this definition is the average correct operation of the software during the testing process. In other words, if a software program is being

tested by 100 test cases and 5 of these cases cause failures, then the reliability of this software will be 95%. If the testing is continued to 1000 test cases and still the 5 test cases are the only ones that cause the failures, then the reliability would be 99.5%.

Goel [32] classified some of the existing software reliability models according to their failure processes. *Times-Between-Failures* models [40] estimate software reliability by expecting the times between each two failures. *Failure-Count* models [40] estimate the remaining number of failures in the software based on priori stochastic processes. *Fault-Seeding* models [41], where known faults are seeded in the software program and the unknown faults are estimated during the test process, is another technique to estimate software reliability. *Input Domain Based Models* are unlike other types of software reliability models. The input test cases are drawn from an assumed distribution, while the reliability measurements start during the testing process. Most of the models of this type partition the input domain into sub-domains to avoid complexities. Each sub-domain usually corresponds to a path in the program.

Software reliability models [32] aim at estimating the remaining faults in a given software program. Hence, the direct use of such models in estimating the number of remaining uncovered branches in a behavioral model is not beneficial, because we know exactly how many branches are left. Instead, one could slightly modify the estimation process to focus on the expected number of faults, or coverage items in the case of behavioral model verification, within the next unit of testing time. Unfortunately, all the existing software reliability models assume that failures occur one at a time. Based on this assumption, expectations of the times between failures are carried on. In observing new coverage items in a behavioral model, branches are typically covered in clumps. Besides, the assumption made in the software reliability model that failures are independent of each other in the program makes such models ineffective and inaccurate.

3.2 Confidence Based Models (Howden's Models)

This approach takes advantage of hypothesis testing in determining the saturation of the software failures [33, 34]. A null hypothesis H_0 is performed and later examined experimentally based on an assumed probability distribution for the failures. We assume that H_0 is false and observe the outcome as a failure. Suppose that the outcome has a probability less than or equal to B; then we are at least $1 - B$ confident that H_0 is true. Similarly, if the failures for the next period of testing time have the same equal probability of at least B to occur, then for the next N test cycles, we have a confidence of at least C that no failures will happen, where

$$C = 1 - (1 - B)^N \tag{3.1}$$

If we experience a failure during the next N tests, then we continue testing and examine the hypothesis again. Otherwise, we stop testing at the end of the N tests.

To apply Howden's model to the process of HDL model verification, we first need to treat failures as interruptions, where an interruption is an incident where one or more new parts of the model are exercised. Using branch coverage as a test criterion, an interruption thus indicates that one or more new branches are covered. We set an upper probability value for the interruption rate B and choose an upper-bound level of confidence C. Experimentally, we do not examine the hypothesis unless the interruption rate becomes smaller than the preset value B. If the interruption rate becomes smaller than B, we calculate the number of test patterns needed to have at least C confidence that there will not be new branches in the next N test patterns and run them. If an interruption occurs, we continue examining the hypothesis until we prove it and then stop.

In this approach, we assume that coverage items, or indeed interruptions, are independent and have equal probabilities of being covered. One could use the fact

that the rate of interruption is decreasing and that we assume no interruptions will occur in the next N test cases; then the expected probability of interruptions will be:

$$B_t = \left(\frac{B}{t+T}\right) \tag{3.2}$$

where T is the last checked point in testing. This will then change the confidence equation as follows:

$$C = 1 - \prod_{t=1}^{N} \left(1 - \frac{B}{t+T}\right) \tag{3.3}$$

To implement Howden's formulae, the following steps can be taken:

1. B' is estimated by taking the cumulative sum of successes up to test case t and dividing by the number of test cases executed.

2. Repeat the calculation of B' on the next test case until it becomes less than the predetermined level B.

3. The number of test cases N needed to gain confidence C is calculated from either of the above formulae.

4. Execute the N test cases.

5. If no new coverage is gained during the execution of the test strategy, it indicates that the current verification strategy is no longer effective. The stopping point is $t + N$. If new coverage is gained at test case k, go back to step 1 and repeat the steps for $t = t + k$ times.

In Howden's model, the assumption that failures or interruptions have a given probability B independently is erroneous. Branches in an HDL model, as we know, are strongly dependent of one another. In fact, we can classify some branches to be dominant to other branches where it is impossible to cover the lower level ones without

covering their dominants. Moreover, the sizes of the interruptions are not modeled in this study, making the decision of continuing or stopping the testing process inaccurate. Lastly, this work doesn't incorporate the cost of testing or releasing the product, and the goal of testing in the first place is not only to have a high-quality product but also to minimize the testing costs.

3.3 Binary Markov Model

Sanping Chen and Shirley Mills, Statistics Department of Columbia University, developed a statistical Markov process model [35]. The probabilistic distribution assumptions of the model are the same as Howden's except that failures are statistically dependent with a certain unknown correlation constant ρ. Again, if interruptions are correlated, the probability of having no interruptions in the next N test cases is then given by:

$$p(0|N, B, \rho) = (1 - B)(1 - B + \rho B)^{N-1} \tag{3.4}$$

which makes the confidence as:

$$C \geq 1 - p(0|N, B, \rho) \tag{3.5}$$

If we set ρ to be zero, interruptions are independent; then we get the same model as Howden's. The implementation of this stopping rule is similar to that of Howden's except in the calculation of N for the confidence.

The basic assumption that interruptions have this simple probability distribution is not well understood or proven. Furthermore, the value of ρ in this model is unknown, and authors experimentally assumed different values ranging from 0 to 0.9 and obtained different results. Thus, this correlation needs to be determined theoretically or experimentally.

3.4 Testing Cost Based Models (Dalal-Mallows)

Dalal and Mallows [42] assumed a loss function associated with the testing and releasing of software programs. If, up to time t, there are $K(t)$ number of bugs in the model, then $aK(t)$ is the cost of fixing these bugs while testing, and $b(N - K(t))$ is the cost of fixing the remaining bugs in the field after releasing the product for some constant a and b. N is the expected number of total bugs in the program. A fixed increasing cost of the test setup and running of $f(t)$ also exists. Thus, the loss function is defined as:

$$L(t, N) = f(t) + aK(t) - b(N - K(t)) \tag{3.6}$$

Under these cost assumptions, the suggested stopping rule is to stop when the loss function is not decreasing. That reduces the loss function to a reward function defined as $cK(t) - f(t)$ $(c = b - a)$, because N is a fixed number, yet unknown. Testing stops when the expected reward function is no longer increasing. Now, $K(t)$ is a random process distributed as a nonhomogenous Poisson process with increments $\lambda\, g(t)$. That simplifies the stopping rule to the following:

$$\frac{f'(t)\; G(t)}{c\; g(t)} \geq K(t) \tag{3.7}$$

If we assume $g(t)$ to be exponential, and the cost function $f(t)$ is linear with time, then the decision to stop is when:

$$\frac{f}{\mu c}(e^{\mu t} - 1) \geq K(t) \tag{3.8}$$

where μ is maximum likelihood estimate of the history that $K(t)$ is Poisson with mean $\lambda(1 - e^{\mu t})$. μ is the solution of the following equation:

$$\frac{\mu\; (e^{\mu t} - 1)}{e^{\mu t} - 1 - \mu t} = \frac{K(t)}{S(t)} \tag{3.9}$$

$S(t)$ is the sum of all failures' life times up to test case t. This model incorporated the cost of testing and gave reasonable assumption to failures occurring in a certain program. However, applying this model to coverage items as failures suffers the independency problem of the Poisson process, where the times between failures are independent of the history of testing, although the parameters of the distributions are modified by time and cost constants. The model also implies the assumption of not having clumped failures, which reduces efficiency of the model when applying it to branch coverage estimation. Finally, this stopping rule diverged in some testing phases of some VHDL models. The reason is that the mathematical constructions used by this rule theoretically allow major changes in the internal constants values during the testing process, which ultimately make the rule diverge.

3.5 Compound Poisson Stopping Rule

This stopping rule was the first attempt to model the branch coverage process of VHDL circuits utilizing the benefits of the cost modeling of Dalal and Mallows [42] and solving the clumps phenomenon of branches (explained in the previous section) being covered in the testing process [43]. This model uses the empirical Bayesian principles for the compounded counting process. It was previously introduced as a software reliability model for failure estimation in 1992 [37] and later modified to incorporate the cost modeling proposed by Dalal and Mallows in 1995 [44, 45]. The model was recently formulated to model the branch coverage process in VHDL models [43].

The idea is to compound potentially two probability distributions, for both the number of interruptions and the size of interruptions. The resulting compound distribution is assumed to be the probability distribution function of the total number of failures, or coverage items, at a certain testing time point. The parameters of the

distributions are also assumed to be random variables calculated empirically, based on the well-known Bayesian estimation.

For modeling the branch coverage process for HDL models, it is assumed that the number of interruptions over the time, $N(t)$, is a Poisson process with mean λ, and the size of each given interruption, W_i, is distributed as a Logarithmic Series Distribution (LSD). The resulting compound distribution for the total number of failures, the sum of the sizes, is also known as a Negative Binomial distribution (NBD), if the Poisson parameter λ is set to $-k\ln(1-\theta)$.

The expected value of the total number of coverage in the next unit of testing time when testing is observed up to time t is estimated as:

$$E(X) = k\,\frac{\alpha + x}{\beta + k} \tag{3.10}$$

where x is the number of branches covered up to time t, α and β are the constants of the *priori* distribution of the parameter of the Logarithmic Series Distribution, and k is set so that the compound distribution becomes a Negative Binomial. So:

$$e^{\frac{\lambda}{k}} = 1 + \frac{\alpha + x}{\beta + k} \tag{3.11}$$

is a nonlinear equation that can be solved for k using the Newton-Raphson method with λ being the rate of interruptions up to time t.

This expected number of coverage in the next unit of testing time is then used in the cost model proposed in [42] to decide whether to continue or to stop testing with the current testing strategy. If the expected cost of testing for the next unit time is more than the expected cost of stopping, then the decision is to stop, and vise versa. This decision can be formulated as:

$$aE(X) < bE(X) + c \tag{3.12}$$

where a is the cost of one coverage item as yet uncovered, c is a fixed cost of one unit of testing time, and b is the variable cost of testing one uncovered branch. In other words, the decision to stop is when $E(X) < d$, for $d = \frac{c}{a-b}$, and checking this decision is to be made sequentially after applying each test pattern or case.

This stopping rule models the clumps of the coverage items statistically updating the assumed probability distribution parameters in every test case based on the testing history. However, since the interruption sizes are distributed as LSD, there should be at least one new coverage item per interruption covered. The problem involved was that after a short period of time, the coverage activity died for most of the test patterns applied sequentially, and for few patterns after that, one or more branches were covered. To overcome this problem, the outcomes of the simulation were packed into groups as if the time scale of the testing process were compact, and the packed branch coverage was applied to the stopping rule for the stopping decision. Nevertheless, the packing number of the input test patterns is still an issue. When using the same packing number of the Sys7 model in [43] as for the 8251 model in [48], the stopping rule failed to give meaningful stopping decisions. When using a much higher packing number for the 8251 model, the model seemed to work well.

3.6 The Sequential Sampling Models

All previously discussed stopping rules assume that failures or interruptions are random processes according to a given probability distribution. Another software reliability technique that doesn't involve any assumptions about probability distributions for the failure process was presented in [46]. Recently, the technique has been applied to VHDL models to determine stopping points for a given testing history of branch coverage [47]. The model evaluates the stopping decision based on three key factors: the discrimination ratio (γ), the supplier risk (α), and the consumer risk (β).

The discrimination ratio, γ, represents the maximum number of input test patterns accepted that have no yield in coverage. The supplier risk is the probability of falsely saying that the testing process should be stopped; the consumer risk is the probability of falsely saying that the testing process should continue. If the number of cumulative coverage at time t is $X(t)$, then the testing process should be stopped when:

$$X(t) \leq \frac{ln\left(\frac{1-\beta}{\alpha}\right) - t\ ln(\gamma)}{1-\gamma} \tag{3.13}$$

Figure 3.1 shows the decision boundary of the sequential sampling model applied to one of the behavioral models at some testing phases. The stopping decision is made when the boundary line intersects with the increasing coverage at a certain point of time. The stopping decision depends much more strongly on the value of γ than on α and β, and we can see clearly that this decision doesn't incorporate any cost model of the testing process. In [47], we modified the variable γ with respect to testing strategies so that if higher coverage were achieved in the previous testing strategy, the value of γ is increased in the current testing strategy in order to decrease the expectation of achieving more coverage in the current strategy. The new value of γ, therefore, becomes:

$$\gamma^{+} = \gamma\ ln(\Delta) \tag{3.14}$$

where Δ is the new coverage increase achieved in the previous testing strategy. The value of γ, however, remains the same if $\Delta \leq e$.

This type of statistical modeling doesn't use any priori probability distribution for the data provided. This is one reason sequential sampling models are widely used in many testing areas. However, the cost of testing is not modeled in making the stopping decision. Moreover, the stopping point determined by the sequential sampling model is very sensitive to the γ value chosen during the testing process.

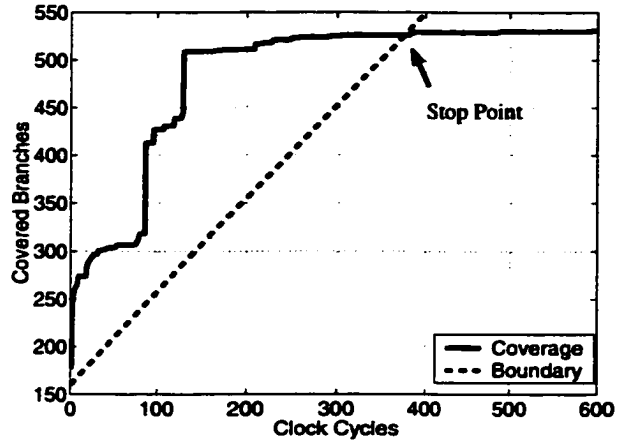


Figure 3.1: An Example of the Sequential Sampling Decision Line

3.7 Summary

From the above review of the existing models for determining stopping points, several important observations can be made:

- The underlining assumptions about the statistical behavior of coverage play an important role to determine stopping points.
- Different models assume different statistical behavior of coverage.
- It is fundamentally wrong in all the existing models to assume that the probability of having new interruptions (coverage) at a given time point during the verification process is independent of the history of the verification process.

Table 3.1 lists the problems involved when applying each of the existing stopping rule to the VHDL verification field.

A new statistical stopping rule is therefore needed to overcome shortcomings of the existing models. Solving the problem of independent interruptions is the most important characteristic that the new model needs. Secondly, the size of each interruption in the prediction process should accurately be modeled in order to have high

Software Reliability Models	. one failure at a time . failures are independent . deciding on the levels of reliability
Confidence-Based Models **HW1, HW2**	. mapping interruptions to failures without their sizes . independent interruptions . simple Binomial distribution assumption . cost of testing is not modeled
Binary Markov Model **BM**	. same problems as in HW1 and HW2 . correlation constant ρ is unknown . ρ is fixed during the process
Testing Cost Based Model **DL**	. independent failures . experimentally diverges for some phases . clumps of branch coverage are not modeled
Compound Poisson Model **CP**	. sensitivity to the packing number . interruptions are independent
Sequential Sampling Models **SS1, SS2**	. linear method in the stopping decision . cost of testing is not modeled . sensitivity to the choice of γ

Table 3.1: Problems of the Existing Models

prediction accuracy. In Chapter 4, we explain in detail the experiments conducted in order to have a better understanding of branch coverage behavior of behavioral VHDL models during the verification process and to have the best assumptions needed in performing the proposed model.

Chapter 4

STATISTICAL BEHAVIOR OF VHDL MODELS: THEORY AND EXPERIMENTS

If the verification strategy is not expected to show potential, the process should be stopped or switched to a different strategy. Thus, the outcome we are observing is branch coverage for every clock cycle of testing time, and what we want to estimate first is the expected number of branches to be covered at time t, i.e., $E\{X_t\}$, where, X_t is the branch coverage random process.

As in [43, 48, 49], we can decompose the branch coverage process X_t into two random variables: interruptions N_t, where one or more new branches are covered at time t, and sizes of the interruptions W_t, given that we have an interruption at this time. Unlike the Compound Poisson model [48], we believe that interruptions are strongly dependent on each other, governed by some correlation function, and the assumption that N_t is a Poisson Process violates this dependency relationship. The sizes of the interruptions, however, might not be that strongly dependent especially after a long period of testing where achieving new coverage will be more and more difficult, and hence, statistically, W_t, representing the sizes of interruptions, do not exist. Thus, the random variables, W_ts, conditional on interruption occurrences, N_t, can be assumed statistically independent.

To assume correct probability distributions for the sizes of interruptions, we conducted PMF extraction experiments (also known as *distribution harvesting*) to esti-

mate the best fitted distribution function to the histograms of the actual interruption sizes, W_t, at every discrete time t.

For the number of interruptions, N_t, we estimated the probability of having an interruption during the testing process as $p(t)$ for every discrete time t. Moreover, this $p(t)$ function is further decomposed into a shape function and an amplitude value:

$$p(t) = \zeta_t \, f(t) \tag{4.1}$$

where the ζ_t values can be determined statically or dynamically based on the history of testing the behavioral model. The shape function, however, should be statically chosen so that it best describes the power of increments of the interruptions N_t. For that purpose, we also conducted a *shape fitting* experiment to find out the shape function that models the interruption increments.

4.1 PMF Extraction of Interruption Size W_t

The choice of a *probability mass function* for the size of branch coverage clumps, given that there will be an interruption, could either be assumed or can be fitted experimentally into a set of known distribution functions. We focused on choosing one of three distinguished, widely-used probability mass functions in the field of statistics: *Poisson, Geometric, and Logarithmic.* Fortunately, all of these three distributions have one parameter that makes the mathematical derivations analytically easier and feasible. One could question why we did not use other distributions of multiple parameters and rewardingly get better modeling. The question is valid; however, the goal of developing this model is to understand and direct an ongoing testing process to stop or switch testing, and that decision ought to be produced and calculated fast (even faster than the testing process itself). Thus having a statistical model that utilizes numerical calculations may not be beneficial.

Fitting a distribution to a given data histogram is based on a statistical method for estimating the error between the fitted function and the actual data. Typically, the *Least Square Method* is the one most frequently used. The squared difference between the data and the fitted model is calculated as:

$$\sum_{i=1}^{N} (f(i;\gamma) - x_i)^2 \tag{4.2}$$

where $f(t;\gamma)$ is the chosen fitting function, x_i is the i^{th} value of the data, and N is the number of data. γ is the parameter of the fitting function $f(t)$, and the best estimated value for γ is the value so that the above error quantity is minimum.

Another known method suitable for fitting distributions to data is the *Maximum Likelihood* estimation method. The rationale of this method is that it is robust, consistent, and straightforward. For a given distribution, the Maximum Likelihood Estimate (MLE) for its parameter is the minimum γ value of:

$$\log\left(\prod_{i=1}^{N} f(x_i;\gamma)\right) \tag{4.3}$$

Table 4.1 lists the MLE of the parameters of the three chosen distributions.

Distribution	Param	Estimation
Poisson	λ	$\frac{\sum x_i}{N}$
Geometric	p	$\frac{N}{N+\sum x_i}$
Logarithmic	θ	$\theta = (1 + \frac{\sum x_i}{N})(1-\theta)\ln(1-\theta)$

Table 4.1: Maximum Likelihood Estimates

Finally, the data to be fitted out of the behavioral models has to be prepared. Statistically speaking, one should run infinitely many experiments; each run is seeded differently and applied for each behavioral model. The resulting coverage as interruptions and sizes are then to be averaged out every time unit, i.e. clock cycle. A large

number of runs would, of course, be satisfactory. This method is known in statistics as *many short runs distribution harvesting.* We chose to run 100 differently seeded random test patterns for a length of 50,000 clock cycles for 13 VHDL models and collect the results. At each clock cycle, we observed the branch coverage increment. If an interruption occurred, we recorded its time and size. We conducted this procedure for all the behavioral models for each 100 differently seeded runs. Thus, we collected 65 billion data points ($100 \times 13 \times 50,000$).

After the collection of the actual data, we constructed the histogram at each time slot (clock cycle) that had at least one new branch covered. We then eliminated the points of the runs that had no interruptions from the averaging process and built the *histogram* of W_t. These histograms became our data that would be fitted to a probability mass function.

Out of the 13 VHDL models and the 100 different seeded test pattern runs, we found 57,266 out of 650,000 possible time slots that had interruptions. Figure 4.1 shows an example of one of the histograms and its fitted *pmf's* for Poisson, Geometric, and Logarithmic distributions.

We fitted all the active time slot histograms to the three chosen *pmf's* using the two methods of fitting: LSQ and MLE. Table 4.2 shows the averaged errors of fitting all the VHDL benchmarks for each chosen distribution and using both fit evaluation methods.

From Table 4.2 we determined that the best fit for the size of branch coverage increments W_t using the two methods of fitting evaluation is the Poisson process for all the VHDL benchmarks. Thus, we let W_t be a random process distributed as a Poisson Process with parameter β_t:

$$W_t \sim e^{-\beta_t} \frac{\beta_t^{w-1}}{(w-1)!} \tag{4.4}$$

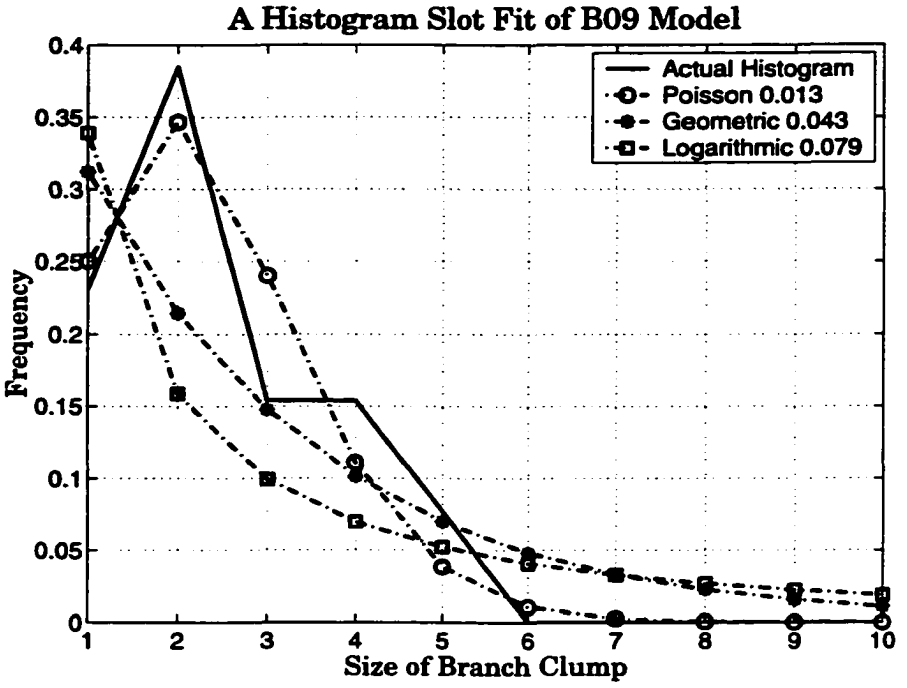


Figure 4.1: A Histogram Fitting Example of W_t

VHDL	Poisson		Geometric		Logarithmic	
Model	LSQ	MLE	LSQ	MLE	LSQ	MLE
8251	0.1746	0.1788	0.2310	0.2499	0.2548	0.2896
B01	0.3077	0.3161	0.3875	0.4133	0.4211	0.4902
B04	0.1316	0.1392	0.1802	0.2004	0.2032	0.2381
B05	0.1894	0.2000	0.2422	0.2610	0.2669	0.3064
B06	0.2250	0.2345	0.2968	0.3225	0.3284	0.3743
B07	0.2224	0.2316	0.2919	0.3165	0.3224	0.3667
B08	0.1889	0.1995	0.2454	0.2659	0.2704	0.3114
B09	0.1873	0.1977	0.2424	0.2630	0.2685	0.3110
B10	0.2129	0.2212	0.2763	0.2984	0.3042	0.3451
B11	0.1313	0.1389	0.1798	0.2001	0.2028	0.2378
B12	0.1890	0.1997	0.2437	0.2636	0.2692	0.3097
B14	0.1870	0.2001	0.2449	0.2686	0.2726	0.3164
B15	0.1669	0.1759	0.2176	0.2360	0.2410	0.2779
Ave.	0.1934	0.2026	0.2523	0.2738	0.2789	0.3211

Table 4.2: Distribution Fitting Errors of W_t

where β_t is a random variable representing the parameter of Poisson distribution that should be estimated from the history of the simulation. Fortunately, the mathematical derivation using Poisson *pmf* is simpler and produces nicely closed analytical forms.

4.2 Fitting the Interruption Correlation Function $p(t)$ Using RSM

Since the number of branches in a given VHDL design is a known constant, coverage rate during the verification process is increasing (or non-decreasing) until all the branches are covered and the coverage rate becomes steady at 100%. Considering interruption rate, we divided the $p(t)$ function illustrated previously in Equation (4.1) into two quantities: ζ_t being the amplitude value, and $f(t)$ being the general shape of the interruption decreasing rate. $f(t)$ is an increasing function of time that should experimentally be considered to generally apply for all VHDL behavioral models. ζ_t value should adjust the overall interruption rate, $p(t)$, specific to the model under test. Consider the hypothetical example of Figure 4.2 where the bars represent increasing amount of cumulative interruptions. The envelope line superimposed on the bars is the shape describing the increase of the activity.

To find the best shape function that fits the cumulative number of interruptions over time, we ran 1,000,000 different random test patterns for each VHDL benchmark and observed the resulting cumulative number of interruptions, $n(t)$. We chose one million patterns in these experiments so as to make sure the correlation of the interruptions is modeled even after a long period of simulation time, where the coverage activity becomes more important. Once the million $n(t)$ data points are collected for each of the 13 benchmarks, we should take into account the following circumstances:

- The chosen fitting function should fit all the benchmarks with the same shape, i.e. one class of functions with different constants for the different benchmarks

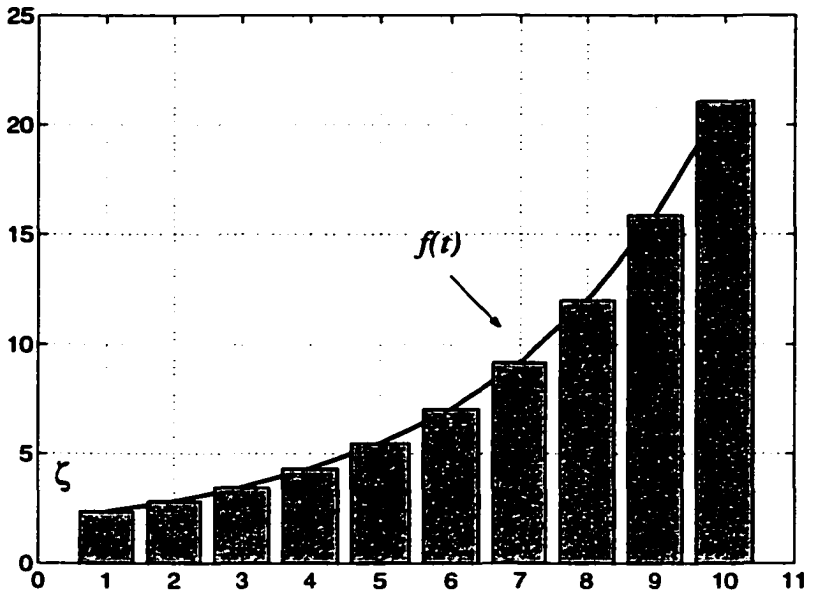


Figure 4.2: A Hypothetical Example of $n(t)$ Activity

- Exact fitting to $n(t)$ in terms of constants is not required, since the fitting function will be differentiated to the required shape function $f(t)$ and further multiplied by an adjusting amplitude value, ζ_t . In other words, if the fitting process resulted in $g(t)$ being the best fit, $f(t)$ is:

$$f(t) \propto \frac{\partial}{\partial t} g(t) \tag{4.5}$$

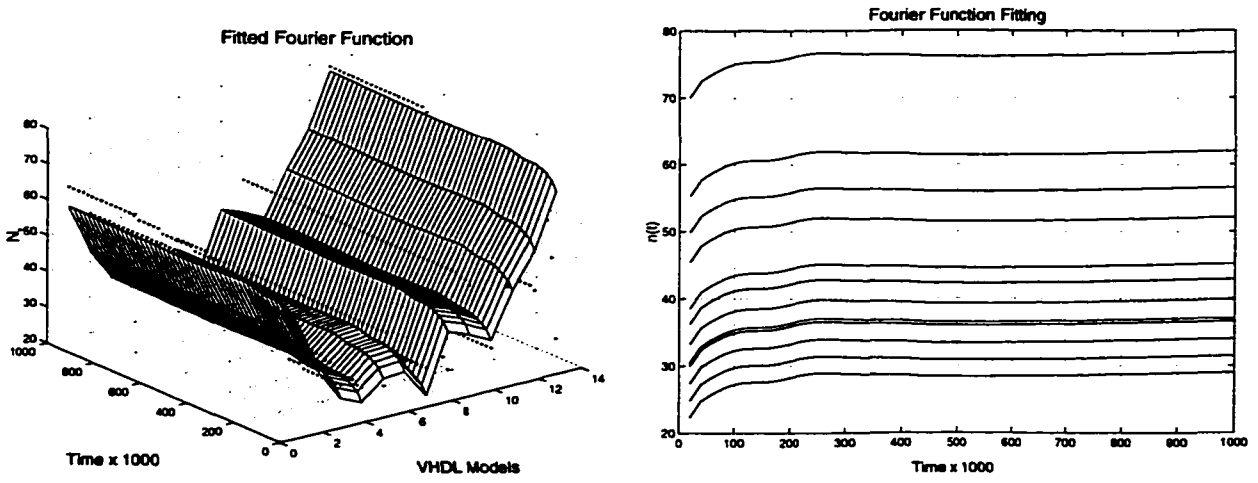
- Since $n(t)$ is the cumulative of non-negative numbers of interruptions, the fitting function should be non-decreasing in nature, meaning that if the derivative of the resulted fitting function can be negative, the resulted fitting function is rejected

Based on the above requirements, we use Response Surface Method (RSM) [128] to derive the shape function. To construct the 3D data set and run it in a reasonable time, we sampled 1000 points out of the one million $n(t)$ for each of the 13 models. Thus, the data set to be fed to the RSM process had 13 points in the x-axis and 1000

points in the y-axis, which made the length of the data set 13,000 lines, each line had three points (x, y, z). The results from the RSM process are as follows:

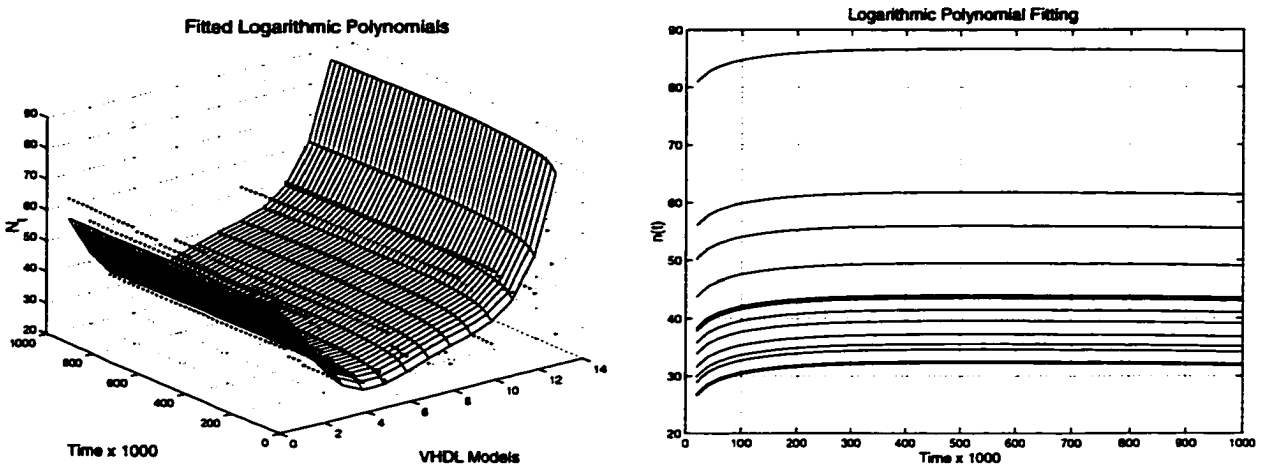
The best shape function was a *Fourier* Series Simple Order 2×10 function as in Figure 4.3. Unfortunately, we rejected this function because the Fourier function is a series of trigonometric function, *sin* and *cos*, that incorporate the x values with the time variable making the fitting function for each benchmark literally different. Besides, the computation complexity of the Fourier function is high making the use of it in predicting coverage expensive. Finally, the nature of trigonometric functions can accept negative values, in general, which prevent its use to describe the interruption activity. The best shape functions appropriate for our applications were a group of polynomial functions shown in Figure 4.4 and illustrated as:

$$\begin{aligned} n(t) &= a + bx + cx^2 + dx^{2.5} + ex^3 + fe^{\frac{x}{w(x)}} \\ &+ g\sqrt{x}\ln x + h\sqrt{t}\ln t + i\ln t \end{aligned} \tag{4.6}$$



(a) (b)

Figure 4.3: Fourier Fitted Function for $n(t)$



(a) (b)

Figure 4.4: Logarithmic Polynomial Fitted Function for $n(t)$

The shape derived from different VHDL models is consistent, and we approximate the generic shape function, $n(t)$, as:

$$n(t) = h\sqrt{t}\ln t + i\ln t \tag{4.7}$$

where $h = -0.03$ and $i = 3.03$ with i being almost 100 times bigger than h. It shows that the stronger impact on forming the shape of $n(t)$ is due to the logarithmic term of time t. Thus, the shape function of $n(t)$ varies logarithmically.

The mathematical shape of $p(t)$, which is the probability that there will be an interruption at time t, is the discrete derivative of the logarithmic function $\ln(t)$:

$$p(t) = \zeta_t \times \ln\frac{t}{t-1} \tag{4.8}$$

where ζ_t is the estimated amplitude of the number of interruptions at time t.

Chapter 5

THE STATIC AND DYNAMIC BAYESIAN MODELS

Using the distribution and the correlation for branch coverage process, X_t, presented in Chapter 4, we derive the statistical model for the branch coverage increase for a given history of verification. First, we define the conditional distribution functions of the interruption occurrences, D_t, and the sizes of interruptions, W_t. We then start the Bayesian analysis by calculating the likelihood function of the Bayesian parameter, namely β_t, and expect the coverage at time $t > T$ given the verification history. Finally, we expect the total number of branches to be covered at any future time $t > T$ given the verification history up to time T. Having the closed form equation for the expected coverage, we impose different stopping criteria to develop the proposed stopping rules. We finally derive the forecasting error equations of the proposed coverage behavior model to evaluate its efficiency in predicting coverage.

5.1 Mathematical Derivations

Let x_t, $t = 1, 2, 3, \ldots, T$, be the cumulative number of actual branches covered from the beginning of the simulation up to time t. Let n_T be the number of interruptions up to time T. Let W_t be the random variables of the sizes of the interruptions $t = 1, 2, \ldots, n_T$. We have:

$$X_T = \sum_{t=1}^{n_T} W_t \tag{5.1}$$

Based on Equation 4.4, the conditional distributions for the sizes of interruptions, W_ts, given that there will be interruptions at time t, and the occurrences of interruptions,

D_t, have the following distributions:

$$W_t|\{D_t, \beta_t\} \sim \frac{e^{-\beta_t}\ \beta_t^{w_t-1}}{(w_t-1)!}\ d_t \tag{5.2}$$

$$D_t \sim p(t)\ d(t) + q(t)\ (1-d(t)) \tag{5.3}$$

In order to start the Bayesian analysis, the likelihood function of the parameter β should first be estimated from the data of the verification history, $\vec{w}$ and $\vec{d}$. Given the distribution functions of W and D in Equation (5.3), the likelihood function of β_t given the history data is:

$$L(\beta_t|\vec{w}, \vec{d}) = \prod_{j=1}^{t} f_X(w_j, d_j|\beta_j) \tag{5.4}$$

The product terms of the likelihood function (5.4) can be decomposed into two sets: the terms where there are interruptions, and the terms where there is no interruption:

$$\prod_{j=1}^{t} f_X(w_j, d_j|\beta_j) = \prod_{j:d_j=1} p_j\ \frac{e^{-\beta_j}\beta_j^{w_j-1}}{(w_j-1)!} \times \prod_{j:d_j=0} q_j \tag{5.5}$$

Expanding the product of the two terms and separating the constants from the variable quantities of w would make the likelihood function as:

$$L(\beta_t|\vec{w}, \vec{d}) = \frac{\prod_{j:d_j=1} p_j \prod_{j:d_j=0} q_j\ e^{-\left(\sum_{j:d_j=1} \beta_j\right)} \prod_{j:d_j=1} \beta_j^{w_j-1}}{\prod_{j:d_j=1} (w_j-1)!} \tag{5.6}$$

Now, the product of p_js and q_js are constants with respect to the likelihood function of w. Thus

$$L(\beta_t|\vec{w}, \vec{d}) = K_1\ K_2\ e^{-\left(\sum_{j:d_j=1} \beta_j\right)} \times \frac{\prod_{j:d_j=1} \beta_j^{w_j-1}}{\prod_{j:d_j=1} (w_j-1)!} \tag{5.7}$$

Let $\beta_t = \beta\ g(t)$ for some constant β and a decreasing function $g(t)$. Let $G(t) = \sum_{j:d_j=1} g(j)$. The likelihood of β_t from Equation (5.6) is proportional to:

$$L(\beta_t|\vec{w},\vec{d}) \propto e^{-\left(\sum_{j:d_j=1} \beta\ g(j)\right)} \times \frac{\prod_{j:d_j=1} (\beta\ g(j))^{w_j-1}}{\prod_{j:d_j=1} (w_j-1)!} \tag{5.8}$$

The quantity $(w_j - 1)!$ is a constant with respect to β_j. Also, the product terms $g(j)^{w_j-1}$ can be separated from β and become a constant with respect to β. Substituting $G(t)$ in Equation (5.8), the likelihood is proportional to:

$$L(\beta_t|\vec{w},\vec{d}) \propto e^{-\beta\ G(t)} \times \beta^{\left(\sum_{j:d_j=1} w_j - 1\right)} \tag{5.9}$$

Utilizing Equation (5.1), the summation of w_js over j where there are interruptions up to time t is x_t. The summation of 1's over the same range of j is nothing but the total number of interruptions up to time t, n_t. Thus,

$$L(\beta_t|\vec{w},\vec{d}) = K_0\ e^{-\beta\ G(t)} \times \beta^{x_t - n_t} \tag{5.10}$$

for some constant K_0.

From the likelihood of Equation (5.10), we then estimate the constant β, that is assumed to be distributed as $\Gamma(r,\gamma)$, using the Bayesian method [50]:

$$\hat{\beta} = \frac{\int_0^\infty \beta\ L(\beta|\vec{w},\vec{d})\ \Gamma(\beta;\gamma,r)\ d\beta}{\int_0^\infty L(\beta|\vec{w},\vec{d})\ \Gamma(\beta;\gamma,r)\ d\beta} \tag{5.11}$$

where the Gamma function of β is given by:

$$\Gamma(\beta;\gamma,r) = \frac{\gamma^r}{\Gamma(r)}\ e^{-\gamma\ \beta}\ \beta^{r-1} \tag{5.12}$$

Expanding Equation (5.11) using Equations (5.10, 5.12) we get:

$$\hat{\beta} = \frac{\int_0^\infty \beta\ K_0\ e^{-\beta G(t)}\ \beta^{x_t-n_t}\ K_\gamma\ e^{-\gamma\ \beta}\ \beta^{r-1}\ d\beta}{\int_0^\infty K_0\ e^{-\beta G(t)}\ \beta^{x_t-n_t}\ K_\gamma\ e^{-\gamma\ \beta}\ \beta^{r-1}\ d\beta} \tag{5.13}$$

The fraction term of Equation (5.12), $\frac{\gamma^r}{\Gamma(r)} = K_\gamma$, is a constant with respect to β. Hence we can cancel all the equal constants from the numerator and the denominator of Equation (5.13):

$$\hat{\beta} = \frac{\int_0^\infty \beta^{x_t - n_t + r} \times e^{-\beta\,(G(t)+\gamma)}\, d\beta}{\int_0^\infty \beta^{x_t - n_t + r - 1} \times e^{-\beta\,(G(t)+\gamma)}\, d\beta} \tag{5.14}$$

The Gamma function of Equation (5.12) is a probability distribution function. Thus, it should integrate to 1 over the range of all possible values of β. The numerator and the denominator of Equation (5.14) are of the form of Gamma probability functions with missing constant fractions. The integrals, therefore, should integrate to the reciprocal of the missing constants K_γs. Equation (5.14) would then be:

$$\hat{\beta} = \frac{\Gamma(r + x_t - n_t + 1)}{(\gamma + G(t))^{r + x_t - n_t + 1}} \times \frac{(\gamma + G(t))^{r + x_t - n_t}}{\Gamma(r + x_t - n_t)} \tag{5.15}$$

Simplifying the above Equation of $\hat{\beta}$ by expanding the Gamma functions and eliminating the equal powers of the $(\gamma + G(t))$ terms we get:

$$\hat{\beta} = \frac{r + x_t - n_t}{\gamma + G(t)} \tag{5.16}$$

for some constants γ and r describing the prior distribution of β. Both constants, γ and r, can be set to 1 initially and will be modified through the sequential updates of the Bayesian process.

5.2 The Expectations

Since W_t is a shifted Poisson random process, the expected value of W_t given β_t is:

$$E\{W_t|\beta_t\} = \sum_{w=1}^{\infty} w \times \frac{e^{-\beta_t}\, \beta_t^{w-1}}{(w-1)!} \tag{5.17}$$

We can shift the index w to start from 0 and replace w by $w+1$:

$$E\{W_t|\beta_t\} = \sum_{w=0}^{\infty} (w+1) \times \frac{e^{-\beta_t}\ \beta_t^w}{w!} \tag{5.18}$$

$$= \sum_{w=0}^{\infty} \frac{e^{-\beta_t}\ \beta_t\ \beta_t^{w-1}}{(w-1)!} + \sum_{w=0}^{\infty} \frac{e^{-\beta_t}\ \beta_t^w}{w!} \tag{5.19}$$

$$= \beta_t \sum_{w=0}^{\infty} \frac{e^{-\beta_t}\ \beta_t^{w-1}}{(w-1)!} + \sum_{w=0}^{\infty} \frac{e^{-\beta_t}\ \beta_t^w}{w!} \tag{5.20}$$

The quantities in the summations of Equation (5.20) are all Poisson probability functions summed up over the range of all possible w values. Thus they sum up to 1's, and the expected value of W_t given β_t is:

$$E\{W_t|\beta_t\} = \beta_t + 1 \tag{5.21}$$

Now, we expect this conditional expectation of W_t over the values of β_t to get the absolute expected value of W_t at any time t, and we get:

$$E\{E\{W_t|\beta_t\}\} = E\{(1+\beta_t)\} \tag{5.22}$$

β_t is set to $\beta\ g(t)$. Thus the expectation of β_t is the expected value of β, which is $\hat{\beta}$ given the history date $\vec{x}$, multiplied by the function $g(t)$. We then get:

$$E\{W_t|\vec{x}\} = 1 + \hat{\beta}\ g(t) \tag{5.23}$$

By utilizing equation (5.16), the expected size of interruption at certain time t is given by:

$$E\{W_t|\vec{x}\} = 1 + \frac{r + x_t - n_t}{\gamma + G(t)}\ g(t) \tag{5.24}$$

From this expectation, we can expect the total number of branches X_t to be covered at any time $t > T$ for a given verification history up to time T. At the time T, we know that we have covered x_T branches. The expectation of the size of the interruption at time $T+1$, for example, is $E\{W_{T+1}|\vec{x}\}$ given that there will be an interruption at

time $T+1$. Now, our expectation that there will be an interruption at any time is the fitted correlation function, $p(t)$, at time $T+1$. Thus the overall expected value of X_t at any time $t > T$ given the verification history up to time T is the sum of all the expected sizes of interruptions after time T:

$$E\{X_t|\vec{x}\} = x_T + \sum_{j=T+1}^{t} E\{W_j|\vec{x}\} \times p(j) \tag{5.25}$$

$$= x_T + \sum_{j=T+1}^{t} \left(1+\hat{\beta}_T\, g(j)\right)\, p(j) \tag{5.26}$$

$$= x_T + \sum_{j=T+1}^{t} \left(1+\frac{\tau + x_T - n_T}{\gamma + G(T)}\, g(j)\right)\, \zeta\, f(j) \tag{5.27}$$

where ζ can either be set such that $p(t) = 1$ at the first prediction time $t = 2$:

$$\zeta = \frac{1}{ln(2)} = 1.44 \tag{5.28}$$

or, it can also be *dynamically* updated based on the verification history of the number of interruptions up to time T. For the sequential testing times $t = 1, 2, \ldots, T$, the best amplitude, ζ, that fits the cumulative function of $p(t)$ to the total number of interruptions using the Least Squares method is given by:

$$\frac{\partial}{\partial \zeta} \sum_{t=1}^{T} (n_t - \zeta\, \ln(t))^2 = 0 \tag{5.29}$$

which gives the closed form solution for ζ_T at each t as:

$$\zeta_T = \frac{\sum_{t=1}^{T} n_t \times \ln(t)}{\sum_{t=1}^{T} \ln^2(t)} \tag{5.30}$$

We refer the stopping rule that chooses the static ζ value as Static Bayesian rule (SB). For the dynamically changing ζ_T stopping rule, we refer to it as Dynamic Bayesian rule (DB).

5.3 The Stopping Criteria

The decision to stop or switch the testing technique can be viewed as two parts: a good prediction of the future coverage during testing and a good criterion that utilizes this expectation to decide when to stop. Although the assumptions of Howden's probability distributions [33] for coverage are inaccurate, the stopping criterion he chose involves calculating the confidence that the next N test cases yield no coverage. If the next N test cases are indeed executed and yield no coverage, testing will stop. This stopping criterion requires the following:

1. Having accurate branch coverage prediction for the near future $E\{X_t\}$.
2. Deciding on the confidence level C.
3. Calculating the cost effectiveness of continuing the process if the decision is to continue.

In the Howden model [33], however, two of the above three requirements are not fulfilled: the good expectation $E\{X_t\}$ and the incorporation of the testing cost in the stopping decision.

Another approach to having a stopping criterion is the use of the cost model by Dalal-Mallows [42]. The authors suggested that there is a cost of $\$a$ for each fault that is uncovered during testing and that it is subsequently exercised in the field. The cost of fixing a bug during testing is $\$b$, and there is an overall testing cost of $\$f(t)$ as a general increasing function of time. Thus the decision to stop testing is when the cost of continuing the test is greater than the cost of releasing the model. In other words, testing will stop if the expected reward at a certain time becomes zero. This criterion models the cost of testing, although it doesn't have the confidence idea involved.

The Compound Poisson model [48] used this cost model in testing VHDL models for branch coverage assuming that the decision of stopping is made at every single case of testing (every pattern applied). That makes the expectation of the branch increment at a certain period of time t to be $e\{X_t\} = E\{X_{t+1}\} - E\{X_t\}$, and the stopping decision will then be to stop when $e\{X_t\} < d = \frac{c}{a-b}$.

In this research, we used the stopping criterion similar to that in [48, 42] not only for the next simulation cycle at time t but also for the remaining expected coverage (i.e. $E\{X_t\}$) utilizing the proposed expectation of Equation (5.27) derived in Section 5.2 along with the dynamic privilege of Equation (5.30).

Another modification to this proposed stopping criterion is to incorporate the confidence-based criterion of [33] into the cost-based criterion. When the cost-based criterion suggests that the testing process be stopped, we then statistically check the confidence that there will be no more branches to be covered in the next Z simulation cycles in the future. If the confidence level is equal to or more than a preset confidence value, say C_0, we stop testing as suggested by the cost-based criterion. Otherwise, we continue the testing till the satisfactory level of confidence is reached. We refer to the dynamic Bayesian stopping rule that utilizes both the cost-based criterion and the confidence-based criterion in its stopping decision as the Confidence-Based Dynamic Bayesian rule (CDB). In the next section, we derive the mathematical formula for the confidence of having no interruptions at certain time T for the next Z simulation cycles in the future.

5.4 The Confidence Derivation

Given the random process of Equation (5.1), the confidence that there will be no interruptions in the next Z simulation cycles in the future given that the testing

process has stopped at time T is:

$$C = \prod_{t=T+1}^{T+Z} 1 - p(t) \tag{5.31}$$

where $p(t)$ is the probability of having an interruption at time t. From Equation (4.8) we get:

$$C = \prod_{t=T+1}^{T+Z} 1 - \zeta_T \, ln\left(\frac{t}{t-1}\right) \tag{5.32}$$

The index t of Equation (5.32) can be mapped to start from 1. Taking the logarithm of both sides of Equation (5.32) makes the confidence as follows:

$$ln(C) = ln\left(\prod_{t=1}^{Z} 1 - \zeta_T \, ln\left(\frac{t+T}{t+T-1}\right)\right) \tag{5.33}$$

$$= \sum_{t=1}^{Z} ln\left(1 - \zeta_T \, ln\left(\frac{t+T}{t+T-1}\right)\right) \tag{5.34}$$

The confidence level is to be checked when the cost of testing is expected to be higher than that after the release. We approximate the summation of Equation (5.34) to a polynomial function of the reciprocal of the stopped testing time T, and a linear function of both ζ_T and Z as follows:

$$ln(C) \approx \zeta_T \left(\frac{a_1}{T} + \frac{a_2}{T^2} + \frac{a_3}{T^3} + \frac{a_4 Z}{T} + \frac{a_5 Z}{T^2}\right) \tag{5.35}$$

$$= h(\zeta, T, Z) \tag{5.36}$$

where, a_1, a_2, a_3, a_4, a_5 are the constants of the Least Square solutions of the fitted function:

$$\frac{\partial}{\partial a_i} \sum_{\zeta,T,Z} \left[h(\zeta, T, z) - \zeta \left(\frac{a_1}{T} + \frac{a_2}{T^2} + \frac{a_3}{T^3} + \frac{a_4 Z}{T} + \frac{a_5 Z}{T^2}\right)\right]^2 = 0 \tag{5.37}$$

Differentiating the squared sums with respect to all a_i's gives the matrix solution:

$$\sum_{\zeta,T,Z} \zeta^2 \times \begin{bmatrix} \frac{1}{T^2} & \frac{1}{T^3} & \frac{1}{T^4} & \frac{Z}{T^2} & \frac{Z}{T^3} \\ \frac{1}{T^3} & \frac{1}{T^4} & \frac{1}{T^5} & \frac{Z}{T^3} & \frac{Z}{T^4} \\ \frac{1}{T^4} & \frac{1}{T^5} & \frac{1}{T^6} & \frac{Z}{T^4} & \frac{Z}{T^5} \\ \frac{Z}{T^2} & \frac{Z}{T^3} & \frac{Z}{T^4} & \frac{Z^2}{T^2} & \frac{Z^2}{T^3} \\ \frac{Z}{T^3} & \frac{Z}{T^4} & \frac{Z}{T^5} & \frac{Z^2}{T^3} & \frac{Z^2}{T^4} \end{bmatrix} \times \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \sum_{\zeta,T,Z} \zeta \, h(\zeta, T, Z) \times \begin{bmatrix} \frac{1}{T} \\ \frac{1}{T^2} \\ \frac{1}{T^3} \\ \frac{Z}{T} \\ \frac{Z}{T^2} \end{bmatrix} \tag{5.38}$$

The numeric solution to Equation (5.38) is:

$$\vec{a} = \{-149.0,\ 5230.0,\ -40509.0,\ -0.2,\ 1.9\} \tag{5.39}$$

5.5 The Forecasting Derivations

It is essential to predict or forecast the future of branch coverage in behavioral model verification processes. One important aspect of developing a statistical stopping rule is to direct the verification process to the testing strategy that is most likely to be efficient or to stop the verification process based on the expectation of the coverage. All the reviewed existing stopping rules, including our proposed models, use the instant expectation for the coverage at the next simulation cycle, which limits the time of making decisions on the testing process and makes them instantaneous too.

Equation (5.27) gives the expected total number of branches to be covered at any time t, given the history of testing up to time T, where $t > T$. However, we expect, as the nature of any statistical model, to have increasing prediction errors when we attempt to predict coverage in the far future of testing. To overcome this problem, we can use parts of the prediction information that would still be useful and can guide the verification process more accurately to reach better stopping decisions. We select two pieces of information:

1. The probability of having an interruption within the next Z simulation cycles given the history of testing up to time T: $\mathcal{P}(T, Z)$

2. The expected waiting time to an interruption within the same prediction window, given that there will be an interruption: $EWT(T, Z)$

Thus we derive $\mathcal{P}(T, Z)$ and $EWT(T, Z)$ and examine the ability of our proposed statistical behavior to have higher prediction accuracy.

The probability of having an interruption at any time during the next Z simulation cycles when the testing process is stopped at time T is the conjugate of the probability of having no interruption at all times from $T+1$ till $T+Z$. Thus

$$\mathcal{P}(T,Z) \;=\; 1 - Prob\{no\ interruptions\ in\ [T+1:T+Z]\} \tag{5.40}$$

From Equation (4.8) having no interruptions at all the times is the product of the conjugates of having a single interruption at a certain time t in the interval $[T+1 : T+Z]$. That makes the probability $P(T,Z)$ as:

$$\mathcal{P}(T,Z) \;=\; 1 - \prod_{t=T+1}^{T+Z} \left(1 - \zeta_T \ln\left(\frac{t}{t-1}\right)\right) \tag{5.41}$$

where ζ_T is the estimated ζ at the stopped time T.

To derive the expected waiting time to have an interruption within the next Z simulation cycles, we first derive the distribution of the waiting times to have an interruption. The probability of waiting one simulation cycle to have an interruption is the probability of having an interruption at time $T+1$. The probability of waiting n simulation cycles requires that there should be an interruption at time $T+n$ and that there should not be any interruption from $T+1$ until $T+n-1$. Thus, the probability distribution function of the waiting times, f_{WT}, is:

$$f_{WT}(t) \;=\; \prod_{i=T+1}^{T+t-1} (1 - p(i)) \;\times\; p(t) \tag{5.42}$$

From this distribution, we can estimate the conditional waiting time to have an interruption within a window of size Z, given that there will be an interruption at any time in $[T+1 : T+Z]$. This conditional estimation is the statistical expectation of $f_{WT}(t)$ divided by the probability of having an interruption in $[T+1 : T+Z]$, which is $P(T,Z)$. Thus

$$EWT(T,Z) \;=\; \frac{\sum_{t=T+1}^{T+Z} t \;\times\; f_{WT}(t)}{\mathcal{P}(T,Z)} \tag{5.43}$$

From Equations (5.42) and (5.43) we get:

$$EWT(T,Z) = \frac{\sum_{t=T+1}^{T+Z} t \times \prod_{i=T+1}^{T+t-1} [1-p(i)] \times p(t)}{\mathcal{P}(T,Z)} \tag{5.44}$$

Utilizing Equation (4.8), we get:

$$EWT(T,Z) = \frac{\sum_{t=T+1}^{T+Z} t \times \prod_{i=T+1}^{T+t-1} \left(1-\zeta_T \ln\left(\frac{i}{i-1}\right)\right) \times \zeta_T \ln\left(\frac{t}{t-1}\right)}{\mathcal{P}(T,Z)} \tag{5.45}$$

5.6 Summary

In summary, we derived two sets of equations: the proposed stopping rules equations, and the forecasting error equations of the proposed model. The three proposed stopping rules are the Static Bayesian stopping rule (SB), the Dynamic Bayesian stopping rule (DB), and the Confidence-Based Dynamic Bayesian stopping rule (CDB). The stopping rules are described in Equations (5.24), (5.28), (5.30), and (5.36).

Two quantities are defined to measure the prediction accuracy of the proposed model: the probability of having interruptions during the next Z simulation cycles and the expected waiting time to reach the first interruption in the same future window. Equations (5.41) and (5.45) evaluate the prediction errors that will be estimated in Chapter 7.

Chapter 6

EXPERIMENT SETUPS

In order to verify the static and dynamic Bayesian models developed in Chapter 5, a set of experiments were performed on a set of collected VHDL models. We first used a model named Sys7 from the VLSI Design lab at Colorado State University in our research. The Sys7 model is a real-time model using the tree-matching algorithm to detect and recognize images. The model contains about 3800 lines of VHDL code (LOC) with 591 branches. The system is organized with several systolic arrays as the processing elements and a global controller. The use of the systolic array increases the sequential depth, thus making validation of the model more difficult. Similar to the complexity of Sys7, the 8251 model is a serial universal asynchronous receiver/transmitter microprocessor from Intel USART [53] used for data communication as a peripheral device. This Intel model contains about 3000 lines of VHDL code with 207 branches in 9 levels of hierarchy, which makes reaching branches at the lower levels difficult.

Lastly, we added to our suite a large collection of behavioral VHDL models from two sources: Intrinsix Corporation, an independent provider of ASIC and System Design and Verification Services [53], and the Collaborative Benchmarking Laboratory, the Department of Computer Science at North Carolina State University [52]. The collected models' sizes range from 2000 to 6000 LOC with 250-400 branches. Table 6.1 lists the models with a brief description of operation. Table 6.2 gives the statistical information about each model. The second column of Table 6.2 is the number of

Model	Description
Sys7	Real-Time Tree-Matching Recognizer
8251	Serial Universal A/Synchronous Receiver/Transmitter
B01	16 megabit byte-wide top boot 150 ns
B04	CMOS SyncBIFIFO 256x36x2
B05	SyncFIFO With Bus-Matching 1024x36
B06	SyncFIFO 2048x36
B07	SyncFIFO 2048x36
B08	CMOS SyncBIFIFO 1024x36x2
B09	SyncFIFO With Bus-Matching 1024x36
B10	SyncFIFO 2048x36
B11	CMOS SyncBIFIFO 512x36x2
B12	SyncFIFO With Bus-Matching 512x36
B14	SyncBiFIFO With Bus-Matching 512x36x2
B15	SyncBiFIFO With Bus-Matching 1024x36x2

Table 6.1: Benchmark Descriptions

lines of the VHDL model. The third column is the number of concurrent processes in each benchmark; the more process blocks we have, the more complicated model we simulate. The fourth through the seventh columns list the number and type of prime input and output signals of the benchmarks. B in the eighth column is the number of branches in each benchmark. The last column reflects the depth of the control flow graph of the branches. Having the branches in three levels, for example, means that in order to cover a certain branch in the deepest level, one must first cover its two dominant branches on the first and the second levels. CFG depth indicates the level of difficulty covering all the models' branches.

6.1 Code Coverage Collection History

Commercial and academic developers have focused on VHDL compilers since the standardization of the VHDL language in the 80s. Unfortunately, code coverage features in a language such as VHDL have not been considered until recently. In 1989

Model	LOC	Processes	I/P	O/P	Data	Control	B	CFG Depth
Sys7	3785	62	69	43	62	7	591	7
8251	3113	3	20	6	8	12	207	9
B01	1880	7	110	1	101	9	373	8
B04	4657	42	87	10	72	15	251	3
B05	5015	46	91	6	72	19	302	6
B06	4667	39	88	6	72	16	225	6
B07	4710	39	88	6	72	16	225	6
B08	4949	52	87	10	72	15	296	3
B09	4963	46	91	6	72	19	302	6
B10	4777	39	88	6	72	16	225	6
B11	4752	42	87	10	72	15	251	3
B12	4973	46	91	6	72	19	302	6
B14	5498	53	93	10	72	21	399	6
B15	5770	66	93	10	72	21	470	6

Table 6.2: Benchmark Statistics

a free source code for a VHDL compiler was published by the University of Pittsburgh and later modified to be capable of simulating VHDL designs [54]. This compiler and its simulator is very limited to a certain subset of the VHDL standard syntax and rules; however, it was the best choice for further enhancement by adding the code coverage features and for use in this research. In 1996 the Departments of Electrical Engineering and Computer Science at Colorado State University started to append the compiler and simulator codes to include branch and bit coverage calculators for given VHDL designs [55]. The modified versions were then used to theoretically prove the value of using code coverage in behavioral verification.

When we started this research, the Sys7 benchmark was the largest VHDL model we had in our suite. The Sys7 benchmark was remodeled from its original version to overcome the limitations of the Pittsburgh compiler. Table 6.3 lists the limitations currently found in the Pittsburgh tool. We then collected code coverage results to help us understand the behavior of branch coverage. Later, when we considered the

Intel 8251 model, it was almost impossible to convert the VHDL constructs into the limited syntax recognized by the Pittsburgh tool.

AFTER	ALIAS	ARRAYS
ASSERT	Attributes	Ascending Indices
BUILT-IN Functions	CONFIGURATION	FUNCTION & PROCEDURE
GENERATE	GENERIC	INTEGER Operations on VARIABLES
NATURAL	NESTED CASE BLOCKS	NESTED NOT GATES
Numbers in base notation	Numeric powers	OTHERS Assignment
PACKAGE	SHARED VARIABLE	STRING and TEXT
TYPE and SUBTYPE	WAIT ON	32-Bit Maximum Width

Table 6.3: VHDL Syntax Not Implemented in the Pittsburgh Tool (MVSIM)

Meanwhile, *TransEDA Limited* has implemented an interface tool named *VHDL-Cover* that is capable of reporting many different types of coverage on both VHDL and Verilog languages [56] (see Table 6.4 for various types of coverage in VHDL-Cover). For academic purposes, the tool was licensed to the VLSI Design lab at Colorado State University for two years. VHDLCover automatically instruments a given VHDL model for a specific coverage and then uses the *MentorGraphics Quick-VHDL* package for compilation and simulation. Results of simulations are reported in a file, which will then be used to assess and debug the VHDL codes.

One disadvantage of using VHDLCover in our experiments is the simulation-time overhead needed to report coverage results. VHDLCover, unfortunately, reports only final coverage results of a testbench for a given stimulus, ignoring all intermediate coverage results during the testing process. For our research, we must collect coverage results for every simulation cycle (clock cycle) after applying test patterns to the model in order to extract needed statistical process properties.

One way to report branch coverage for every simulation cycle (clock cycle) using this tool is to repeatedly run many testbenches, each having the ordered accumulated number of test patterns, and then collect the results from simulating each testbench. For example, if we were to run 100 random test patterns to a given VHDL model

Coverage Type	VHDL Syntax Used
Statement Coverage	Processes
	Signal assignments
	Variable assignments
	Procedure calls
Branch Coverage	If statements
	Case statements
	Next-Exit statements
	Conditioned concurrent assignments
Condition Coverage	Check all possible combinations of conditions
Path Coverage	Order of statements under conditions
Trigger Coverage	Check sensitivity list signals
Signal Coverage	Check all signals over their ranges
Toggle Coverage	Bit coverage

Table 6.4: Coverage Types of VHDLCover

and collect coverage, we would then have to build 100 different testbenches. The first testbench has only one test pattern, which is the first test pattern of the simulation. The second testbench will then have the first and the second test patterns, and so on. Obviously, this method is not practical; in fact, if we were to run 300,000 patterns for a model like Intel 8251 and extract acceptable statistics out of the VHDLCover interface, we would then need almost 10 years to finish simulating the model. (Calculations are based on the performance of the local Unix machines.) Therefore, we needed a different way to solve this problem.

The mechanism of operation of the VHDLCover interface, as introduced earlier, is to first instrument the given VHDL model and produce another VHDL model that calculates coverage through extra inserted signals and variables. This gave us the idea of *manually* instrumenting the coverage signals in a given VHDL benchmark before a simulation that uses only QuickVHDL. The procedure involves adding one extra bit_vector array signal at the *architecture* of the VHDL benchmark that has the size of the number of branches in the model. The bits of this signal are initially reset to

zeros, corresponding to having all the branches at time zero of simulation not covered. Under each branch statement of the model, one of the bits of that signal is assigned to '1' when that branch is visited. Thus, at each simulation cycle (clock cycle), we would know how many bits of that signal have 1's, and we can then easily report the branch coverage sequence. This method takes a lot of effort and careful manual insertions, but it is only a one-time job. Finally, we had in our suite 13 ready-to-simulate VHDL benchmarks with their testbenches successfully and correctly tested for the manual insertion. We checked the manually instrumented models against the ones instrumented by VHDLCover tool for some of the test benches and got a perfect match. Now, the simulation time needed to run the 300,000 patterns for 100 different seeds is less than 10 minutes as opposed to 10 years.

6.2 Verification Strategies

When verifying VHDL models, hardware designers often start with a limited number of functional simulations that represent common or typical usage of the design's capabilities. The advantage of using such test cases is that they exercise many parts of the design against the required function. Thus, the functional patterns are expected to give higher coverage. However, since these patterns are manually generated, they are very limited in number and/or very expensive to generate. *Random testing*, sometimes called *statistical testing*, is applied [46] after functional testing. Figure 6.1 shows how effective the functional testing is compared with the random testing for a given VHDL model.

Other methods of test generation have been discussed and researched [15, 16, 18, 36, 57, 58, 59, 60, 61] in the past. Most of these methods, however, are based on knowledge of the model under the test, and they are categorized under the *white-box* test methods. In this research, we focus on estimating coverage of random testing

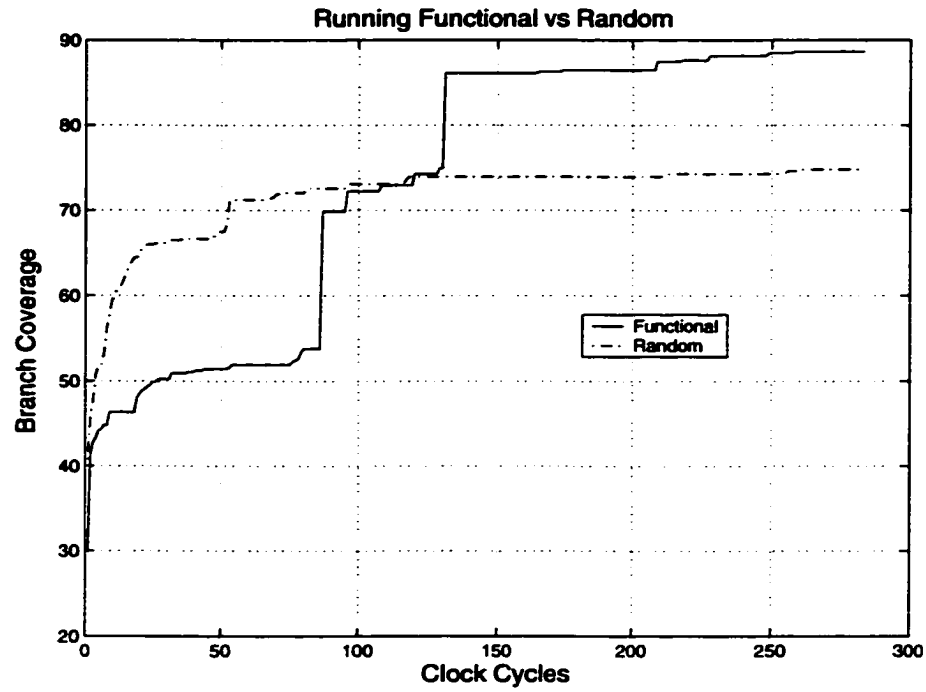


Figure 6.1: Functional vs. Random Testing

because it is still the dominant part of the testing process and because the decision on when to stop random testing is much more important than any other limited time testing strategy.

80K Setup				800K Setup			
Phase	Patterns	Clock Cycles	Seed	Phase	Patterns	Clock Cycles	Seed
Functional patterns if any				Functional patterns if any			
Random $h = 1$	10,000	10,000	1	Random $h = 1$	100,000	100,000	1
Random $h = 2$	10,000	20,000	2	Random $h = 2$	100,000	200,000	2
Random $h = 4$	5,000	20,000	4	Random $h = 4$	50,000	200,000	4
Random $h = 6$	5,000	30,000	6	Random $h = 6$	50,000	300,000	6

Table 6.5: Experiment Setups

Random test patterns are usually generated uniformly with a preset seed value. Bits of the testing patterns are all binary digits, i.e. 1's and 0's. Thus, binary random

generators, or better converted-integer random generators, are used to produce one test pattern at a time. Each test pattern is a string of bits of the same size as the number of prime input bits of the design. At every simulation cycle (clock cycle), a new pattern is generated and fed into a model. Nevertheless, sometimes, especially when data bits have to be fixed for certain times, holding a pattern for a certain number of clock cycles seems to help improve the coverage. In our benchmark suite, we examined this idea against the idea of applying pure random test patterns without holding them steady for a period of clock cycles. We then randomly used four different phases of different size, each of a different holding number and seed. Table 6.5 shows the setup for the experiments that are also used throughout this research, unless otherwise specified. The "h" values in Table 6.5 indicate the number of clock cycles a random input was held in the given random testing phase. We used two different setups for the experiments; the second setup was 10 times larger in terms of the number of test patterns than the first (named 80k and 800k).

To show the efficiency of using mixed strategies of random testing over using ordinary random testing of the same length, we conducted the 80k setup experiments on every VHDL benchmark and compared coverage results against ordinary random setup. Figure 6.2 shows an example of branch coverage activity of each setup for a given model. Figure 6.3 shows the overall increase in branch coverage for all the benchmarks when 80k and 800k setups are used against the setups where no patterns were held for more than one clock cycle. Out of the 13 models we tried with the 80k setup, two models yield to a lower branch coverage of 2 branches. In the 800k setup experiments, only one model has lower branch coverage of one branch when using the multiple phase setup. A maximum of 63 more branches are covered using the 80k setup, and 80 more branches are covered using the 800k setup. On the average, an increase of 14% to 17% in branch coverage is gained if the multiphase setups are used.

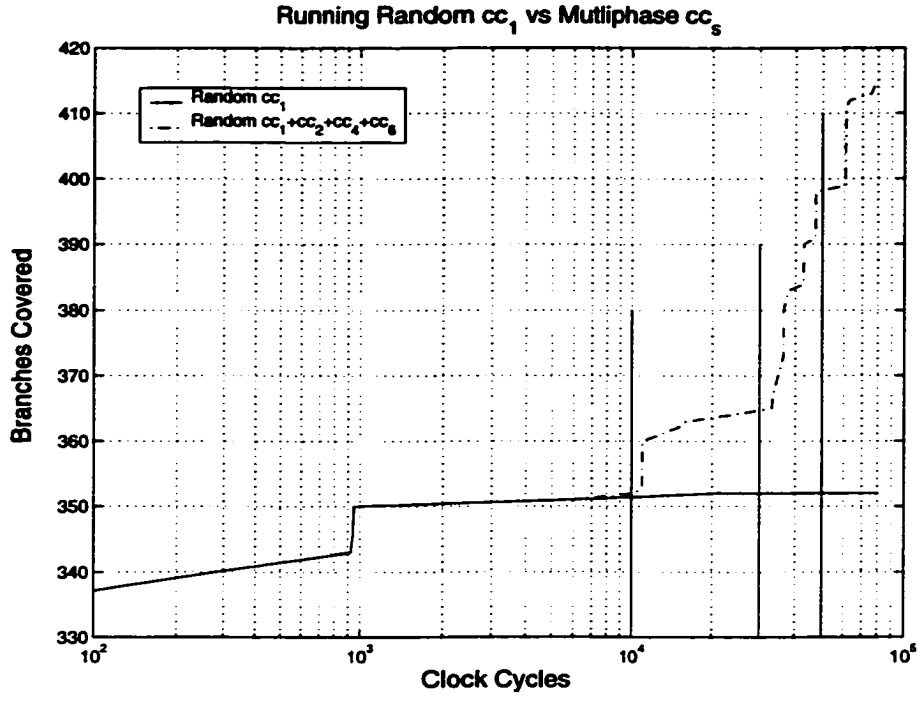


Figure 6.2: Example of Coverage Increase with Multiphase Setup

6.3 Stopping Rules Experiments

We compared five stopping rules and their variations (reviewed in Chapter 3) and our proposed stopping rules (presented in Chapters 4 and 5). The rules are listed in Table 6.6, where each rule is abbreviated. One might consider the original setup as a manual stopping rule that is to be compared with the other 10 stopping rules.

We used the MentorGraphics QuickVHDL tool to run the testbenches of the experiments. At each phase of the experiment, we subtracted the contribution of its preceding phase from the cumulative branch coverage and fed the data to the stopping rule. The stopping rule will then decide at what point on that data the testing should be stopped and switched to a different phase. If, for some cases, the decision is not to stop that phase, the random testing will then be continued until the end of the mission testing time, i.e. 80,000 simulation cycles (clock cycles) for the 80k setups and 800,000 clock cycles for the 800k setups.

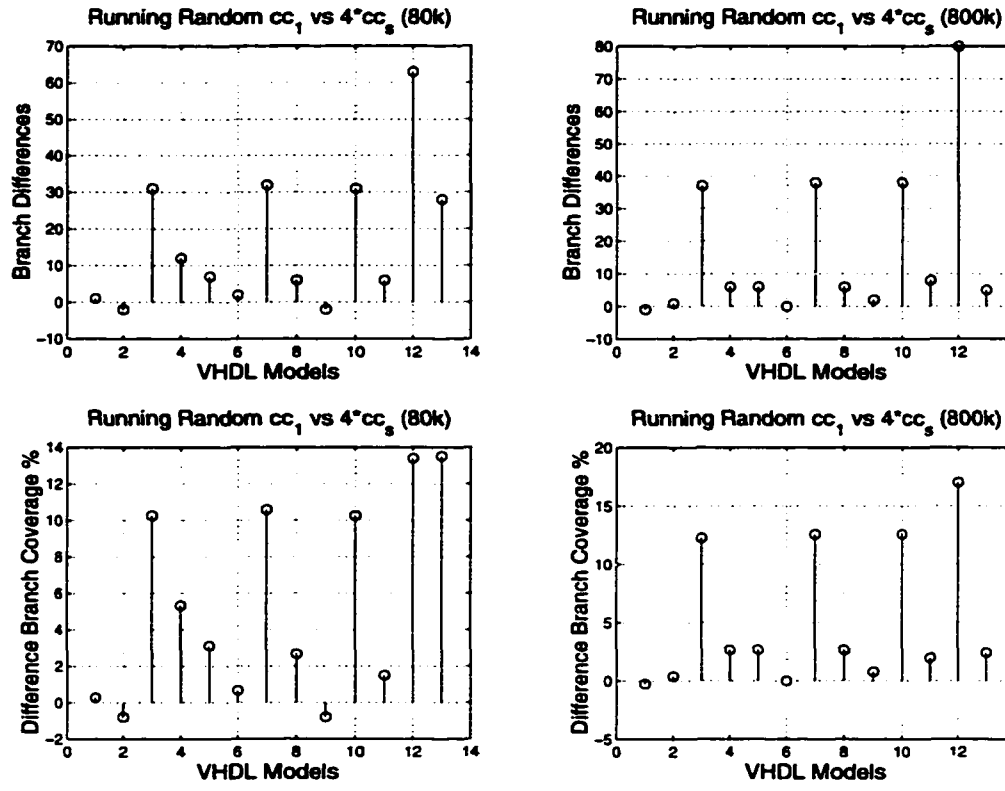


Figure 6.3: Applying Different Pattern Holding Numbers

#	Rule	Abbreviation
1	Original Run without stopping	Orig
2	Sequential Sampling Fixed γ	SS1
3	Sequential Sampling Variable γ	SS2
4	Howden First Formula	HW1
5	Howden Second Formula	HW2
6	Binary Markov Model	BM
7	Dalal-Mallows Cost Rule	DL
8	Compound Poisson Rule	CP
9	Static Bayesian (Proposed)	SB
10	Dynamic Bayesian (Proposed)	DB
11	Confidence-Based Dynamic Bayesian	CDB

Table 6.6: Stopping Rules

The choices for the stopping rules' parameters were set so that all the stopping rules are equally compared against one another for all different VHDL benchmarks:

- For the sequential sampling models, SS1 and SS2, the α value is set to 0.5, which means that if the objective is met, then we are totally unaware of that. The β value is set to 0.01, indicating that we are 99% confident that if the objective is not met, we have a correct assumption. In any case, these values have little impact on the slope of the boundary line of the decision. The discrimination ratio γ is set to 250 for the fixed rule SS1 and 100 as the initial value for the variable rule SS2. This γ value means that we are willing to accept 250 patterns at the beginning of the test without any coverage. This value is reduced to 100 in the case of SS2 since it will be increased later. The parameters α and β remain the same when SS1 and SS2 rules are used in the 800k setup since they are confidence probabilities. The γ value is magnified by ln(10) to correct the slope of the decision line of the rule when the whole figure of the coverage process is enlarged.

- For the Howden's stopping rules, HW1 and HW2, we set the confidence level at 95% and the maximum probability that a pattern may produce new coverage at 0.03, which corresponds to expecting new coverage after an average of 30 testing patterns. In fact, we set a value N_0 to 30 for almost all the rules so that if no coverage is observed during the first N_0 test patterns, testing will be stopped. This parameter is also applied to the Sequential Sampling rules, SS1 and SS2. The level of the probability B is reduced to be 0.003 in the 800k setup experiments.

- The Binary Markov stopping rule, BM, has exactly the same parameter values as the Howden's. The correlation constant ρ, however, is chosen to be 0.5, which indicates a moderate pattern correlation.

- The Compound Poisson stopping rule, CP, uses the following parameters: $a = 6$, $b = 1$, and $c = 1 \times 10^{-6}$. This gives the stopping criterion $d = 0.2 \times 10^{-6}$ where if the difference of expectations becomes lower than d, the testing process will be stopped. One problem with this rule is that the initial assumption requires that there should be at least one new branch covered at each time step during the testing process. This problem can be partially solved by sampling the outcomes as if the testing patterns are packed in groups, each contributes to coverage. We first chose to pack each 16 data points together; however, this amount of packing is enough for the Sys7 model but not for the other VHDL benchmarks. Thus, we chose to pack every 300 patterns in order to have a reasonable data set to feed to the CP stopping rule. We further increased this packing number to 3000 for the 800k setup experiments to match the application.

- The Dalal-Mallows stopping rule, DL, has similar values for the costs as the CP stopping rule does, except that in order to have equal costs, they should be divided by the amount of packing used in the CP stopping rule. Thus, for the DL stopping rule, the parameters are set to: $a = 0.0625$, $b = 0.375$, and $f = 10^{-6}$ for the 80k setup. The value f is reduced to 10^{-7} for the 800k setup experiments.

In our proposed stopping rules, SB, DB, and CDB, the dominant parameters that should be set are the d values. We chose to have different decreasing d values for each phase of the testing, believing that covering branches in the later stages of testing is more difficult than at the beginning. Thus, in order to give the later phases a better chance to cover more branches, we should stop when the expected value of coverage at the early phases is not that high. The values used are $d = 0.02,\ 0.01,\ 0.005,\ 0.001$. These values correspond to the costs of \$0.1 and \$0.6 for covering branches during

testing and in the field, respectively. The unit cost of testing is then \$0.01, which makes the first d value start from 0.02. The logarithmic decrease in the value of d across the phases comes from imitating the fact that the coverage takes the logarithmic shape. The d values in the 800k setup were divided by 10 to match the application costs. We also imposed the criterion of $N_0 = 30$ (or $N_0 = 300$ for the 800k setup) in the proposed stopping rules, in order to have equivalent comparisons.

The proposed rules automatically set their own internal parameters based on the history of testing as mentioned previously, which makes them more robust. For the Static Bayesian stopping rule, SB, we set ζ so that the probability of having an interruption in the first predicted time is 1. The analysis was shown previously in Equation (5.28). For the confidence level of the Confidence-Based Dynamic Bayesian stopping rule, CDB, we chose the same confidence level as in HW1, HW2, and BM (C =95%). Table 6.7 lists the entire stopping rules' parameters used in both the 80k and the 800k-setups.

6.4 Figure of Merit fm: An Efficiency Evaluating Function

Results of applying a stopping rule to a certain benchmark are the total number of simulation cycles used, tt, and the final cumulative branch coverage percentage reached, cov, using these simulation cycles. If the total number of simulation cycles used by a stopping rule is more than another stopping rule to reach the same or even less coverage, then it is clear that the second stopping rule outperforms the first one. However, when the coverage increased when more simulation cycles are used, then it is not obvious which stopping rule has the better efficiency. Thus, an objectively evaluating function is needed.

The development of such evaluating function is not straightforward, depending on the model under verification and the reliability requirement of its application.

Rule		Parameters
SS1	80k	$\alpha = 0.5,\ \beta = 0.01,\ \gamma = 250,\ N_0 = 30$
	800k	$\alpha = 0.5,\ \beta = 0.01,\ \gamma = 576,\ N_0 = 30$
SS2	80k	$\alpha = 0.5,\ \beta = 0.01,\ \gamma_0 = 100,\ N_0 = 300$
	800k	$\alpha = 0.5,\ \beta = 0.01,\ \gamma_0 = 230,\ N_0 = 300$
HW1	80k	$C = 0.95,\ B = 0.03,\ N_0 = 30$
	800k	$C = 0.95,\ B = 0.003,\ N_0 = 300$
HW2	80k	$C = 0.95,\ B = 0.03,\ N_0 = 30$
	800k	$C = 0.95,\ B = 0.003,\ N_0 = 300$
BM	80k	$C = 0.95,\ B = 0.03,\ \rho = 0.5,\ N_0 = 30$
	800k	$C = 0.95,\ B = 0.003,\ \rho = 0.5,\ N_0 = 300$
DL	80k	$a = 0.0625,\ b = 0.375,\ f = 10^{-6},\ N_0 = 30$
	800k	$a = 0.0625,\ b = 0.375,\ f = 10^{-7},\ N_0 = 300$
CP	80k	$\alpha = 8,\ \beta = 2,\ pack = 300,\ d = 2 \times 10^{-7},\ N_0 = 30$
	800k	$\alpha = 8,\ \beta = 2,\ pack = 3000,\ d = 2 \times 10^{-7},\ N_0 = 300$
SB	80k	$d = 0.02,\ 0.01,\ 0.005,\ 0.001,\ N_0 = 30,\ \zeta = 1.44$
	800k	$d = 0.002,\ 0.001,\ 0.0005,\ 0.0001,\ N_0 = 300,\ \zeta = 1.44$
DB	80k	$d = 0.02,\ 0.01,\ 0.005,\ 0.001,\ N_0 = 30$
	800k	$d = 0.002,\ 0.001,\ 0.0005,\ 0.0001,\ N_0 = 300$
CDB	80k	$d = 0.02,\ 0.01,\ 0.005,\ 0.001,\ N_0 = 30,\ C = .95$
	800k	$d = 0.002,\ 0.001,\ 0.0005,\ 0.0001,\ N_0 = 300,\ C = .95$

Table 6.7: Stopping Rules' Parameters

Game software, for example, may not value the gained coverage more than the cost of testing it. In contrast, critical software applications such as NASA missions tolerate spending more testing time to ensure high reliability. Thus, we define a measure that utilizes the costs associated with the verification process and its goals:

Assume that covering one branch can, on average, translate to a potential saving of $\$a$ during the product's lifetime in terms of bug fixing, repairs, etc., and assume that the computing cost for testing for a time unit (clock cycle) is $\$b$. If the required branch coverage is $C\%$, the total number of branches in the model is B, and the maximum amount of test time allowed is X clock cycles, then the net profit of the verification process is:

$$Net = a\,B\,C - b\,X \tag{6.1}$$

Equation (6.1) can be normalized in terms of potential savings and computing costs to define a *Figure of Merit* function, fm, as:

$$fm = cov - \alpha\, tt \tag{6.2}$$

where *cov* is the coverage percentage reached, and *tt* is the amount of test time spent to achieve the coverage. *tt* is measured using the number of clock cycles in this study. α is the normalized cost of testing per test time unit (per clock cycle) and can be formulated as:

$$\alpha \approx \frac{b\,X}{a\,B\,C} \tag{6.3}$$

Every application has a different α value, depending on the ratio of the testing cost and coverage requirement. For $a = \$10$, $b = \$10^{-6}$, $C = 75\%$, and $X = 100,000$ clock cycles, α is a value less than or equal 53×10^{-6} for a model of $B = 250$ branches.

The larger value of fm obtained from applying stopping rules will point out the best stopping rule for that specific application. Comparisons throughout our results analysis exploit a wide range of α values centered around 50×10^{-6}.

6.5 Forecasting Experiment Setup

One of the main goals in developing the proposed statistical model for behavioral verification is forecasting coverage during the verification process. We derived in Chapter 4 the probability of having an interruption at a certain future time point. We also derived the expected waiting time to that interruption, given that there will be an interruption within a certain future interval of time. To examine the accuracy of our statistical modeling in forecasting coverage, we conducted two sets of experiments: one to extract the exact probability of having an interruption within the next Z simulation cycles given that the testing process stopped at certain time T and the other experiment to have the exact waiting time to that interruption

within the same time interval. The experiments required setting certain stop points from which to start the forecasting process and also required setting certain window sizes of future forecasting for each stop point. We define a block of testing time as 100 simulation cycles (clock cycles). We ran 100 different seeded patterns each for 1 million simulation cycles (=10,000 blocks) for all the benchmarks in our suite. We chose to have 10 different window sizes starting at a size of 10 blocks of future forecasting with increments of 10 blocks up to 100 blocks. (The window sizes were $Z = [10, 20, \ldots, 100]$ blocks.) Thus, the process of extracting statistics out of experiments was conducted $13 \times 10 \times 100 \times \frac{1,000,000}{100} = 130$ million times. At each time, we measured the frequency of having interruptions within the designated interval $[T : T+Z]$ across the 100 different seeded runs and the average waiting times to the first interruption within the same interval. Results of the statistics extracted from these experiments and the predicted probabilities and expected waiting times will be discussed in detail in the next chapter.

Chapter 7

EXPERIMENTAL RESULTS

We present in this chapter the experimental results and their analysis in two parts: the first part presents the efficiency of different stopping rules and a comparative study including three proposed stopping rules (SB, DB, and CDB). We evaluated the figure of merit function defined in Equation (6.2) for a wide range of α values to determine the efficiency of a given stopping rule. Although the figure of merit function gives an objective measure of the efficiency for a given stopping rule, it requires the knowledge of α that is related to the costs of testing and the benefits of coverage. The α value also reflects the determination of the stopping rules' parameters that should match the verification environment. A direct relationship between the choice of the stopping rules' parameters and the verification costs is not straightforward or even possible to obtain. Thus, estimates of such parameters are common, especially when facing different verification environments. For this purpose, we define a *Degree of Inefficiency* function (DOI) that estimates the expected loss in the fm value for a given stopping rule compared with that of the highest possible fm value for all α. Suppose that for a given benchmark a set of N stopping rules were applied and yielded the figure of merit function: $fm_1(\alpha)$, $fm_2(\alpha)$, ..., $fm_N(\alpha)$. Let α_M be the maximum allowed application constant which is often set to twice the typical α value for a given application. Let $\vec{a} = [0, \alpha_1, \alpha_2, \ldots, \alpha_M]$ be the intersection points of all the fm functions as illustrated in the hypothetical example in Figure 7.1. We define

the Degree of Inefficiency function for a given stopping rule x given α_M, DOI(x), as follows:

$$DOI(x) = \frac{1}{\alpha_M} \sum_{i:\vec{a}} \int_{\alpha_i}^{\alpha_{i+1}} [B_i(\alpha) - fm_x(\alpha)] \, d\alpha \tag{7.1}$$

where $B_i(\alpha)$ is the highest possible fm vlaue in the interval $[\alpha_i : \alpha_{i+1}]$. It is clear that the DOI is not a function of α; rather, it is the measure base on the aggregated of a wide range of α values. Therefore, it is a more objective measure than the figure of merit.

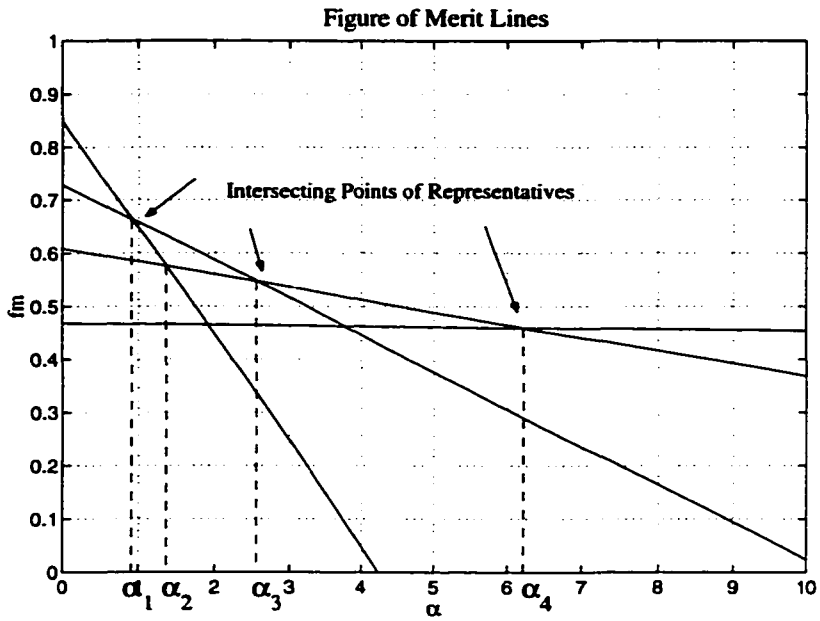


Figure 7.1: Intersecting α Points in the Figure of Merit Lines

The second part of the experimental analysis includes the forecasting experiments of the proposed dynamic Bayesian model given a verification history up to a certain testing time T and for the next Z simulation cycles. We show the ability and the accuracy of the proposed model in predicting the coverage during the verification process.

7.1 Comparative Study I

This comparative study used the 80k setup described in Chapter 6. Table 7.1 shows the results of the total number of covered branches and the total number of simulation cycles used by a given stopping rule in a given benchmark. For the entries in Table 7.1, the number on the top is the number of branches covered during the verification process, and the number at the bottom is the total number of test patterns applied using the corresponding stopping rule. The column marked "Orig" gives the number of covered branches and the number of test patterns used when no stopping rule was used. The total number of branches in each benchmark can be found in Table 6.2. Appendix A shows the detailed results of each stopping rule for all the benchmarks used in the 80k setup. The 'Δ' values in the tables of Appendix A indicate the number of new covered branches by the given phase, and the 'CC' values indicate the number of simulation cycles (clock cycles) applied.

The results show that applying any of the stopping rules saves large number of simulation cycles (clock cycles) while maintaining high coverage. For example, applying the stopping rules in verifying the Sys7 benchmark saved 96% to 98% of test cases, while at most 5% of the branches were missed. For the B14 benchmark, the stopping rules saved 73%-99% while missing at most 3% of the branches. Thus applying stopping rules to behavioral model verification saves testing time without much loss in quality.

To better analyze the results, we grouped the stopping rules for a given benchmark model that yielded the same coverage into sets. Among each set, we selected the stopping rule that used the least number of testing patterns as a leading candidate. For example, 5 stopping rules yielded the same branch coverage of 536 branches in the Sys7 benchmark. The stopping rules in this set include SS1, HW1, HW2, BM, and DL, and the number of test patterns used are 1039, 927, 969, 1025, and 1235,

respectively. Thus, the HW1 stopping rule used the least number of simulation cycles to cover the same number of branches, and it represents its group. The comparisons take place now among the candidate stopping rules for each given benchmark model. Thus, the figure of merit function, fm, points out which candidate stopping rule outperforms the others for a given benchmark at certain α value. Appendices C and D show the detailed figure of merit lines for each benchmark in both setup environments (80k and 800k). The candidate stopping rules had the highest fm values in some interval range of α values. For example, Figure 7.2 shows four intervals of α where the candidate stopping rules Orig, CP, HW2, and DB had the highest fm values for the B09 benchmark. We extracted the intervals of α values from the plots of Appendices C and D where the fm values are the highest and illustrate them in Figures 7.4 and

Model	Orig	SS1	SS2	HW1	HW2	BM	DL	CP	SB	DB	CDB
Sys7	568	536	538	536	536	536	536	547	535	535	535
	54283	1039	1858	927	969	1025	1235	6287	661	563	569
8251	161	73	73	79	79	81	75	112	74	73	67
	81500	3259	3812	2769	2906	3033	5712	9600	2275	2239	2091
B01	200	177	142	128	155	128	128	135	128	128	128
	80000	8169	3352	1010	11084	1211	1211	4200	1854	914	897
B04	223	220	218	206	214	214	219	217	199	202	202
	80000	10282	5047	1894	2468	2557	11755	17100	631	674	742
B05	259	234	251	251	251	251	252	253	232	233	233
	80000	10795	7122	2092	2343	2431	5318	10800	808	745	744
B06	210	192	192	192	192	192	204	204	192	192	192
	80000	8725	4618	1240	1407	1439	7110	4500	673	708	708
B07	210	196	198	196	204	196	196	204	195	195	195
	80000	8963	4660	1322	3904	1621	1132	4500	789	704	731
B08	274	268	268	263	263	273	273	273	273	273	273
	80000	12447	6122	1392	1405	2283	8427	9600	2249	2033	1829
B09	260	234	251	234	251	251	252	253	232	233	233
	80000	10795	7122	1512	2053	2470	5324	7800	809	734	735
B10	210	197	198	204	204	204	196	208	208	208	208
	80000	9068	4660	1488	1711	1781	915	4200	2181	1488	1240
B11	223	220	218	206	214	214	219	217	199	202	202
	80000	10282	5047	1894	2468	2557	11755	17100	631	674	742
B12	259	234	251	234	251	251	252	253	232	233	233
	80000	10795	7122	1545	2085	2462	5318	6900	808	745	744
B14	257	248	248	244	244	244	248	253	245	245	245
	80000	11367	5712	1892	1900	1991	1618	21000	1982	735	748
B15	415	351	351	350	350	350	418	383	364	364	364
	80000	16190	7906	1892	1900	1991	80002	9000	2080	2298	2010

Table 7.1: Stopping Rules' Coverage Results

7.5. The indices on the intervals of Figures 7.4 and 7.5 point to their corresponding stopping rules as shown previously in Table 6.6.

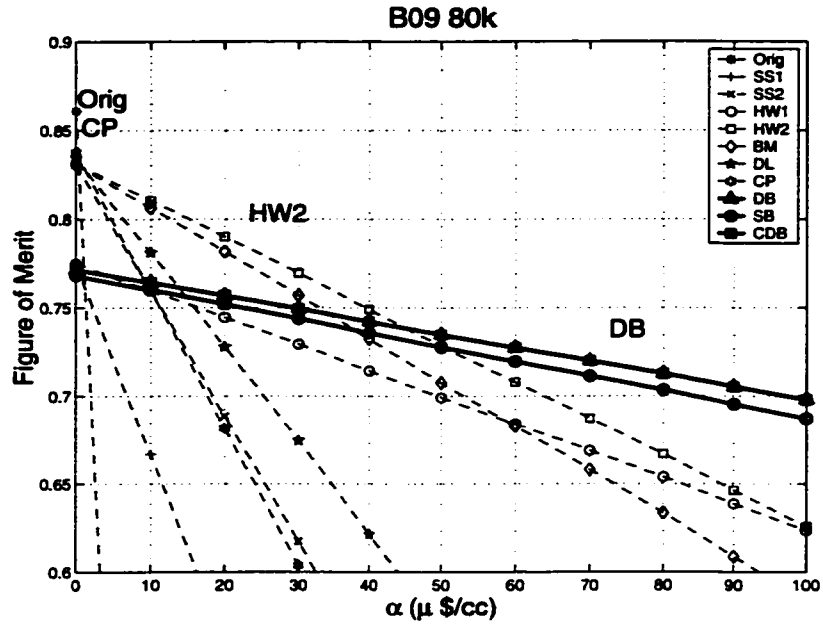


Figure 7.2: B09: Figure of Merit Lines

Two interesting results of the B10 and the B14 benchmarks in the 80k setup need to be highlighted. The CDB stopping rule yielded the highest fm value for most of the α values starting at 0.11×10^{-6} up to ∞ for the B10 benchmark. The DB stopping rule performed similarly to the CDB except being in a closed interval of α values but for most of the α range up to 92.84×10^{-6}. In the interval $0 \leq \alpha \leq 0.11 \times 10^{-6}$, the original run without applying any stopping rule yielded the highest fm value. Obviously, when the cost of testing is negligible, there is no need to apply any stopping rule.

It is also interesting to point out that, for the B14 benchmark model, 245 branches were covered using 735 test patterns when the DB stopping rule is applied. A fewer number of branches were covered using almost twice as many testing clock cycles as when using four other stopping rules: HW1, HW2, BM, and SB. The reason why

we sometimes get lower coverage while using more testing patterns is clearly due to the sequential nature of the behavioral models, which is one of the issues to be solved when developing the DB stopping rule. Due to the dynamic updating of the expectation, the DB stopping rule tends to force the switching of testing strategies earlier than other stopping rules. The more difficult-to-cover branches tend to be covered in different phases using different testing strategies, thus reducing the number of test vectors required early in the verification process while maintaining or further improving the overall coverage. This is clearly demonstrated by the results for B14 shown in Table A.13 where HW1/HW2/BM allow a tremendous amount of effort during a phase, while leaving no coverage interruptions during the remaining testing phases. When there is no interruption during a testing phase, stopping rules will not be effective in guiding the verification process, which results is zero coverage for B14 during phases 2, 3, and 4. The DB stopping rule, on the other hand, forces the process to stop early during phase 1 with lower coverage but leaving room for the

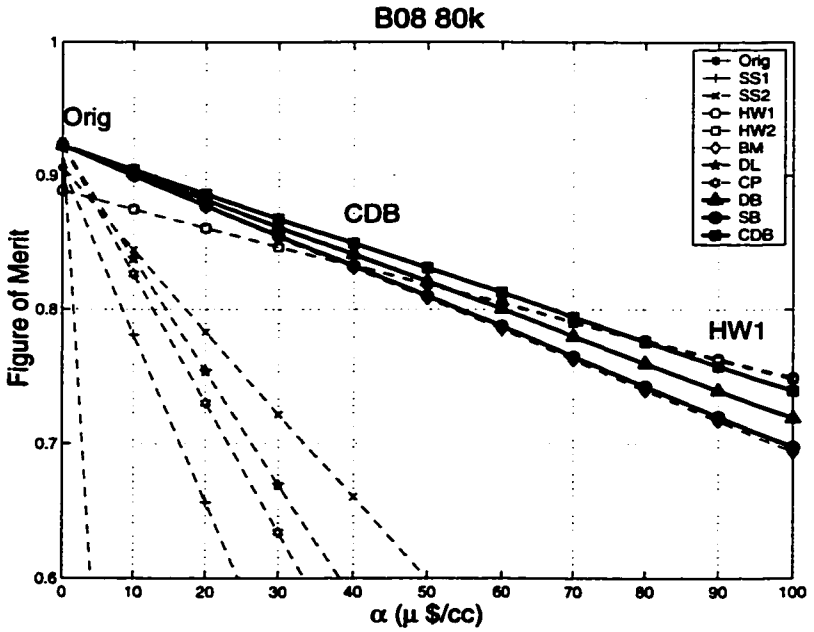


Figure 7.3: B08: Figure of Merit Lines

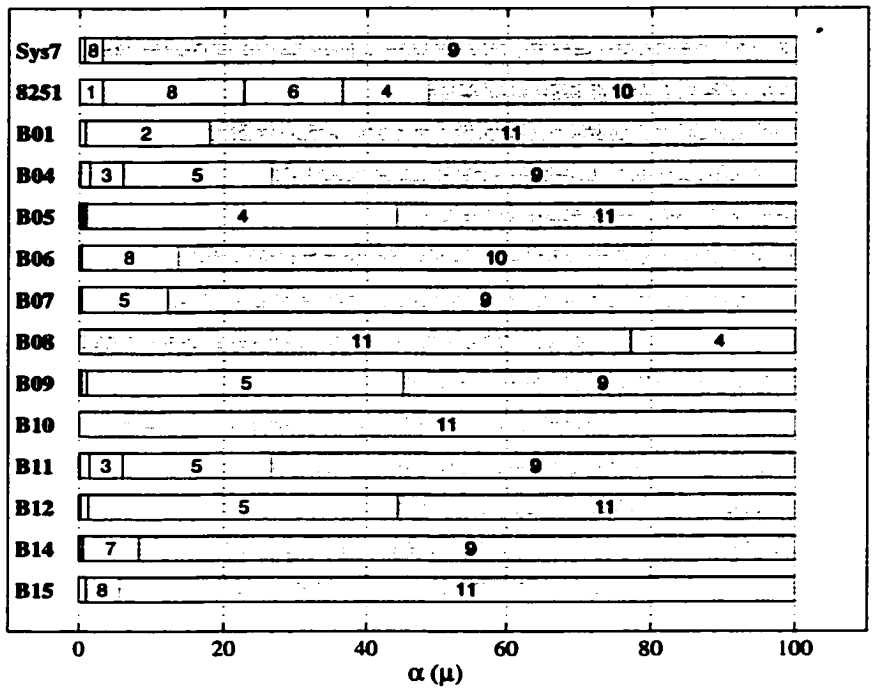


Figure 7.4: Intervals of High Figure of Merit Values (80k Setup)

rest of the phases to catch the remaining branches more effectively, resulting in fewer test vectors and higher branch coverage. The SB and the CDB stopping rules have characteristics similar to those of the DB stopping rule. Figures 7.6 and 7.7 show the figure of merit lines of the models.

7.2 Comparative Study II

Scaling down the verification costs and conducting the 800k setup show how different stopping rules behave when used in different test environments. Table 7.2 summarizes the results of the total number of covered branches and the total number of test vectors applied using a given stopping rule for a benchmark model. Appendix B contains all detailed results of this experiment.

In the 800k setup, the cost parameters used by the stopping rules were proportionally scaled as discussed in Chapter 6. For all but two benchmark models in this

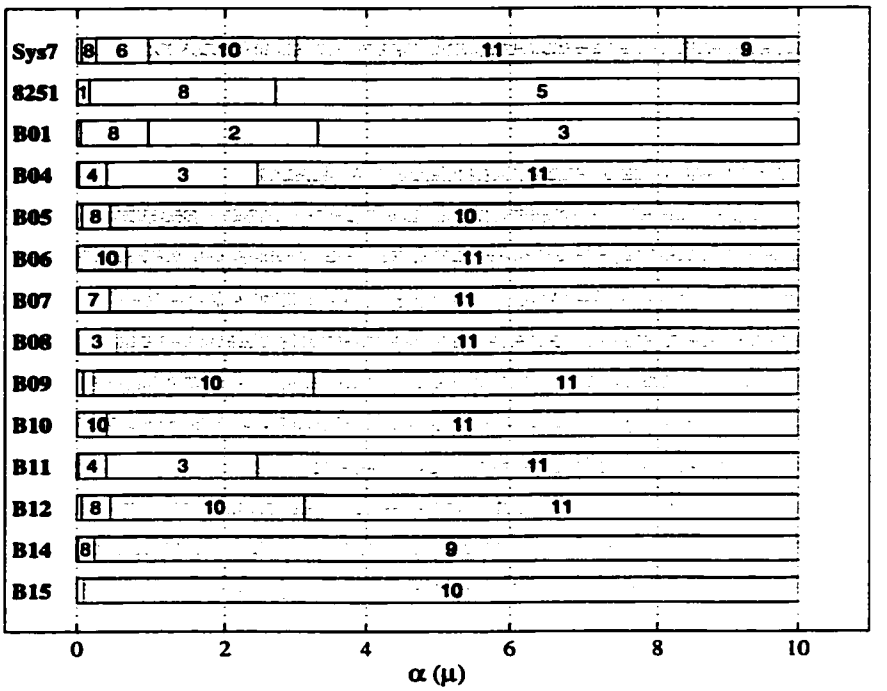


Figure 7.5: Intervals of High Figure of Merit Values (800k Setup)

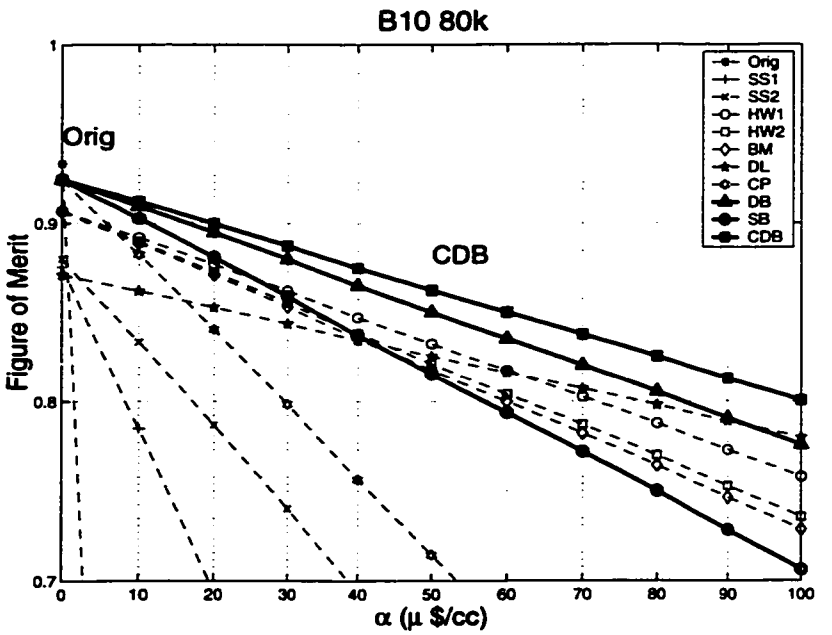


Figure 7.6: B10: Figure of Merit Lines

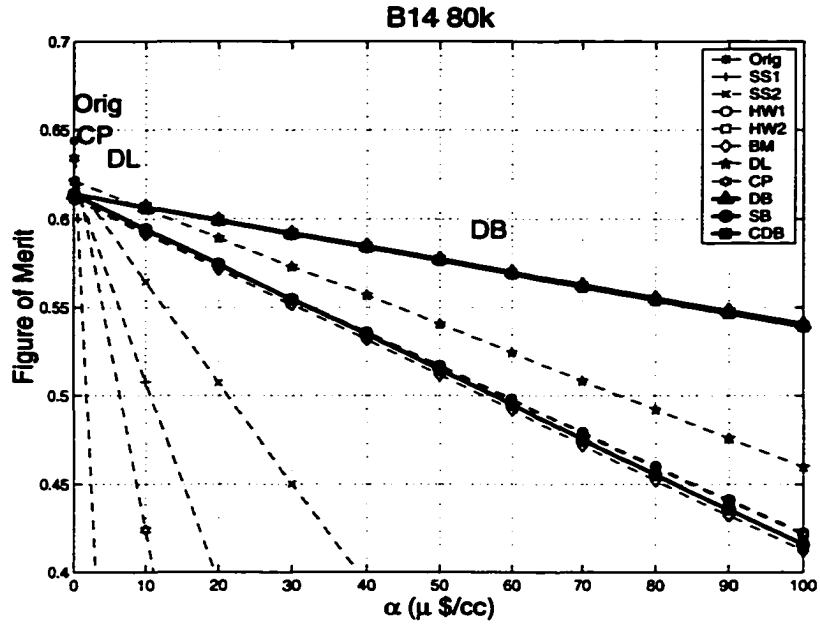


Figure 7.7: B14: Figure of Merit Lines

Model	Orig	SS1	SS2	HW1	HW2	BM	DL	CP	SB	DB	CDB
Sys7	578	539	540	541	557	557	576	561	545	539	545
	540283	3350	5230	8451	22842	19500	540283	45283	5640	3196	4406
8251	184	92	127	103	132	117	75	158	95	87	87
	800K	5890	28214	16965	23988	25399	6219	70500	8769	6242	6419
B01	219	186	175	186	191	191	213	205	147	135	137
	800K	19229	10274	19399	33346	35225	800K	72000	7549	4396	4488
B04	227	223	222	223	223	223	221	227	216	218	218
	800K	22857	12129	21908	800K	24286	20645	800K	7574	5875	5620
B05	274	251	252	252	252	252	267	259	252	251	251
	800K	25110	16206	17679	18670	19861	800K	57000	6182	6351	6340
B06	211	202	204	207	209	207	209	210	209	207	207
	800K	18672	11360	15770	48076	19314	60578	800K	20860	8501	7837
B07	211	202	204	207	210	208	210	210	205	207	207
	800K	18672	10536	15569	800K	20941	37130	800K	7380	8359	7679
B08	277	271	274	273	273	273	272	275	273	273	273
	800K	28116	12600	17508	27772	20764	11633	800K	20432	6601	6399
B09	274	251	252	252	252	252	266	255	252	251	251
	800K	25110	16206	17345	18340	19527	800K	51000	7451	6456	6428
B10	211	204	208	209	210	209	210	210	210	209	209
	800K	19404	11254	14929	800K	17923	32670	800K	19710	9981	9149
B11	227	223	222	223	223	223	221	227	216	218	218
	800K	22857	12129	21908	409K	24286	20645	800K	7574	5875	5620
B12	274	251	252	252	252	252	267	259	252	251	251
	800K	25110	16206	17679	18670	19861	800K	57000	7352	6298	6285
B14	261	248	248	248	248	248	248	257	248	248	248
	800K	26036	12082	19931	20001	20929	13371	99000	7796	6476	6688
B15	443	364	363	381	381	381	445	409	409	383	383
	800K	34718	17385	23988	25653	27118	800K	54000	18413	14817	13017

Table 7.2: Stopping Rules' Final Results for the 800k Setup

setup, at least one of the proposed stopping rules was a leading candidate for a wide range of α values. However, the proposed stopping rules did not perform well for the B01 benchmark model in this setup. If the α value range is extended as shown in Figure 7.8, the proposed stopping rules outperform all the existing stopping rules. By extending the α value, the proposed stopping rules had better performance for B01 with the same testing cost parameters as those in the 80k setup.

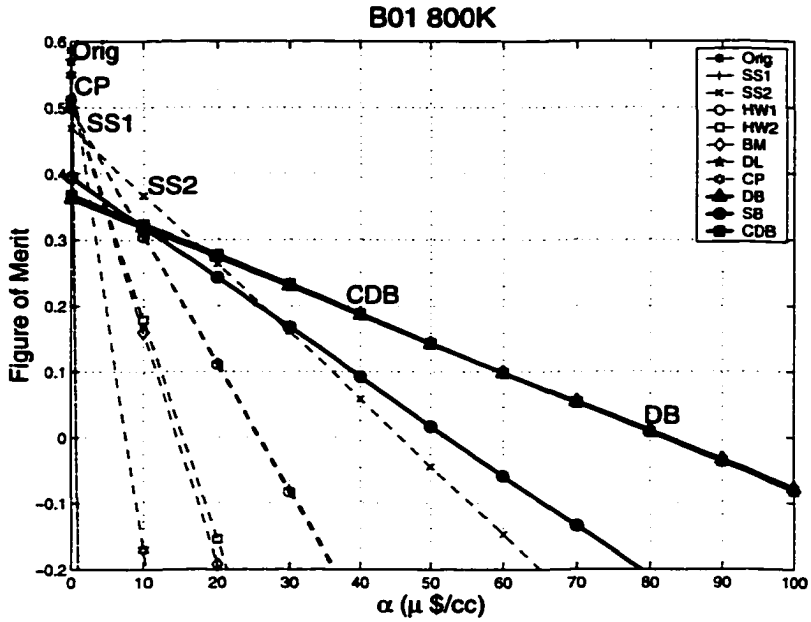


Figure 7.8: B01: Extended Figure of Merit Lines

The Intel 8251 benchmark is the other model where the proposed stopping rules did not perform well. The figure of merit lines of the 8251 benchmark are shown in Figure 7.9. The HW2 stopping rule yielded the highest fm values for a wide range of α values. To check the stability of coverage behavior of the Intel 8251 benchmark, we conducted the 800k setup experiment using different seed values in each of the random phases. Originally, the seeds were 1, 2, 4, and 6 for phases 2 to 5, respectively; now they are set (randomly) to 73, 38, 26, and 92, respectively. Results of this experiment are shown in Figure 7.10.

Branches of the 8251 model are located in nine levels of the control hierarchy (the control flow depth = 9), which makes it very difficult for the testing process to

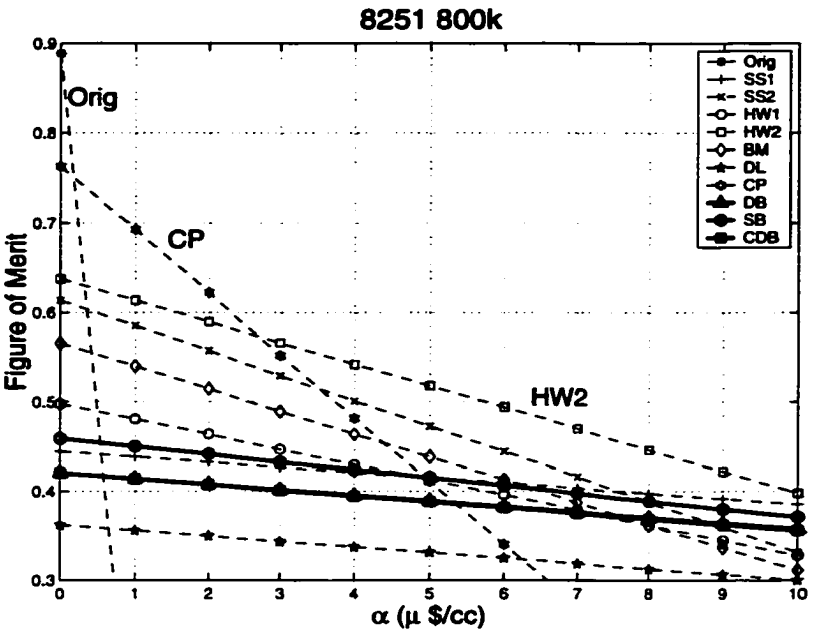


Figure 7.9: Intel 8251: Figure of Merit Lines

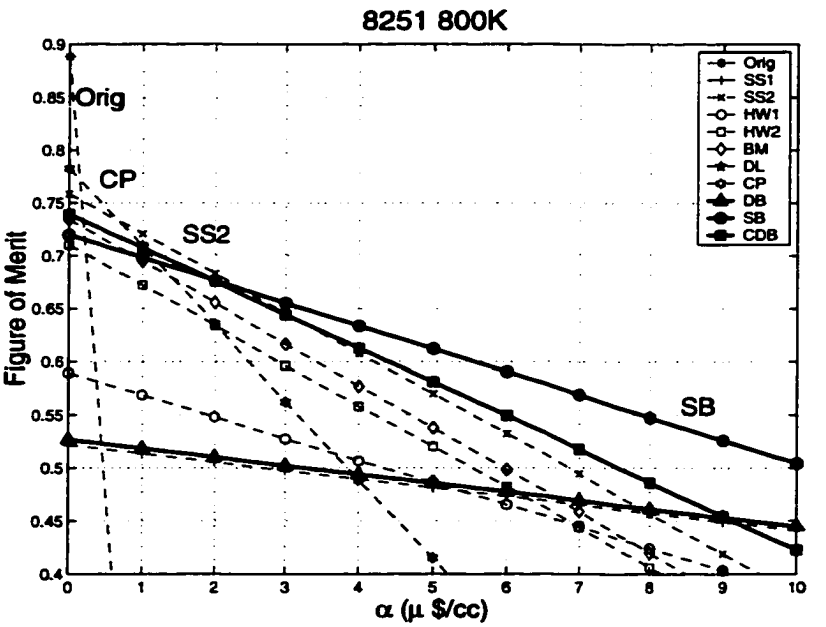


Figure 7.10: Intel 8251: Figure of Merit Lines for the New Setup

cover the branches in the lower levels. The experimental results of the new setup indicate that covering the branches in the lower levels are very sensitive to the testing strategy used. Thus directing the verification process to a group of branches clumped deep in some regions of the control flow graph needs more information about the benchmark structure besides the history of the testing process; our research was based on a black-box technique, which doesn't require internal information about the unit-under-test. We also noticed the unstable behavior of the stopping rules' efficiencies in this case when changing the random pattern seeds in regard to coverage for this specific benchmark. For instant, the SB stopping rule yielded low fm values in the original 800k setup experiment, whereas the fm values of the SB at the second run are among the highest.

There is no strong statistical reason why this is the case for the 8251 model because the science of probability was based on chance; however, one can draw accurate conclusions from the statistical experiments by averaging a large sample of the data and developing a model to fit the majority of the cases.

The observations we made in Section 7.1 about the different behaviors between the proposed stopping rules and the existing stopping rules during the early testing phases can also be made for the experiments based on the 800k setup. This is one of the contributions to the fact that the proposed stopping rules are better than the existing stopping rules in most of the cases.

From the analysis of comparative studies, we conclude that the proposed stopping rules yielded the highest fm values for the vast majority of the behavioral models in wide ranges of α values and directed the verification process to the optimum switching points. In the next section, we discuss the potential of all proposed stopping rules and compare their results with the two proposed setups.

7.3 Comparison of Bayesian-Based Stopping Rules

The three proposed stopping rules are all based on the Bayesian Logarithmically-dependent Poisson behavior discussed in Chapter 4. All of the three stopping rules incorporate the cost-based stopping criterion in making the switching/stopping decisions. However, differences in the stopping criteria and in the parameter choices of the rules make the three stopping rules perform differently.

The DB stopping rule takes advantage of determining its ζ_t parameter automatically based on the verification history yields. The evaluation of ζ_t is dynamically made according to Equation (5.30). In order to show the efficiency of the Bayesian statistical assumptions alone without the improvement of setting the stopping rule's parameter automatically, we set the ζ value to a constant and formed the SB stopping rule, which determines the stopping points based only on the expected Bayesian behavior of coverage. The ζ value is set according to Equation (5.28).

The Confidence-Based Dynamic Bayesian stopping rule, CDB, takes advantage of the confidence-based stopping criterion merged with the cost-based criterion in making the stopping decision. The resulted merged criterion saves some of the testing times where the stopping rule is confident that there will be no coverage. For that purpose, we raised the 'd' values of the cost-based criterion 20% more than that of the DB stopping rule's values to allow the confidence-based criterion to decide whether the testing time after the suggested stopping point by the cost-based criterion is worth simulating or not. If the confidence that there will not be coverage during this time is below a threshold confidence level, C_0, the stopping rule will decide to stop earlier than the DB would suggest to stop. Thus, the CDB stopping rule is expected to perform better than, or at least the same as, the DB stopping rule unless there is a coverage increase in the interval of confidence. Table 7.3 shows the number of phases the CDB stopping rule does not underperform the DB stopping rule for all

the benchmarks. On the average, 60% of the time the CDB performs better than the DB in the 80k setup. The CDB outperforms the DB 64% of the time when the verification environment costs are decreased to the 800k setup costs, which means that the confidence-based criterion along with the cost-based criterion is more suitable to the verification processes of low testing costs and involving large number of test cases.

Setup	Sys7	8251	B01	B04	B05	B06	B07	B08	B09	B10	B11	B12	B14	B15	%
80k	2	1	2	2	3	4	1	4	3	2	2	3	2	4	60%
800k	2	1	1	4	3	2	2	3	4	2	4	3	3	3	64%

Table 7.3: Number of Phases CDB Outperforms DB

7.4 Comparison Using the Degree of Inefficiency (DOI)

As discussed earlier, the DOI(x) basically is the average loss of figure of merit value for a given stopping rule x to the highest figure of merit value. Thus if a stopping rule x outperforms all the other stopping rules in terms of the figure of merit values for all the values of α, then the DOI(x) = 0 indicating that it is the most efficient stopping rule applied to the given benchmark. Table 7.4 lists the stopping rules in the ascending order according to their averaged DOI values across the benchmarks. The CDB stopping rule is expected to outperform other stopping rules with the least DOI values of 10.12 in the 80k setup, and 19.43 in the 800k-setup. The second in the rank for both setups is the DB stopping rule with DOI values of 10.52 and 21.63. As expected, the third best stopping rule in both setups is the SB stopping rule. This further confirms the fact that the proposed stopping rules outperform all other stopping rules when applied in behavioral model verification. Appendix E shows the detailed values of DOI for different stopping rules and each benchmark.

In all cases, the original run without applying any stopping rule performs the worse, which indicates the importance of using stopping rules in the verification processes. The next best stopping rule to the proposed models is the HW1 in the 80k

80K Setup			800K Setup		
Order	**Stopping Rule**	**DOI**	**Order**	**Stopping Rule**	**DOI**
1	CDB	10.12	1	CDB	19.43
2	DB	10.52	2	DB	21.63
3	SB	24.06	3	SB	30.99
4	HW1	34.31	4	SS2	35.07
5	BM	39.98	5	HW1	56.28
6	HW2	70.20	6	BM	71.77
7	SS2	200.96	7	SS1	85.32
8	CP	379.72	8	HW2	1047.72
9	SS1	413.58	9	DL	1640.45
10	DL	447.53	10	CP	1822.37
11	Orig	3769.64	11	Orig	3802.60

Table 7.4: Sorted Rules for the Lowest DOI

setup environment and the SS2 in the 800k setup environment. However, the DOI values of both are about 3 times the DOI values of the proposed CDB stopping rule.

7.5 Forecasting Accuracy

The importance of forecasting coverage in behavioral model verification is that the testing process can be directed to a better strategy, or it can also be stopped. The stopping rules reviewed and proposed earlier make use of the estimated coverage only at the next simulation cycle. In this study, we give a prediction further in the future and show the accuracy of our proposed statistical model in predicting coverage into the future during the verification process. We conducted the two experiments discussed in Chapter 6. In the first experiment, we first extracted the exact probabilities of having coverage within the next Z simulating cycles given that the testing process stopped at a given time T for $T = [10, 20, \ldots, 100] \times 10^3$, and $Z = [1000, 2000, \ldots, 10,000]$. Then, we calculated the estimated probabilities based on the proposed model according to Equation (5.41) derived in Chapter 5 and denoted as $\mathcal{P}(T, Z)$. The second experiment included calculating the exact waiting times to reach coverage for the same T and

Z intervals as in the first experiment. The estimated waiting times were calculated based on Equation (5.45) and denoted as $EWT(T, Z)$.

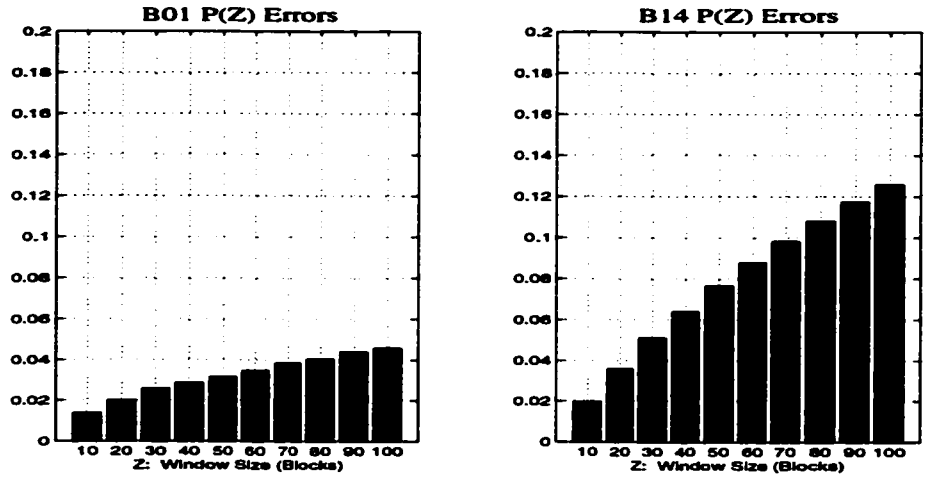


Figure 7.11: Best/Worst Case Errors $\mathcal{P}(Z)$

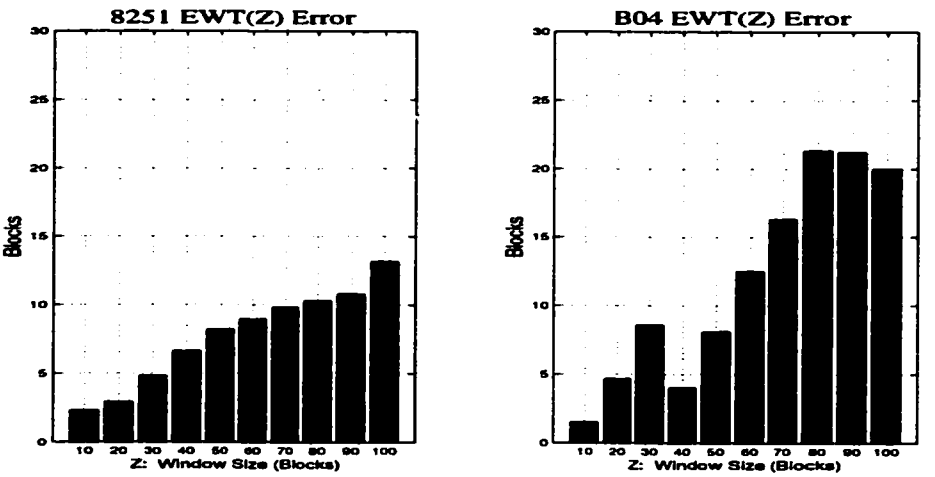


Figure 7.12: Best/Worst Case Errors $EWT(Z)$

The prediction errors of both quantities, the $\mathcal{P}(T, Z)$ and the $EWT(T, Z)$, were estimated for each benchmark model and averaged out across the different verification stop times, T; these averaged quantities are referred to as $\mathcal{P}(Z)$ and $EWT(Z)$. Figure 7.11 shows the best/worst case prediction errors in the $\mathcal{P}(Z)$ values; the x-axis represents the different prediction window sizes, Z, in blocks, and the y-axis

represents the percentage prediction errors. When the prediction window size Z is set to 1000 simulation cycles, the prediction errors across the different benchmark models range from 0.83% to 1.96%. Thus, the prediction accuracy of the proposed forecasting model for $\mathcal{P}(T, Z)$ is at least 98% when the prediction window size is 1000 simulation cycles. When expanding the prediction window size to 10,000 simulation cycles (Z=100 blocks), the prediction errors range between 4.58% and 12.59%, which reduce the prediction accuracy to at least 87%. Appendices F and G show the exact and estimated probabilities and waiting times, respectively.

Similar observations were found regarding the prediction accuracy of the expected waiting times to coverage, $EWT(Z)$. Figure 7.12 shows the best/worst case prediction errors for $EWT(Z)$; the prediction errors range from $\pm$134 to $\pm$299 simulation cycles when the prediction window size is set to 1000 simulation cycles. For Z=10,000 simulation cycles, the prediction errors of $EWT(T, Z)$ range from $\pm$1237 to $\pm$ 2154 simulation cycles. Figures 7.13 and 7.14 illustrate the overall errors of the $\mathcal{P}(Z)$ and the $EWT(Z)$ across the benchmarks.

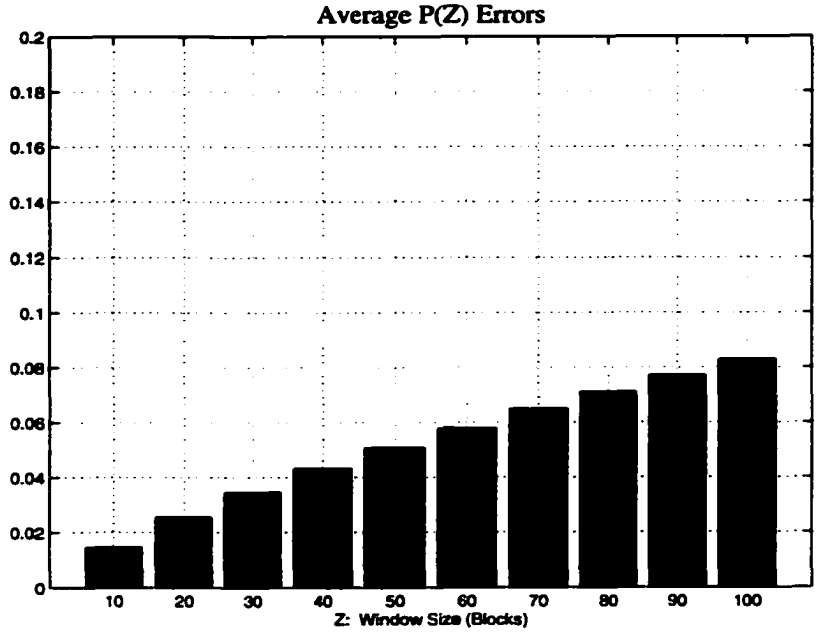


Figure 7.13: Overall Prediction Errors $\mathcal{P}(Z)$

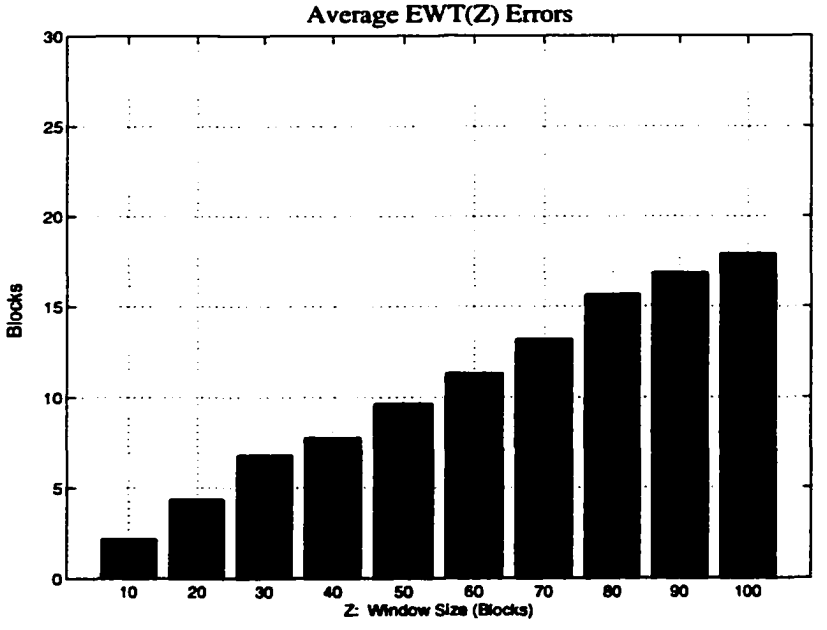


Figure 7.14: Overall Prediction Errors $EWT(Z)$

Chapter 8

CONCLUSION

In this research, we discussed the need and detailed steps in developing statistical stopping rules and a forecasting model for branch coverage in behavioral VHDL model verification. The proposed stopping rules address the issue of when to stop a given testing strategy during the behavioral model verification process and when to switch to a different testing technique. Unlike the existing stopping rules, which mainly suffer from inefficiency and unproven assumptions about behavioral models, our stopping rules use statistical assumptions that are based on the correlation function which best describes the statistical behavior of behavioral models during verification. The research presented here involves:

- Preparing the benchmark models to extract statistical information about coverage suitable for the distribution harvesting experiment and the correlation shape experiment.

- Determining the best shape function that describes the correlation of the interruptions based on the smoothest and the most stable curve across different benchmarks.

- Performing Bayesian analysis which models the history of the testing and incorporates the knowledge of the presumed distribution function and the correlation function for prediction. The expected number of interruptions was estimated

from the history of testing that resembles the logarithmic shape function. Sizes of each expected interruption are estimated based on the Bayesian method of the Poisson process. Our statistical models take advantage of the decreasing property of the logarithmic shaped function for interruptions as well as the beauty of the empirical Bayesian estimation.

- Deriving three versions of the Bayesian stopping rules. The SB stopping rule has a static ζ value to demonstrate the efficiency of the logarithmic correlation function compounded with the Bayesian estimation of the Poisson process in behavioral model verification. The DB stopping rule estimates the ζ parameter automatically based on testing history to further guide the stopping decision to a better stopping point. Finally, the CDB stopping rule takes advantage of both the confidence-based criterion and cost-based criterion in making the stopping decision, which saves unnecessary verification times over the DB yields.

Eight existing stopping rules were collected and adapted to suit behavioral model verification. To compare effectiveness of each stopping rule, a suite of 14 behavioral VHDL benchmarks was collected and experimented with using the existing stopping rules and the proposed stopping rules. In our stopping rules experiments, we designed two different experiment setups to demonstrate the behavior of the proposed stopping rules in different application environments.

Two efficiency evaluating measures, the Figure of Merit function (fm) and the Degree of Inefficiency function (DOI) were developed to objectively determine the efficiency of applying stopping rules in a multiphase verification process.

The results show that using the proposed stopping rules would outperform all the other existing stopping rules for wide ranges of design quality and time-to-market tradeoffs represented by the parameter α.

Using the DOI as a measure of efficiency for a given stopping rule, the proposed three stopping rules are up to 3 times better than the efficiency of using the best existing stopping rule and up to two orders of magnitude better than the efficiency without using any stopping rule. Among the proposed stopping rules, the CDB rule performs at least 60% of the time better than the other two.

Finally, we presented the statistical forecasting model that predicts the probability of a future coverage along with the expected waiting time to have that coverage. The prediction errors of the proposed forecasting model in estimating the near future coverage were found insignificant. At the most, less than 2% off the actual probability of having coverage is expected in predicting within the next 1000 simulation cycles of future verification. The error margin in expecting the waiting time until the next interruption in the worst case is ± 300 simulation cycles. When the prediction window size increases to 10,000 simulation cycles, the expected error in predicting the probability of having coverage is at most 13%, but only 5% in the best case. The marginal error in predicting the waiting time to coverage is less than ± 2200 simulation cycles.

We conclude our research by stating that the proposed stopping rules will transform the existing brute-force approaches in behavioral model verification to more intelligent and statistical approaches to meet the tight time-to-market requirement of complex electronic systems without sacrificing the quality of the design.

Although the proposed stopping rules are proven to outperform the existing stopping rules for the majority of behavioral models, for some models with branches deeply buried in the control flow graph of its behavioral representation, more white-box-type information about behavioral models is needed. The proposed stopping rules describe only the average branch coverage behavior during the verification process, thus, treating the unit-under-test as a black box. Detailed coverage reports such as the coverage progress of specific branches would be more beneficial to direct the

verification process to a better stopping decision. As a future work, we suggest incorporating white-box information into the statistical approach to further increase the effectiveness of the proposed stopping rules for behavioral model verification.

REFERENCES

[1] L. Harrison, P. Saraceni, "Reliability issues for design and test of complex integrated circuits," Digital Avionic System Conference, pp 173-177, November 1995

[2] K. Skahill, "A designer's guide to VHDL design and verification," Electronic Design, pp 149-158, February 1996

[3] http://www.vhdl.org

[4] G. Shaw, A. Anderson, "Executable requirements: opportunities and impediments," IEEE International Conference on Acoustic, Speech, and Signal Processing, pp 1232-1235, May 1996

[5] H. Kumar, S. Erjavic, "Knowledge based testing," International Test Conference, pp 910-917, 1993

[6] R. Bawa, E. Encrenaz, "A tool for translation of VHDL description into a formal model and its application to formal verification and synthesis," Formal Techniques in Real-Time and Fault-Tolerance, pp 471-474, September 1996

[7] M. Morley, "Formal methods for safety in critical systems," Dependable Computing for Critical Applications, pp 37-39, January 1994

[8] M. Bickford, D. Jamsek, "Formal specification and verification of VHDL," International Conference on Computer-Aided Design, pp 310-326, November 1996

[9] D. Appenzeller, A. Kuehlmann, "Formal verification of PowerPC microprocessor," Proceedings of IEEE International Conference on Computer Design, VLSI in Computers and Processors, Austin, TX, 1995

[10] K. Feyerabend, R. Schlor, "Hardware synthesis from requirement specifications," Proceedings of DAC, pp 496-501, September 1996

[11] E. Faucheir, C. Robach, P. Wodey, "Impact of the VHDL description on the testability of integrated systems," Quality Engineering, v8, n4, pp 623-633, 1996

[12] J. Bormann, F. Lohse, M. Payer, G. Venzl, "Model checking in industrial hardware design," Design Automation Conference, San Francisco, California, June, 1995

[13] T. Delong, B. Johnson, "A fault injection technique for VHDL behavioral-level models," IEEE Design and Test Computers, pp 24-33, Winter 1996

[14] R. Baranicki, A. Rosinski, "A testable improvement method for VHDL based synthesis," Electron Technology, v28, n4, pp 267-277, 1995

[15] C. Cho, J. Armstrong, "B-algorithm: a behavioral test generation algorithm," International Test Conference, pp 968-979, 1994

[16] R. Ramchandani, D. Thomas, "Behavioral test generation using mixed integer nonlinear programming," International Test conference, pp 958-967, 1994

[17] G. Hayek, C. Robach, "From specification validation to hardware testing: a unified method," International Test Conference, pp 885-893, 1996

[18] Y. Levendel, "Test generation algorithms for computer hardware description languages," IEEE transactions on Computers, v C-31, n 7, pp 577-588, July 1982

[19] A. Paradkar, K. Tai, "Test generation for Boolean expressions," International Symposium on Software Reliability Engineering, pp 106-115, 1995

[20] T. Lin, S. Su, "The S-algorithm: a promising solution for systematic functional test generation," IEEE Transactions on Computer Aided Design, v 4, n 3, pp 250-263, July 1985

[21] J. Schoen, "Performance and fault modeling with VHDL," Englewood Cliffs, N.J., Prentice Hall, 1992

[22] S. Habinc, P. Sinander, "Accelerated verification of digital devices using VHDL," http://www.vhdl.org

[23] J. Lipman, "Covering your HDL chip-design bets," EDN Design Future Magazine, pp 65-70, October 1998

[24] B. Barrera, "Code coverage analysis-essential to a safe design," Electronic Engineering, pp 41-44, November 1998

[25] T. Boyle, M. Abrahams, "Measuring ASIC test coverage," Electronic Product Design, pp 41-42, October 1996

[26] M. Abrahams, S. Riches, "Optimize ASIC test suites using code-coverage analysis," EDN Design Future, pp 149-152, May 1998

[27] B. Dickinson, S. Shaw, "Software techniques applied to VHDL design," New Electronics, issue 9, pp 63-65, May 1995

[28] P. Bose, "Architectural timing verification and test for super scalar processes," 24th International Symposium on Fault-Tolerant Computing, pp 256-265, 1994

[29] S. Surya, P. Bose, and J. Abraham, "Architectural Performance Verification: PowerPC Processors", IEEE International Conference on Computer Design VLSI in Computers and Processors, pp 344-347, October 1994

[30] J. Gately, "Verifying a million gate processor," Integrated System Design, pp 19-24, 1997

[31] S. Gokhale, K. Trivedi, "Log-logistic software reliability growth model," University of California, pp 34-41, 1998

[32] A. Goel, "Software reliability models: assumptions, limitations, and applicability," IEEE transactions on Software Engineering, v SE-11, n 12, pp 1411-1423, December 1985

[33] W. Howden, "Confidence-based reliability and statistical coverage estimation," Proceedings on the International Symposium on Software Reliability Engineering, pp 283-291, November 1997

[34] W. Howden, "Systems testing and statistical test data coverage," Proceedings of COMPSAC IEEE Computer Society Press, pp 500-505, August 1997

[35] S. Chen, S. Mills, "A binary markov process model for random testing," IEEE transactions on Software Engineering, v22, n3, pp 218-223, March 1996

[36] R. Tupuri, J. Abraham, "A novel hierarchical test generation method for processors," 10th International conference on VLSI Design, pp 540-541, January 1997

[37] M. Sahinoglu, "Compound Poisson software reliability model," IEEE transactions on Software Engineering, v18, n7, pp 624-630, July 1992

[38] A. von Mayrhauser, T. Chen, J. Kok, Ch. Anderson, A. Read, A. Hajjar, "On choosing test criteria for behavioral level hardware design verification," IEEE International High Level Design Validation and Test Workshop, Berkeley, CA, 2000

[39] A. Parrish, S. Zweben, "Analysis and refinement of software test data adequacy properties," IEEE transactions on Software Engineering SE-17, v 6, pp 565-581, June 1991

[40] J. Musa, "A theory of software reliability and its application," Software Engineering, SE-1, n 3, pp 312-327, 1975

[41] D. Mills, "On the statistical validation of computer programs," IBM FSD, Report FSC-72-6015, 1972

[42] S. Dalal, C. Mallows, "When should one stop testing software?" Journal of the American Statistical Association, v 83, n 403, pp 872-879, September 1988

[43] M. Shainoglu, A. von Mayrhauser, A. Hajjar, T. Chen Ch. Anderson, "On the efficiency of a compound Poisson stopping rule for mixed strategy testing," IEEE Aerospace Conference, Track 7, Snowmass, Colorado, March 1999

[44] M. Sahinoglu, U. Can, "Alternative parameter estimation methods for the compound Poisson software reliability model with clustered failure data," Software Testing, Verification, and Reliability, v7, pp 35-57, 1997

[45] P. Randolph, M. Sahinoglu, "A stopping rule for a compound Poisson random variable," Applied Stochastic Models and Data Analysis, v11, pp 135-143, 1995

[46] J. Musa, A. Iannino, K. Okumoto, "Software reliability: measurement, prediction, application," McGraw-Hill, New York, 1987

[47] T. Chen, I. Munn, A. von Mayrhauser, A. Hajjar, "Efficient verification of behavioral models using the sequential sampling technique," Very Large Scale Integration, BRAZIL, 1999

[48] T. Chen, M. Sahinoglu, A. von Mayrhauser, A. Hajjar, Ch. Anderson, "Achieving the quality for behavioral models with minimum effort," 1st International Symposium on Quality in Electronic Design, pp , March 2000

[49] T. Chen, M. Sahinoglu, A. von Mayrhauser, A. Hajjar, Ch. Anderson, "How much testing is enough? Applying stopping rules to behavioral model testing," High Assurance Systems Engineering Symposium, Washington, DC, pp 249-256, November 1999

[50] I. Olkin, L. Gleser, C. Derman, Probability Models and Applications, Prentice-Hall, 1994.

[51] http://www.wkap.nl

[52] http://cbl.ncsu.edu

[53] http://www.seva.com

[54] Pittsburgh University Reports, Department of Electrical Engineering 1985

[55] Colorado State University, Department of Electrical Engineering and Computer Science, NFS Reports 1997

[56] http://www.transeda.com

[57] U. Heinkel, W. Glauert, "An approach for a dynamic generation/validation system for the functional simulation considering timing constraints," European Design and Test Conference, pp 302-306, March 1996

[58] H. Yin, Z. Lebne-Dengel, Y. Malaiya, "Automatic test generation using checkpoint encoding and antirandom testing," Proceedings on the International Symposium on Software Reliability Engineering, pp 1-12, 1997

[59] R. Makki, S. Bou-Ghazale, C. Tianshang, "Automatic test pattern generation with branch testing," IEEE transactions on Computers, v40, n6, pp 785-791, June 1991

[60] E. Weyuker, T. Goradia, A. Singh, "Automatically generating test data from a Boolean specification," IEEE transactions on Software Engineering, v20, n5, pp 353-363, May 1994

[61] A. Bai, A. von Mayrhauser, T. Chen, Ch. Anderson, A. Hajjar, "Fast antirandom (FAR) test generation to improve code coverage," 11th International Software Quality Week, Session 4, May 1998

[62] R. Reetz, "A flowgraph semantics of VHDL: toward a VHDL verification workbench in HOL," Formal Methods in System Design, Chapter 7, pp 73-99, 1995

[63] D. Sinclair et.al., "A formal approach to HW/SW co design: the INSYDE project," IEEE , pp 372-381, 1996

[64] L. Clarke, A. Podgurski, D. Richardson, "A formal evaluation of data flow path selection criteria," IEEE transactions on Software Engineering, v 15, n 11, pp 1318-1332, November 1989

[65] W. Howden, "A functional approach to program testing and analysis," IEEE transactions on Software Engineering, v SE-12, n10, pp 997-1005, October 1986

[66] M. Trachtenberg, "A general theory of software reliability modeling," IEEE transactions on Reliability, v39, n1, pp 92-96, April 1990

[67] J. Whittaker, M. Thomason, "A markov chain model for statistical software testing," IEEE transaction on software Engineering, v20, n10, pp 812-824, October 1994

[68] P. Thevenod-Fosse, H. Waeselynck, "An investigation of statistical software testing," Journal of Software Testing, Verification, and Reliability, v 1, n 2, pp 5-25, July 1991

[69] R. Schlor, "A prover for VHDL-based hardware design," Proceedings of the ASP-Design Automation Conference, pp 643-650, 1995

[70] W. Deason, D. Brown, "A rule-based software test data generation," IEEE transactions on Knowledge and Data Engineering, v3, n1, pp 108-117, March 1991

[71] R. Roy, "Advantage of high level test synthesis over design for test," International Test Conference, p 293, 1995

[72] R. Vermuri, et.al., "An integrated multi-component synthesis environment for MCMs," Computer, pp 62-74, April 1993

[73] J. Moore, "an interactive fuzzy CAD tool," IEEE Micro, pp 68-74, April 1996

[74] E. Bruls, "Analogue fault simulation in standard VHDL," IEE Proceedings on Circuits Devices Systems, v143, n6, pp 380-385, December 1996

[75] Y. Malaiya, "Antirandom testing: getting the most out of black-box testing," Proceedings on the International Symposium on Software Reliability Engineering, pp 86-95, 1995

[76] P. Walsh, D. Hoffman, "Automated behavioral testing of VHDL components," Proceedings on Canadian Conference on Electrical and Computer Engineering, pp 166-169, 1996

[77] A. Wahba, D. Borrione, "Automatic diagnosis may replace simulation for correcting simple design errors," European DAC, pp 476-481, September 1996

[78] P. Vishakantaiah, J. Abraham, M. Abadir, "Automatic test knowledge extraction from VHDL (ATKET)," 29th ACM/IEEE Design Automation Conference, pp 273-278, 1992

[79] C. Shi, N. Godambe, "Behavioral fault modeling and simulation of phase-locked loops using a VHDL-A like language," IEEE International ASIC Conference, pp 245-250, September 1996

[80] F. Ferrandi, F. Fummi, E. Macii, M. Poncino, D. Sciuto, "BDD-based testability estimation of VHDL designs," Proceedings of European Design Automation Conference, pp 444-449, September 1996

[81] C. Ryan, "Circuit partitioning for distributed VHDL fault simulation," IEEE International ASIC Conference, pp 255-258, September 1996

[82] F. Celeiro, L. Dias, J. Ferroira, M. Santos, J. Teixeira, "Defect-oriented IC test and diagnosis using VHDL fault simulation," International Test Conference, pp 620-628, 1996

[83] J. Roy, N. Kumar, R. Dutta, R. Vermuri, "DSS: a distributed high-level synthesis system," IEEE Design and Test of Computers, pp 18-32, 1992

[84] J. Armstrong, G. Frank, F. Gray, "Efficient approach to testing VHDL DSP models," Journal of VLSI Signal Processing, v14, pp 221-234, 1996

[85] G. Doyle, T. Paulson, "Eliminating high-speed gotchas," Printed Circuit Design, pp 1832, 1992

[86] S. Bird, "ESDA - has the bubble burst?" Electronic Product Design, pp 40-43, May 1996

[87] D. Parnas, A. van Schouwen, S. Kwan, "Evaluation of safety-critical software," Communications of the ACM, v 33, n 6, pp 636-648, June 1990

[88] D. Hamlet, J. Voas, "Faults on its sleeve: amplifying software reliability testing," Proceedings on the International Symposium on Software Testing and Analysis, Cambridge, MA, pp 89-98, August 1993

[89] R. Micallef-Trigona, "Formal verification—the new wave in design", Proceedings of the International Conference on Concurrent Engineering and Electronic Design Automation, pp 357-361, 1994

[90] L. Maliniak, "Formal verification copes with complexity problems," Technology Analysis, December 1995

[91] G. Cabodi, P. Camurati, S. Quer, "Full symbolic ATPG for larg circuits," International Test Conference, pp 980-988, 1994

[92] F. Fallah, S. Devadas, K. Keutzer, "Functional vector generation for HDL models using linear programming and 3-satisfiability," 35th Design Automation Conference, June 1998

[93] R. Ferguson, B. Korel, "Generating test data for distributed software using the chaining approach," Information and Software Technology, v38, n5, pp 343-353, 1996

[94] R. Vermuri, "Generation of design verification tests from behavioral VHDL programs using path enumeration and constraint programming," IEEE transactions on Very Large Scale Integration (VLSI) Systems, v3, n3, pp 201-214, June 1995

[95] P. Randolph, "Optimal stopping rules for multinomial observations," Metrika, v 14 , pp 48-61, 1969

[96] L. Ng, C. Jong, "Implementation of synthesized digital systems with VHDL," International Conference on Microelectronics and VLSI, pp 339-342, November 1995

[97] A. Mood, F. Graybill, D. Boes, Introduction to the theory of statistics, McGraw Hill, pp 339-363, 1993

[98] A. Dold, B. Eckmann, Doubly stochastic Poisson processes, Spring-Verlag, 1976

[99] S. Frezza, S. Levitan, P. Chrysanthis, "Linking requirements and design data for automated functional evaluation," Computers in Industry, v30, pp 13-25, 1996

[100] M. Lipman, J. Abrahams, "Minimum average cost testing for partially ordered components," IEEE transactions on Information Theory, v41, n1, pp 287-291, January 1995

[101] J. Grandell, Mixed Poisson processes, Chapman and Hall, 1997

[102] D. Dams, et.al., "Model checking using adaptive state data abstraction," Computer Aided Verification, pp 455-467, June 1994

[103] P. Nandakumar, S. Dater, R. Akella, "Models for measuring and accounting for cost of conformance quality," Management Science, v 39, n 1, pp 1-16, January 1993

[104] J. Yahav, "On optimal stopping," Ann. Mathematical Statistics, v 34, pp 30-35, 1966

[105] G. Hayek, C. Robach, "On the adequacy of deriving hardware test data from the behavioral specification," Proceedings of Hardware and Software Design Strategies, pp 337-342, September 1996

[106] A. Bertolino, L. Strigini, "On the use of testability measures for dependability assessment," IEEE transactions on Software Engineering, v 22, n 2, pp 97-108, February 1996

[107] H. Robbins, "Optimal Stopping," American Mathematicians, v 77, pp 333-343, 1963

[108] K. Trivedi, Probability and statistics with reliability, queuing, and computer science applications, Prentice-Hall NJ, 1982

[109] K. Goossens, "Reasoning about VHDL using operational and observational semantics," Correct Hardware Design and Verification Methods, pp 311-327, 1995

[110] D. Deharbe, D. Borrione, "Semantics of a verification-oriented of VHDL," Correct Hardware Design and Verification Methods, pp 293-310, October 1995

[111] W. Howden, Y. Huang, "Software trustability analysis," ACM Transactions on Software Engineering and Methodology, v 4, n 1, pp 36-64, January 1995

[112] J. Hirase, "Study on the costs of on-site VLSI testing," International Test Conference, paper 20.2, pp 438-443, 1995

[113] F. Korf, "Synthesis of VHDL test environment out of temporal logic specification," 6th International Workshop on High Level Synthesis, pp 354-383, November 1992

[114] J. Jin, "System design verification at the behavioral level," Colorado State University, Electrical Engineering, PhD, 1995

[115] G. Walton, J. Poore, C. Trammell, "Statistical testing of software based on a usage model," Software Practice and Experience, v 25, n 1, pp 97-108, January 1995

[116] B. Littlewood, D. Wright, "Some conservative stopping rules for the operational testing of safety-critical software," IEEE Transactions on Software Engineering, v 23, n 11, pp 673-683, November 1997

[117] P. Sanchez, I. Hidalgo, "System level fault simulation," International Test Conference, paper 27.3, pp 732-740, 1996

[118] C. Papachristou, J. Carletta, "Test synthesis in the behavioral domain," International Test Conference, paper 30.3, pp 693-702, 1995

[119] F. Corno, P. Prinetto, M. Reorda, "Testability analysis and ATPG on behavioral RTL VHDL," International Test Conference, paper 30.4, pp 753-759, 1997

[120] M. Sahinoglu, "The limit of sum of Markov Bernoulli variables in system reliability evaluation," IEEE Transactions on Reliability, v 39, n 1, pp 46-50, April 1990

[121] R. Grinwald, et. al., "User defined coverage - a tool supported methodology for design verification," 35 Design Automation Conference, June 1998

[122] D. Sinclair, E. Holz, D. Witaszek, M. Wasowski, "Validation of hybrid systems by co-simulation," Hybrid Systems, pp 316-326, October 1995

[123] W. Ecker, "Verification methods for VHDL RTL-subroutines," Journal of Systems Architecture, v 42, pp 117-128, 1996

[124] E. Stabler, M. Nassif, R. Paragi, "Verification of ASIC designs in VHDL using computer-aided reasoning," IEEE International ASIC Conference and Exhibit, pp 163-166, 1996

[125] A. Vachoux, "VHDL-A: a future standard for analog and mixed digital-analog description and simulation," IEEE International Symposium on Industrial Electronics, pp 39-44, July 1995

[126] F. Busaba, "VHDL description of self-checking logic circuits," Proceedings on Southeastern Symposium on System Theory, pp 477-481, 1996

[127] A. Hajjar, T. Chen, and A. von Mayrhauser, "On statistical behavior of branch coverage in testing behavioral VHDL models," IEEE International High Level Design Validation and Test Workshop, Berkeley, CA, 2000

[128] W. Hill, and W. Hunter, "A review of Response Surface Methodology: A literature review", Technometrics, v 8, pp. 571-590, 1966

Appendix A

STOPPING RULES EXPERIMENTS (80K SETUP)

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	283	524	283	524	283	524	283	524	283	524	283	524
ϕ_2	35000	29	434	12	476	12	532	12	546	12	1365	14
ϕ_3	4000	2	120	0	120	0	120	0	120	0	120	0
ϕ_4	10000	8	60	0	60	0	60	0	60	0	60	0
ϕ_5	5000	5	30	0	30	0	30	0	30	0	30	0
Totals	54283	568	927	536	969	536	1025	536	1039	536	1858	538

	DL		CP		SB		DB		CDB	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	283	524	283	524	283	524	283	524	283	524
ϕ_2	742	12	1204	14	84	8	70	8	56	8
ϕ_3	120	0	2700	7	204	3	120	3	140	3
ϕ_4	60	0	900	1	60	0	60	0	60	0
ϕ_5	30	0	1200	1	30	0	30	0	30	0
Totals	1235	536	6287	547	661	535	563	535	569	535

Table A.1: Sys7 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	1500	42	1500	42	1500	42	1500	42	1500	42	1500	42
ϕ_2	10000	39	603	25	632	25	709	25	1399	31	1952	31
ϕ_3	20000	46	60	0	60	0	60	0	60	0	60	0
ϕ_4	20000	29	120	0	120	0	584	14	120	0	120	0
ϕ_5	30000	5	486	12	594	12	180	0	180	0	180	0
Totals	81500	161	2769	79	2906	79	3033	81	3259	73	3812	73

	DL		CP		SB		DB		CDB	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	1500	42	1500	42	1500	42	1500	42	1500	42
ϕ_2	3852	33	900	25	77	18	65	10	85	18
ϕ_3	60	0	1200	16	154	6	42	2	82	2
ϕ_4	120	0	1500	13	364	8	452	19	244	5
ϕ_5	180	0	4500	16	180	0	180	0	180	0
Totals	5712	75	9600	112	2275	74	2239	73	2091	67

Table A.2: 8251 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	10000	178	632	122	656	122	731	122	7893	175	3010	140
ϕ_2	20000	7	60	0	60	0	60	0	60	0	60	0
ϕ_3	20000	7	120	0	120	0	120	0	120	0	120	0
ϕ_4	30000	8	198	6	10248	33	300	6	96	2	162	2
Totals	80000	200	1010	128	11084	155	1211	128	8169	177	3352	142

	DL		CP		SB		DB		CDB	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	195	121	900	122	94	121	182	121	187	121
ϕ_2	392	1	1500	5	188	1	72	1	98	1
ϕ_3	120	0	900	4	120	0	120	0	120	0
ϕ_4	504	6	900	4	1452	6	540	6	492	6
Totals	1211	128	4200	135	1854	128	914	128	897	128

Table A.3: B01 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	10000	220	1298	205	1308	205	1397	205	9922	220	4687	218
ϕ_2	20000	1	60	0	60	0	60	0	60	0	60	0
ϕ_3	20000	2	356	1	920	9	920	9	120	0	120	0
ϕ_4	30000	0	180	0	180	0	180	0	180	0	180	0
Totals	80000	223	1894	206	2468	214	2557	214	10282	220	5047	218

	DL		CP		SB		DB		CDB		
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	
ϕ_1	83	177	1200	205	99	183	278	193	266	193	
ϕ_2	11372	42	900	9	60	0	64	6	60	0	
ϕ_3	120	0	1200	2	292	16	152	3	236	9	
ϕ_4	180	0	13800	1	180	0	180	0	180	0	
Totals	11755	219	17100	217	631	199	674	202	742	202	

Table A.4: B04 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	10000	228	1132	228	1143	228	1231	228	10283	228	4902	228
ϕ_2	20000	7	60	0	60	0	60	0	60	0	60	0
ϕ_3	20000	18	720	23	960	23	960	23	272	6	1980	23
ϕ_4	30000	6	180	0	180	0	180	0	180	0	180	0
Totals	80000	259	2092	251	2343	251	2431	251	10795	234	7122	251

	DL		CP		SB		DB		CDB		
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	
ϕ_1	194	223	1200	228	104	214	289	224	276	224	
ϕ_2	60	0	5100	7	148	8	60	0	60	0	
ϕ_3	4884	29	1500	17	376	10	216	9	228	9	
ϕ_4	180	0	3000	1	180	0	180	0	180	0	
Totals	5318	252	10800	253	808	232	745	233	744	233	

Table A.5: B05 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	10000	190	732	183	751	183	831	183	8569	190	4042	188
ϕ_2	20000	16	60	0	60	0	60	0	60	0	60	0
ϕ_3	20000	4	268	9	416	9	368	9	48	1	336	4
ϕ_4	30000	0	180	0	180	0	180	0	48	1	180	0
Totals	80000	210	1240	192	1407	192	1439	192	8725	192	4618	192

	DL		CP		SB		DB		CDB	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	114	183	900	183	101	183	224	183	224	183
ϕ_2	60	0	900	12	60	0	60	0	60	0
ϕ_3	6756	21	1200	1	332	9	244	9	244	9
ϕ_4	180	0	1500	8	180	0	180	0	180	0
Totals	7110	204	4500	204	673	192	708	192	708	192

Table A.6: B06 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	10000	192	698	183	718	183	797	183	8659	192	4128	192
ϕ_2	20000	15	366	12	426	12	464	12	136	3	252	3
ϕ_3	20000	3	120	0	120	0	120	0	120	0	184	2
ϕ_4	30000	0	138	1	2640	9	240	1	48	1	96	1
Totals	80000	210	1322	196	3904	204	1621	196	8963	196	4660	198

	DL		CP		SB		DB		CDB	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	114	183	900	183	101	183	218	183	219	183
ϕ_2	766	12	900	12	176	10	166	10	176	10
ϕ_3	120	0	1200	1	332	2	140	2	156	2
ϕ_4	132	1	1500	8	180	0	180	0	180	0
Totals	1132	196	4500	204	789	195	704	195	731	195

Table A.7: B07 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	10000	268	1032	263	1045	263	1131	263	12087	268	5762	268
ϕ_2	20000	5	60	0	60	0	60	0	60	0	60	0
ϕ_3	20000	1	120	0	120	0	120	0	120	0	120	0
ϕ_4	30000	0	180	0	180	0	972	10	180	0	180	0
Totals	80000	274	1392	263	1405	263	2283	273	12447	268	6122	268

	DL		CP		SB		DB		CDB		
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	
ϕ_1	63	236	1200	263	109	244	239	253	233	253	
ϕ_2	6918	36	1800	8	168	3	60	0	60	0	
ϕ_3	120	0	1200	1	352	9	120	0	120	0	
ϕ_4	1326	1	5400	1	1620	17	1614	20	1416	20	
Totals	8427	273	9600	273	2249	273	2033	273	1829	273	

Table A.8: B08 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	10000	228	1032	226	1045	226	1338	227	10283	228	4902	228
ϕ_2	20000	7	60	0	60	0	60	0	60	0	60	0
ϕ_3	20000	18	240	8	768	25	892	24	272	6	1980	23
ϕ_4	30000	7	180	0	180	0	180	0	180	0	180	0
Totals	80000	260	1512	234	2053	251	2470	251	10795	234	7122	251

	DL		CP		SB		DB		CDB		
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	
ϕ_1	200	223	1500	227	105	214	278	224	267	224	
ϕ_2	60	0	2100	1	148	8	60	0	60	0	
ϕ_3	4884	29	1200	23	376	10	216	9	228	9	
ϕ_4	180	0	3000	2	180	0	180	0	180	0	
Totals	5324	252	7800	253	809	232	734	233	735	233	

Table A.9: B09 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	10000	196	732	183	751	183	831	183	8840	196	4128	192
ϕ_2	20000	11	366	12	426	12	464	12	60	0	252	3
ϕ_3	20000	3	120	0	120	0	120	0	120	0	184	2
ϕ_4	30000	0	270	9	414	9	366	9	48	1	96	1
Totals	80000	210	1488	204	1711	204	1781	204	9068	197	4660	198

	DL		CP		SB		DB		CDB	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	113	183	900	183	101	183	224	183	224	183
ϕ_2	568	12	900	12	170	11	178	11	186	11
ϕ_3	120	0	1200	2	332	1	96	1	116	1
ϕ_4	114	1	1200	11	1578	13	990	13	714	13
Totals	915	196	4200	208	2181	208	1488	208	1240	208

Table A.10: B10 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	10000	220	1298	205	1308	205	1397	205	9922	220	4687	218
ϕ_2	20000	1	60	0	60	0	60	0	60	0	60	0
ϕ_3	20000	2	356	1	920	9	920	9	120	0	120	0
ϕ_4	30000	0	180	0	180	0	180	0	180	0	180	0
Totals	80000	223	1894	206	2468	214	2557	214	10282	220	5047	218

	DL		CP		SB		DB		CDB	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	83	177	1200	205	99	183	278	193	266	193
ϕ_2	11372	42	900	9	60	0	64	6	60	0
ϕ_3	120	0	1200	2	292	16	152	3	236	9
ϕ_4	180	0	13800	1	180	0	180	0	180	0
Totals	11755	219	17100	217	631	199	674	202	742	202

Table A.11: B11 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	10000	228	1065	226	1077	226	1330	227	10283	228	4902	228
ϕ_2	20000	7	60	0	60	0	60	0	60	0	60	0
ϕ_3	20000	18	240	8	768	25	892	24	272	6	1980	23
ϕ_4	30000	6	180	0	180	0	180	0	180	0	180	0
Totals	80000	259	1545	234	2085	251	2462	251	10795	234	7122	251

	DL		CP		SB		DB		CDB		
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	
ϕ_1	194	223	1500	227	104	214	289	224	276	224	
ϕ_2	60	0	1200	1	148	8	60	0	60	0	
ϕ_3	4884	29	1200	23	376	10	216	9	228	9	
ϕ_4	180	0	3000	2	180	0	180	0	180	0	
Totals	5318	252	6900	253	808	232	745	233	744	233	

Table A.12: B12 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	10000	245	1532	244	1540	244	1631	244	11049	245	5268	245
ϕ_2	20000	8	60	0	60	0	60	0	60	0	60	0
ϕ_3	20000	0	120	0	120	0	120	0	120	0	120	0
ϕ_4	30000	4	180	0	180	0	180	0	138	3	264	3
Totals	80000	257	1892	244	1900	244	1991	244	11367	248	5712	248

	DL		CP		SB		DB		CDB		
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	
ϕ_1	520	242	900	242	100	212	357	239	330	239	
ϕ_2	738	3	900	3	60	0	118	5	134	5	
ϕ_3	120	0	1800	3	376	27	80	1	104	1	
ϕ_4	240	3	17400	5	1446	6	180	0	180	0	
Totals	1618	248	21000	253	1982	245	735	245	748	245	

Table A.13: B14 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	10000	351	1532	350	1540	350	1631	350	15830	351	7546	351
ϕ_2	20000	13	60	0	60	0	60	0	60	0	60	0
ϕ_3	20000	34	120	0	120	0	120	0	120	0	120	0
ϕ_4	30000	17	180	0	180	0	180	0	180	0	180	0
Totals	80000	415	1892	350	1900	350	1991	350	16190	351	7906	351

	DL		CP		SB		DB		CDB	
Phase	CC	Δ	CC	Δ	CC	Δ	C€	Δ	CC	Δ
ϕ_1	146	338	900	340	112	336	400	340	376	340
ϕ_2	60	0	1500	20	60	0	60	0	60	0
ϕ_3	79796	80	3600	6	324	10	248	8	248	8
ϕ_4	0	0	3000	17	1584	18	1590	16	1326	16
Totals	80002	418	9000	383	2080	364	2298	364	2010	364

Table A.14: B15 Model: Stopping Rules Comparisons

Appendix B

STOPPING RULES EXPERIMENTS (800K SETUP)

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	283	524	283	524	283	524	283	524	283	524	283	524
ϕ_2	350000	41	5334	16	5635	16	6335	16	1267	14	2975	15
ϕ_3	40000	5	1200	0	5244	7	1200	0	1200	0	1200	0
ϕ_4	100000	7	1334	1	11380	10	11382	17	300	1	472	1
ϕ_5	50000	1	300	0	300	0	300	0	300	0	300	0
Totals	540283	578	8451	541	22842	557	19500	557	3350	539	5230	540

	DL		CP		SB		DB		CDB	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	283	524	283	524	283	524	283	524	283	524
ϕ_2	2114	14	9002	16	735	12	819	14	945	14
ϕ_3	1200	0	15000	10	1200	0	1200	0	2024	4
ϕ_4	536686	42	12000	8	3122	9	594	1	854	3
ϕ_5	0	0	9000	3	300	0	300	0	300	0
Totals	540283	580	45285	561	5640	545	3196	539	4406	545

Table B.1: Sys7 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	1500	42	1500	42	1500	42	1500	42	1500	42	1500	42
ϕ_2	100000	98	7331	34	7518	34	10137	39	2986	33	4196	33
ϕ_3	200000	26	4998	25	5330	25	8432	29	362	4	19518	52
ϕ_4	200000	15	1336	2	7840	31	2996	5	316	5	1200	0
ϕ_5	300000	3	1800	0	1800	0	2334	2	726	8	1800	0
Totals	801500	184	16965	103	23988	132	25399	117	5890	92	28214	127

	DL		CP		SB		DB		CDB	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	1500	42	1500	42	1500	42	1500	42	1500	42
ϕ_2	4719	33	15000	40	727	25	1068	31	1177	31
ϕ_3	0	0	15000	45	1538	20	898	10	994	8
ϕ_4	0	0	27000	30	3204	8	976	4	1176	4
ϕ_5	0	0	12000	1	1800	0	1800	0	1572	2
Totals	6219	75	70500	158	8769	95	6242	87	6419	87

Table B.2: 8251 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	100000	211	15331	182	15406	182	16329	182	16465	182	7202	171
ϕ_2	200000	6	600	0	600	0	600	0	600	0	600	0
ϕ_3	200000	2	1668	4	15540	9	16496	9	364	4	672	4
ϕ_4	300000	0	1800	0	1800	0	1800	0	1800	0	1800	0
Totals	800000	219	19399	186	33346	191	35225	191	19229	186	10274	175

	DL		CP		SB		DB		CDB	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	225	121	33000	195	749	122	1274	122	1388	126
ϕ_2	492	1	15000	3	1964	15	692	5	692	3
ϕ_3	799284	91	12000	5	3036	10	828	4	1028	4
ϕ_4	0	0	12000	2	1800	0	1602	4	1380	4
Totals	800001	213	72000	205	7549	147	4396	135	4488	137

Table B.3: B01 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	100000	225	17664	221	17728	221	18662	221	19993	221	9265	220
ϕ_2	200000	0	600	0	600	0	600	0	600	0	600	0
ϕ_3	200000	0	1844	2	781672	2	3224	2	464	2	464	2
ϕ_4	300000	2	1800	0	0	0	1800	0	1800	0	1800	0
Totals	800000	227	21908	223	800000	223	24286	223	22857	223	12129	222

	DL		CP		SB		DB		CDB	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	241	192	12000	220	758	205	2671	216	2468	216
ϕ_2	14212	27	12000	1	1668	9	600	0	600	0
ϕ_3	4392	2	9000	2	3348	2	804	2	752	2
ϕ_4	1800	0	767004	4	1800	0	1800	0	1800	0
Totals	20645	221	800004	227	7574	216	5875	218	5620	218

Table B.4: B04 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	100000	228	11331	228	11438	228	12329	228	20626	228	9602	228
ϕ_2	200000	25	600	0	600	0	600	0	600	0	600	0
ϕ_3	200000	14	3948	24	4832	24	5132	24	2084	23	4204	24
ϕ_4	300000	7	1800	0	1800	0	1800	0	1800	0	1800	0
Totals	800000	274	17679	252	18670	252	19861	252	25110	251	16206	252

	DL		CP		SB		DB		CDB	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	223	223	9000	228	766	227	2227	228	2192	228
ϕ_2	799778	44	15000	7	600	0	600	0	600	0
ϕ_3	0	0	12000	18	3016	25	1724	23	1748	23
ϕ_4	0	0	21000	6	1800	0	1800	0	1800	0
Totals	800001	267	57000	259	6182	252	6351	251	6340	251

Table B.5: B05 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	100000	198	8664	190	8814	190	9662	190	17370	192	8002	190
ϕ_2	200000	11	3332	11	4078	11	4330	11	300	3	850	5
ϕ_3	200000	2	1200	0	1200	0	1200	0	456	1	456	1
ϕ_4	300000	0	2574	6	33984	8	4122	6	546	6	2052	8
Totals	800000	211	15770	207	48076	209	19314	207	18672	202	11360	204

	DL		CP		SB		DB		CDB		
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	
ϕ_1	130	183	12000	190	760	183	1497	183	1581	183	
ϕ_2	21082	23	12000	16	1660	13	1384	13	1498	13	
ϕ_3	1200	0	15000	4	3344	7	796	1	744	1	
ϕ_4	38166	3	761004	0	15096	6	4824	10	4014	10	
Totals	60578	209	800004	210	20860	209	8501	207	7837	207	

Table B.6: B06 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	100000	203	8997	192	9140	192	9995	192	17370	192	8086	192
ϕ_2	200000	7	2798	9	5632	10	5636	10	300	3	512	3
ϕ_3	200000	1	1200	0	1200	0	1200	0	456	1	456	1
ϕ_4	300000	0	2574	6	784032	8	4110	6	546	6	1482	8
Totals	800000	211	15569	207	800004	210	20941	208	18672	202	10536	204

	DL		CP		SB		DB		CDB		
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	
ϕ_1	130	183	9000	192	760	183	1431	183	1525	183	
ϕ_2	928	12	12000	15	1476	13	1308	13	1396	13	
ϕ_3	34272	15	15000	3	3344	9	796	1	744	1	
ϕ_4	1800	0	764004	0	1800	0	4824	10	4014	10	
Totals	37130	210	800004	210	7380	205	8359	207	7679	207	

Table B.7: B07 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	100000	277	11664	268	11768	268	12662	268	24516	271	11286	268
ϕ_2	200000	0	2388	4	13004	5	3410	4	600	0	358	1
ϕ_3	200000	0	1656	1	1200	0	2892	1	1200	0	416	2
ϕ_4	300000	0	1800	0	1800	0	1800	0	1800	0	540	3
Totals	800000	277	17508	273	27772	273	20764	273	28116	271	12600	274

	DL		CP		SB		DB		CDB			
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ		
ϕ_1	71	236	9000	268	774	263	1983	263	1975	263		
ϕ_2	8562	36	9000	5	1670	8	882	6	1082	6		
ϕ_3	1200	0	15000	1	3300	1	760	1	708	1		
ϕ_4	1800	0	767004	1	14688	1	2976	3	2634	3		
Totals	11633	272	800004	275	20432	273	6601	273	6399	273		

Table B.8: B08 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	100000	228	10997	228	11108	228	11995	228	20626	228	9602	228
ϕ_2	200000	20	600	0	600	0	600	0	600	0	600	0
ϕ_3	200000	19	3948	24	4832	24	5132	24	2084	23	4204	24
ϕ_4	300000	7	1800	0	1800	0	1800	0	1800	0	1800	0
Totals	800000	274	17345	252	18340	252	19527	252	25110	251	16206	252

	DL		CP		SB		DB		CDB			
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ		
ϕ_1	229	223	9000	228	767	224	2092	227	2076	227		
ϕ_2	799772	43	18000	7	1868	4	600	0	600	0		
ϕ_3	0	0	12000	18	3016	24	1964	24	1952	24		
ϕ_4	0	0	12000	2	1800	0	1800	0	1800	0		
Totals	800001	266	51000	255	7451	252	6456	251	6428	251		

Table B.9: B09 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	100000	204	9997	196	10122	196	10995	196	17732	196	8086	192
ϕ_2	200000	5	600	0	600	0	600	0	600	0	512	3
ϕ_3	200000	2	2664	9	4156	9	3664	9	436	1	436	1
ϕ_4	300000	0	1668	4	785124	5	2664	4	636	7	2220	12
Totals	800000	211	14929	209	800002	210	17923	209	19404	204	11254	208

	DL		CP		SB		DB		CDB		
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	
ϕ_1	130	183	12000	196	760	183	1497	183	1581	183	
ϕ_2	684	12	9000	10	1470	13	1314	13	1404	13	
ϕ_3	30056	15	15000	4	2912	9	1332	2	1304	2	
ϕ_4	1800	0	764004	0	14568	5	5838	11	4860	11	
Totals	32670	210	800004	210	19710	210	9981	209	9149	209	

Table B.10: B10 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	100000	225	17664	221	17728	221	18662	221	19993	221	9265	220
ϕ_2	200000	0	600	0	600	0	600	0	600	0	600	0
ϕ_3	200000	0	1844	2	390836	2	3224	2	464	2	464	2
ϕ_4	300000	2	1800	0	0	0	1800	0	1800	0	1800	0
Totals	800000	227	21908	223	409164	223	24286	223	22857	223	12129	222

	DL		CP		SB		DB		CDB		
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	
ϕ_1	241	192	12000	220	758	205	2671	216	2468	216	
ϕ_2	14212	27	12000	1	1668	9	600	0	600	0	
ϕ_3	4392	2	9000	2	3348	2	804	2	752	2	
ϕ_4	1800	0	767004	4	1800	0	1800	0	1800	0	
Totals	20645	221	800004	227	7574	216	5875	218	5620	218	

Table B.11: B11 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	100000	228	11331	228	11438	228	12329	228	20626	228	9602	228
ϕ_2	200000	23	600	0	600	0	600	0	600	0	600	0
ϕ_3	200000	16	3948	24	4832	24	5132	24	2084	23	4204	24
ϕ_4	300000	7	1800	0	1800	0	1800	0	1800	0	1800	0
Totals	800000	274	17679	252	18670	252	19861	252	25110	251	16206	252

	DL		CP		SB		DB		CDB	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	223	223	9000	228	766	224	2174	228	2137	228
ϕ_2	799778	44	15000	7	1770	4	600	0	600	0
ϕ_3	0	0	12000	18	3016	24	1724	23	1748	23
ϕ_4	0	0	21000	6	1800	0	1800	0	1800	0
Totals	800001	267	57000	259	7352	252	6298	251	6285	251

Table B.12: B12 Model: Stopping Rules Comparisons

	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	100000	253	16331	248	16401	248	17329	248	22436	248	10318	245
ϕ_2	200000	0	600	0	600	0	600	0	600	0	600	0
ϕ_3	200000	8	1200	0	1200	0	1200	0	1200	0	864	2
ϕ_4	300000	0	1800	0	1800	0	1800	0	1800	0	300	1
Totals	800000	261	19931	248	20001	248	20929	248	26036	248	12082	248

	DL		CP		SB		DB		CDB	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	603	242	9000	245	762	242	2956	244	2808	244
ϕ_2	2928	3	18000	3	1482	3	236	1	692	1
ϕ_3	8040	3	42000	5	3752	3	1484	3	1388	3
ϕ_4	1800	0	30000	4	1800	0	1800	0	1800	0
Totals	13371	248	99000	257	7796	248	6476	248	6688	248

Table B.13: B14 Model: Stopping Rules Comparisons

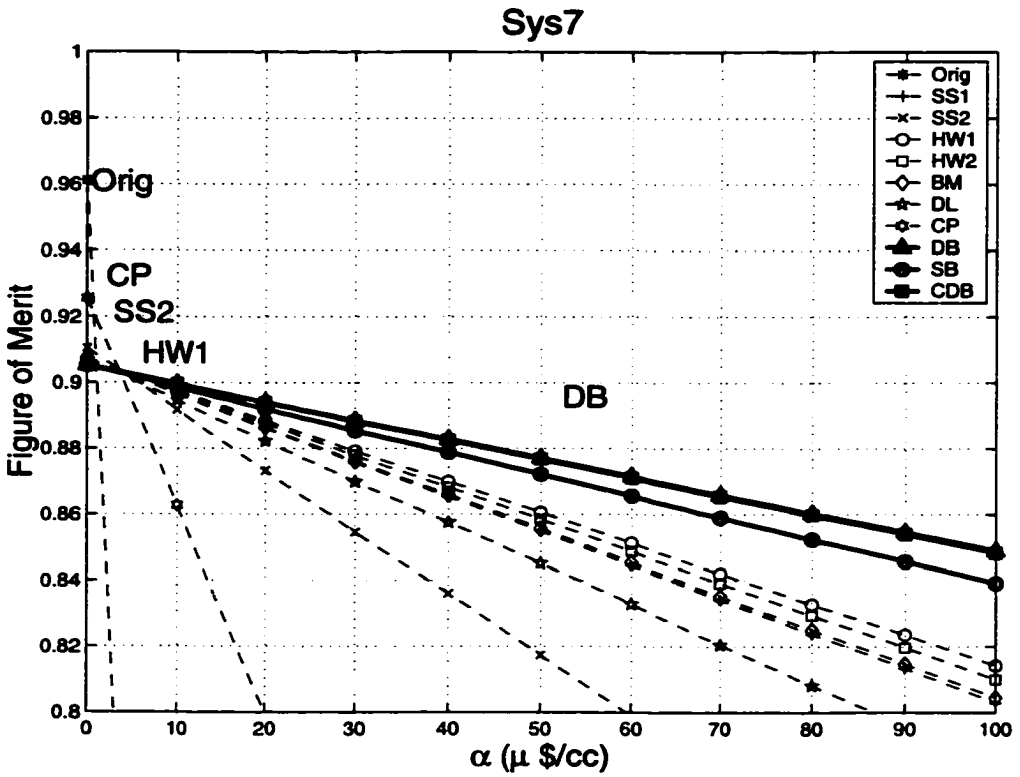
	Orig		HW1		HW2		BM		SS1		SS2	
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ
ϕ_1	100000	352	15664	351	15737	351	16662	351	31844	352	14781	351
ϕ_2	200000	19	3332	9	4078	9	4330	9	600	0	324	1
ϕ_3	200000	59	1200	0	1200	0	1200	0	996	11	1872	10
ϕ_4	300000	13	3792	21	4638	21	4926	21	1278	1	408	1
Totals	800000	443	23988	381	25653	381	27118	381	34718	364	17385	363

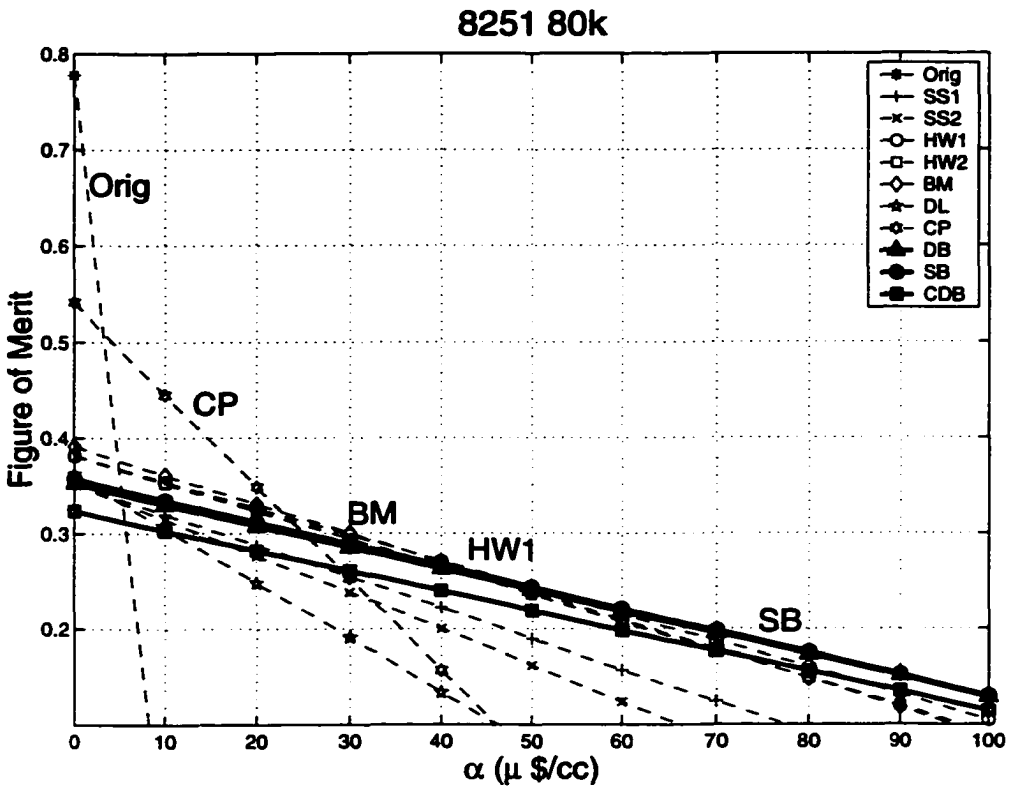
	DL		CP		SB		DB		CDB		
Phase	CC	Δ	CC	Δ	CC	Δ	CC	Δ	CC	Δ	
ϕ_1	167	338	9000	351	779	340	2967	350	2819	350	
ϕ_2	11714	25	12000	12	1578	20	1140	10	1198	10	
ϕ_3	1200	0	12000	20	1200	0	1200	0	1200	0	
ϕ_4	786924	82	21000	26	14856	49	9510	23	7800	23	
Totals	800005	445	54000	409	18413	409	14817	383	13017	383	

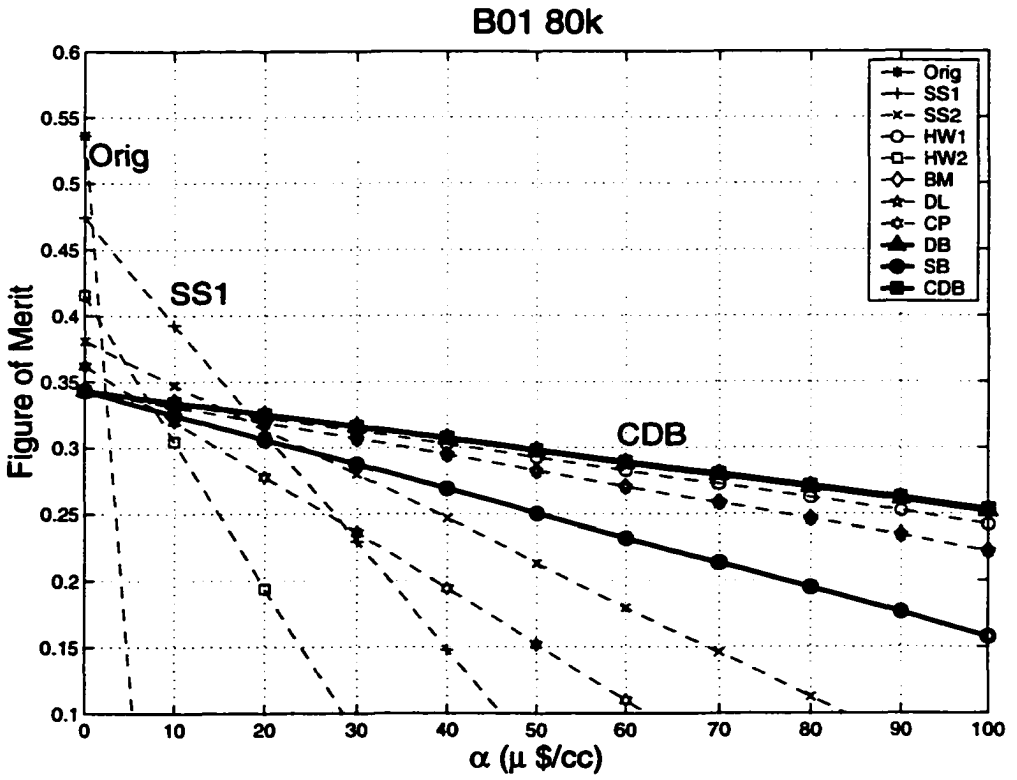
Table B.14: B15 Model: Stopping Rules Comparisons

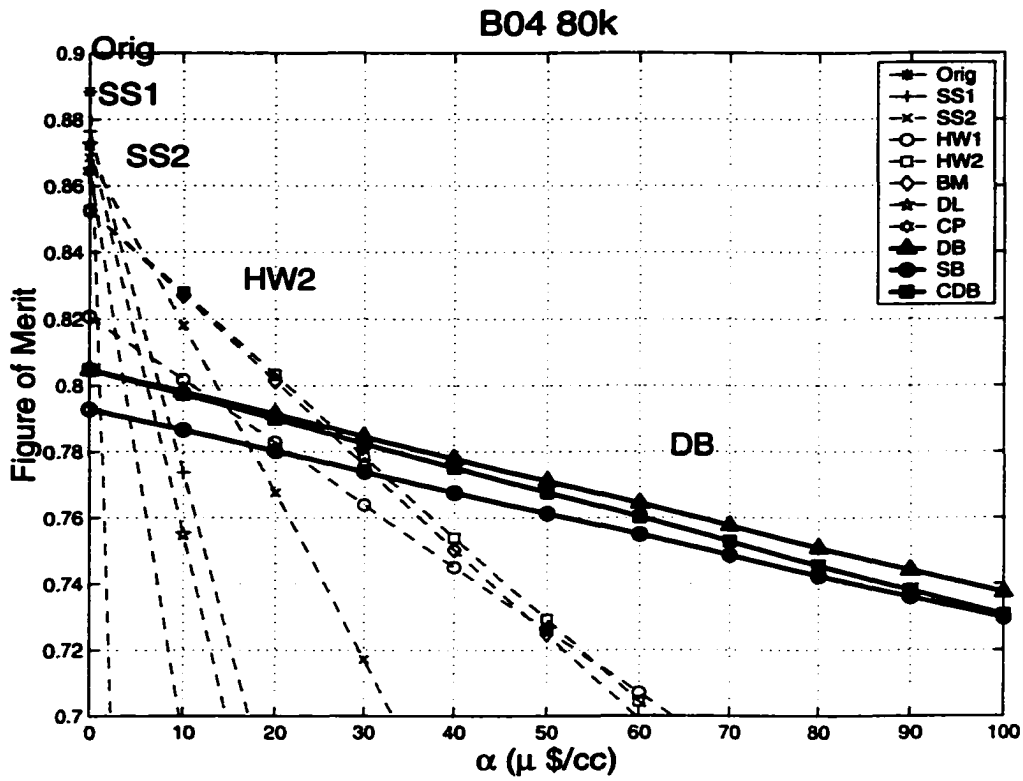
Appendix C

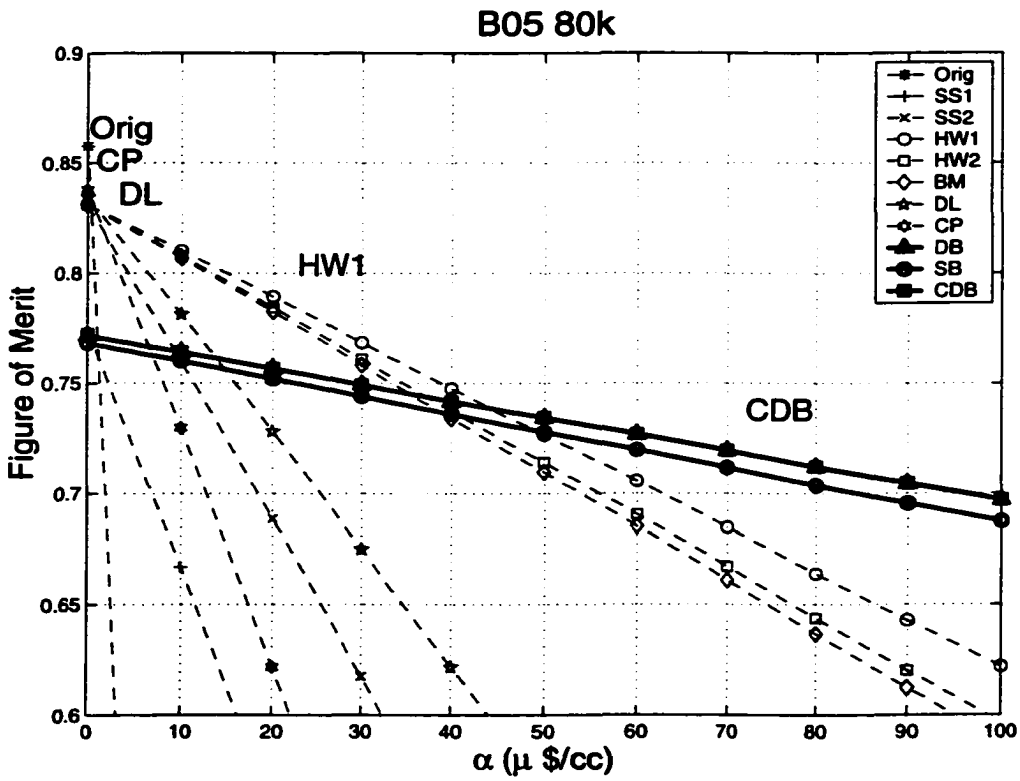
FIGURE OF MERIT EXPERIMENTS (80K SETUP)

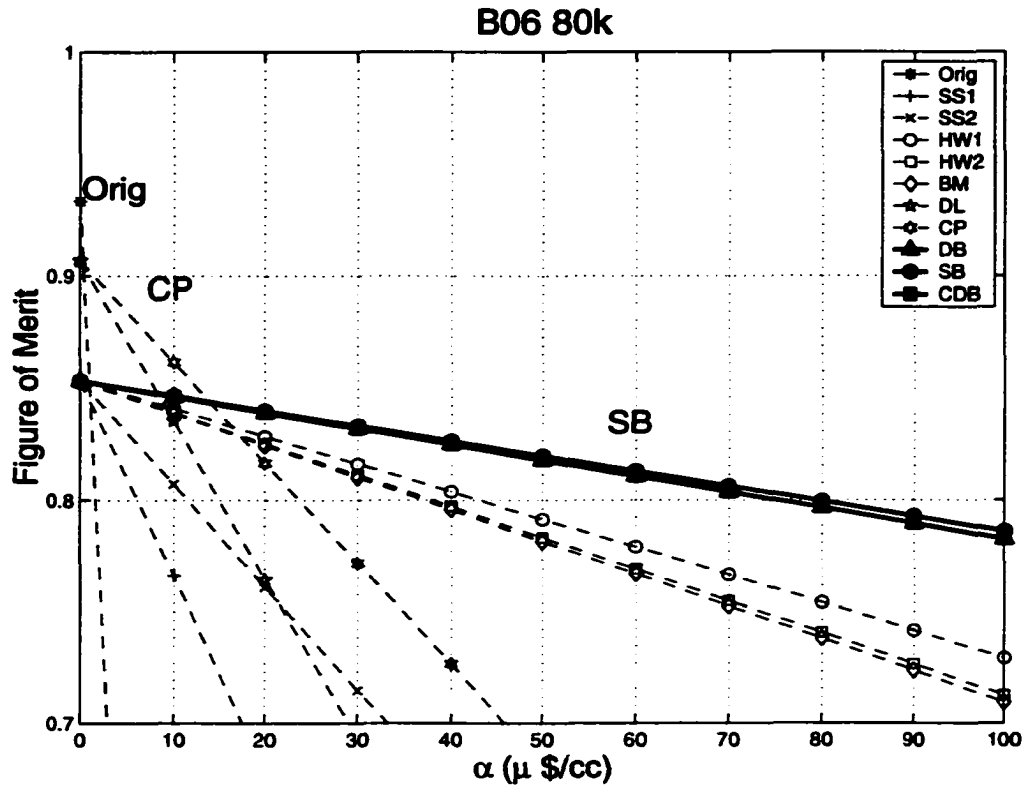


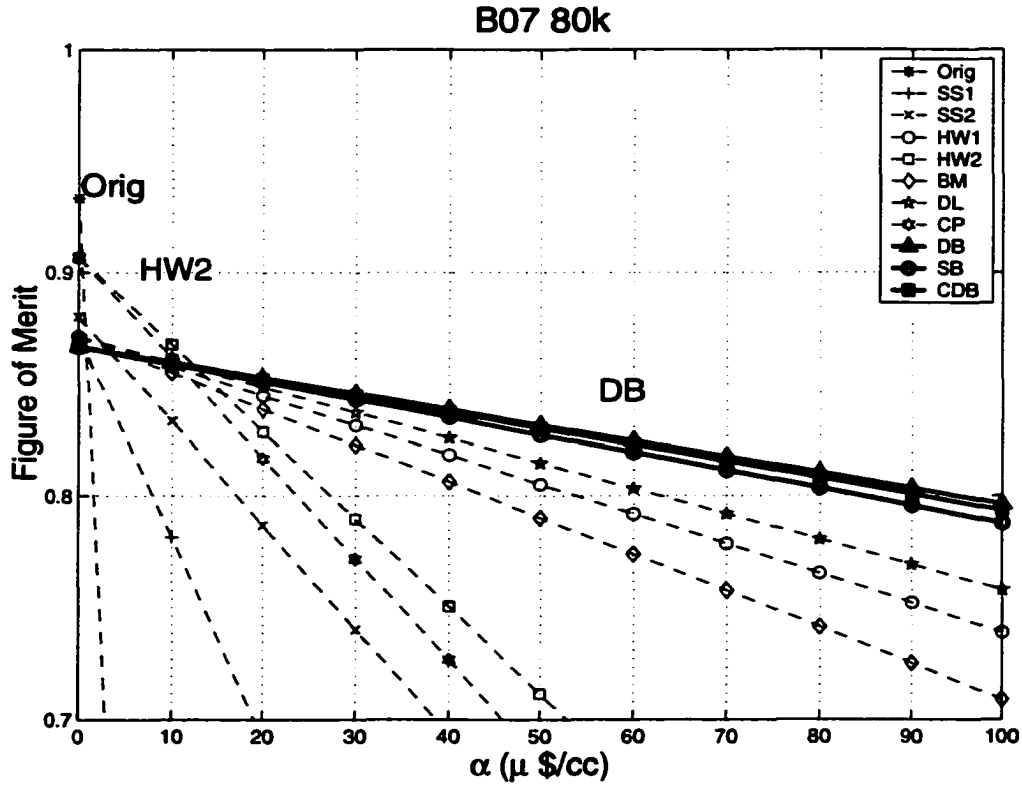


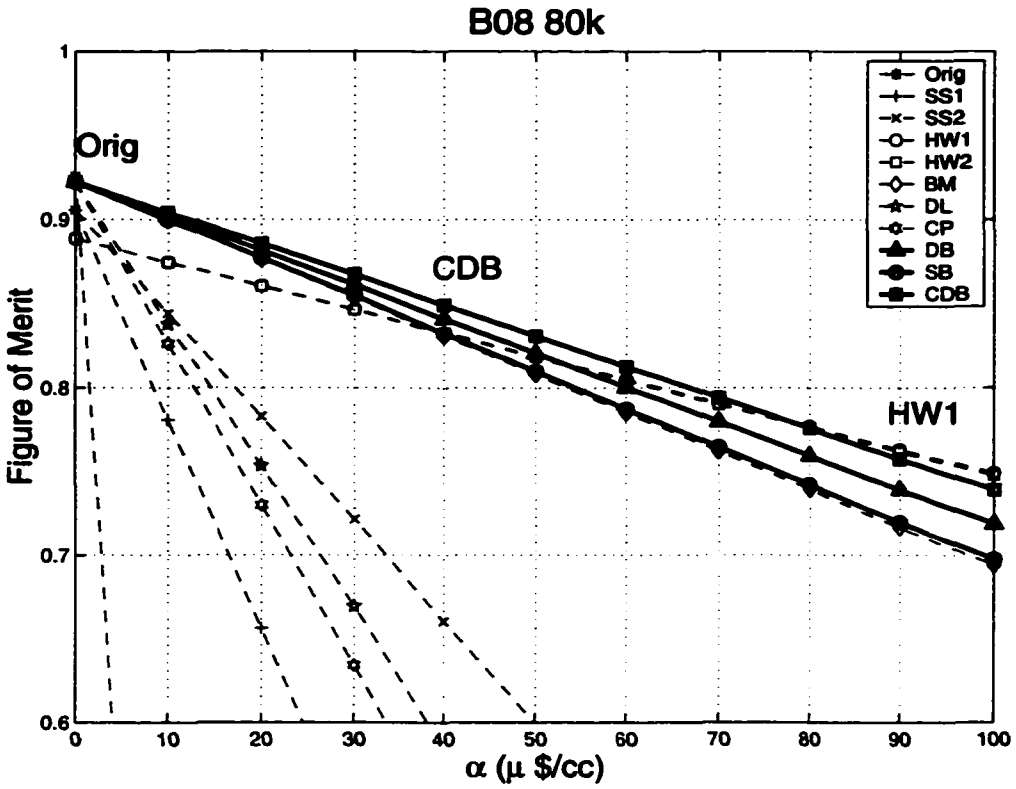


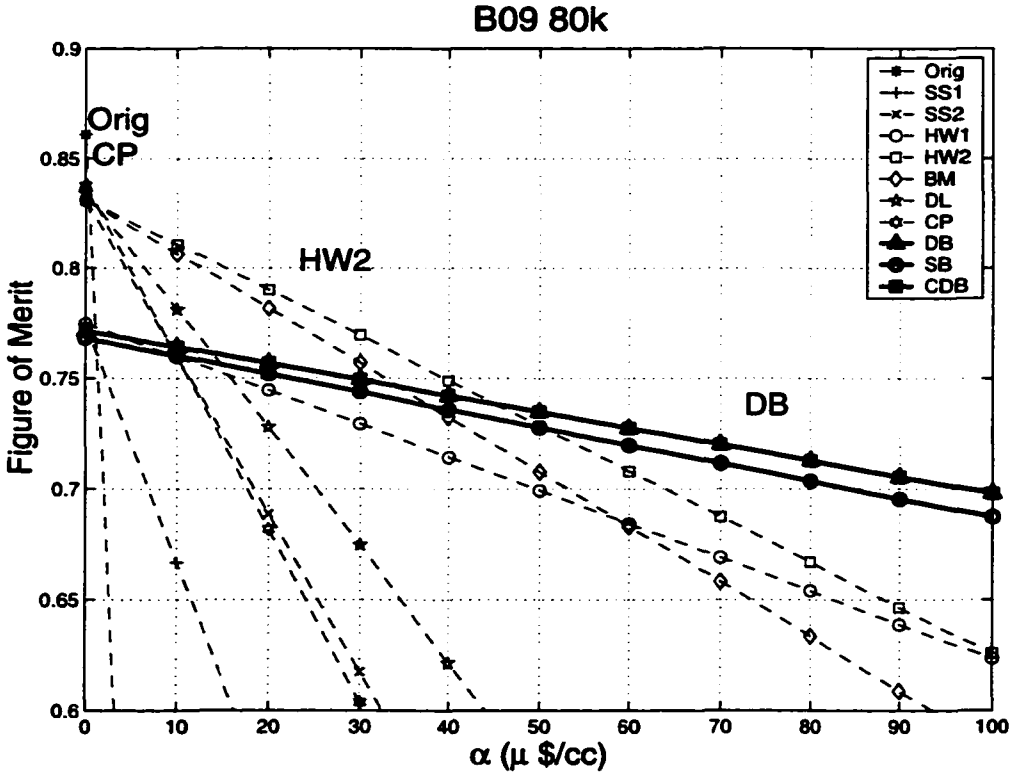


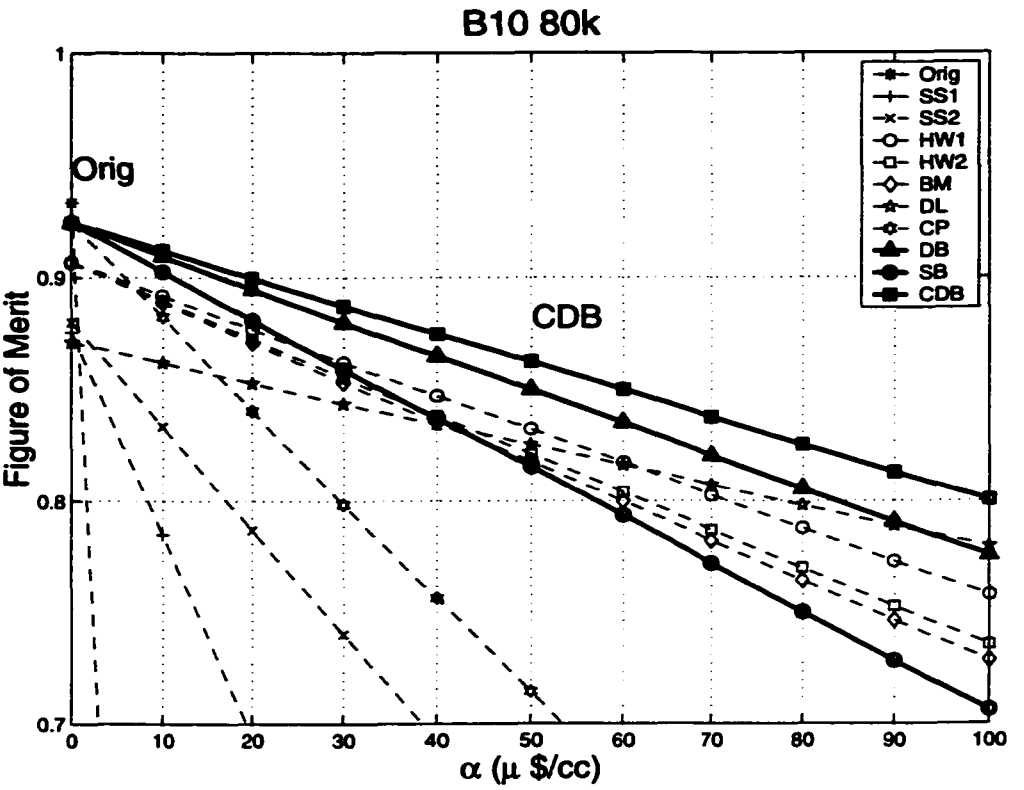


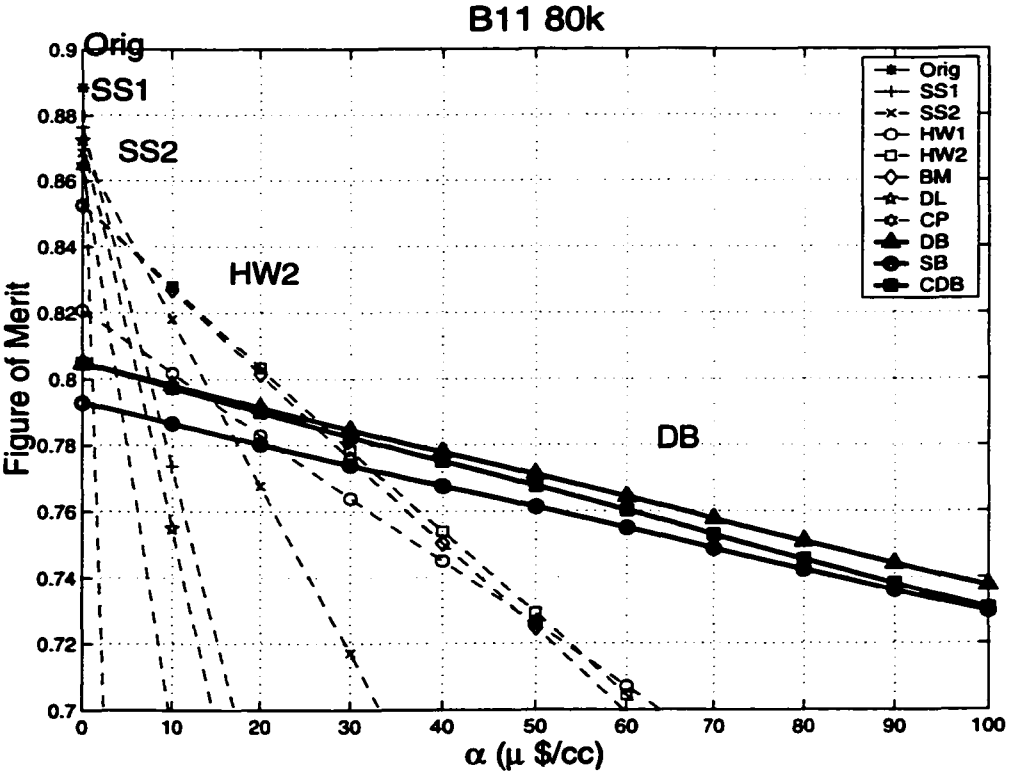


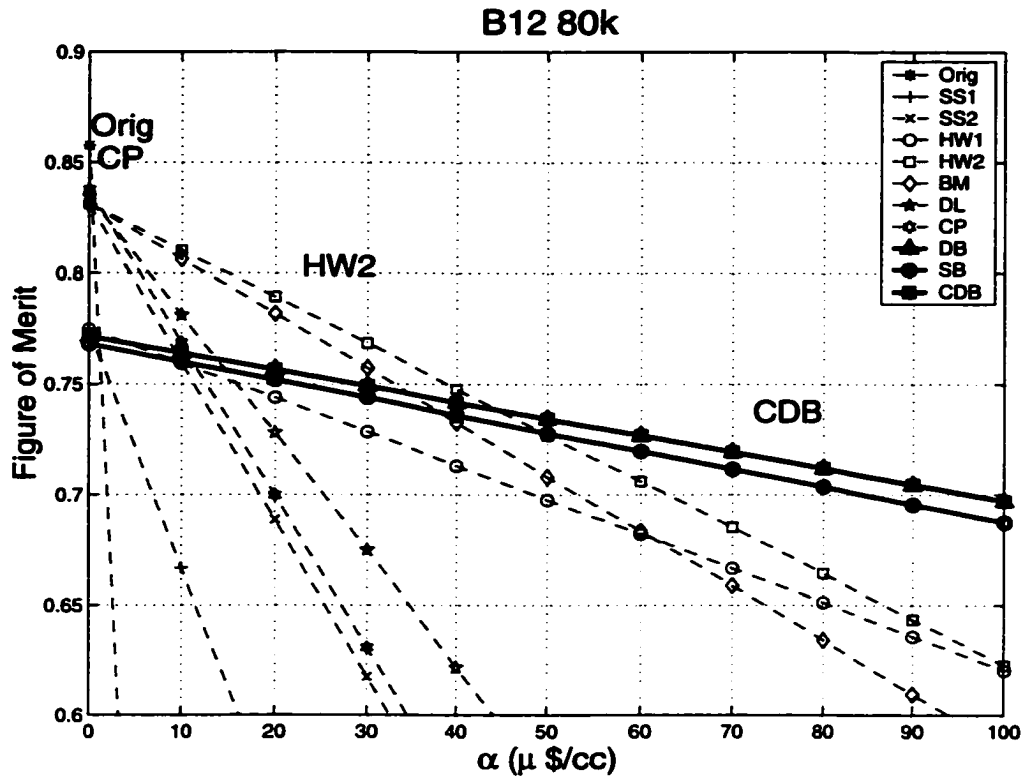


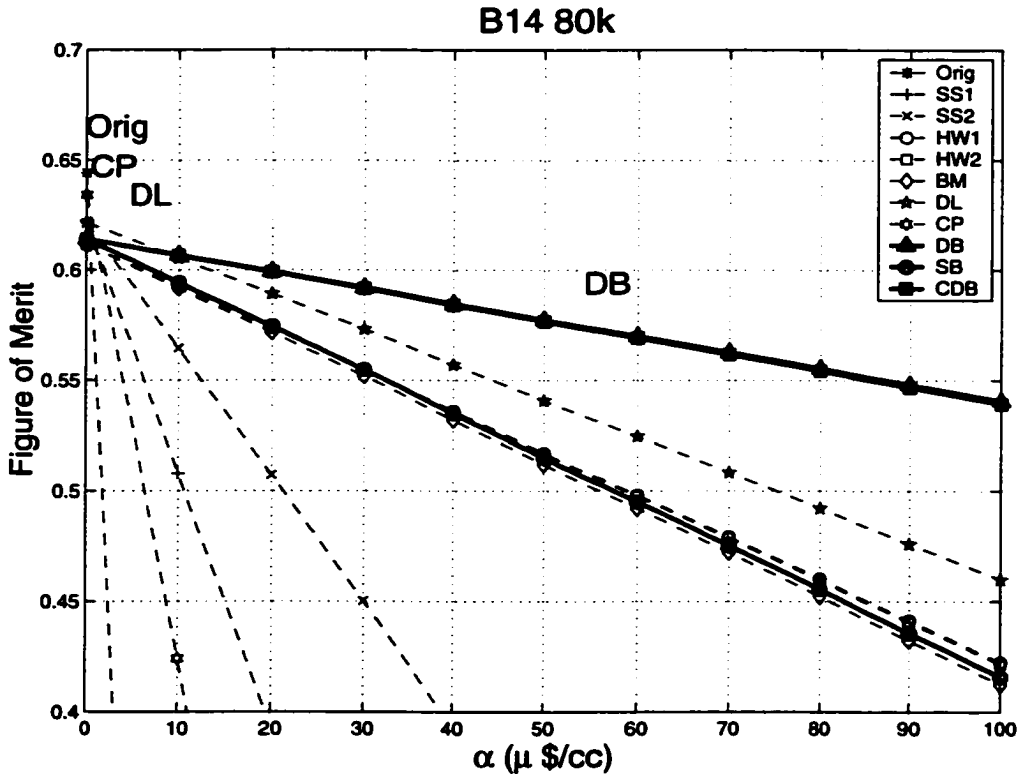


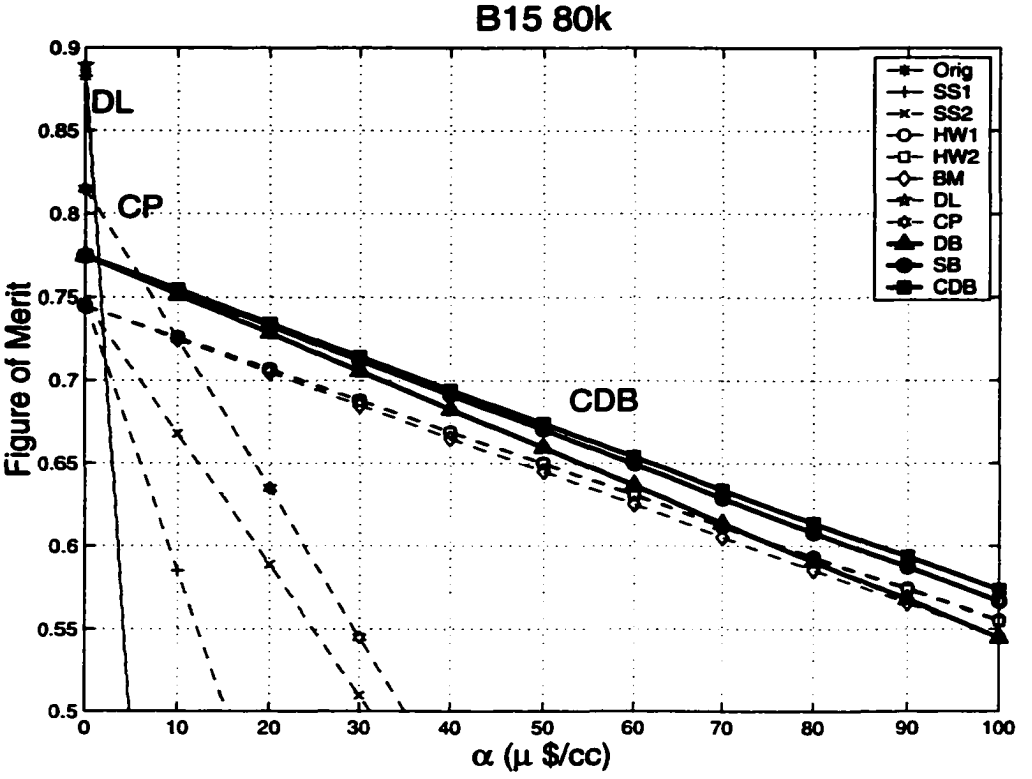






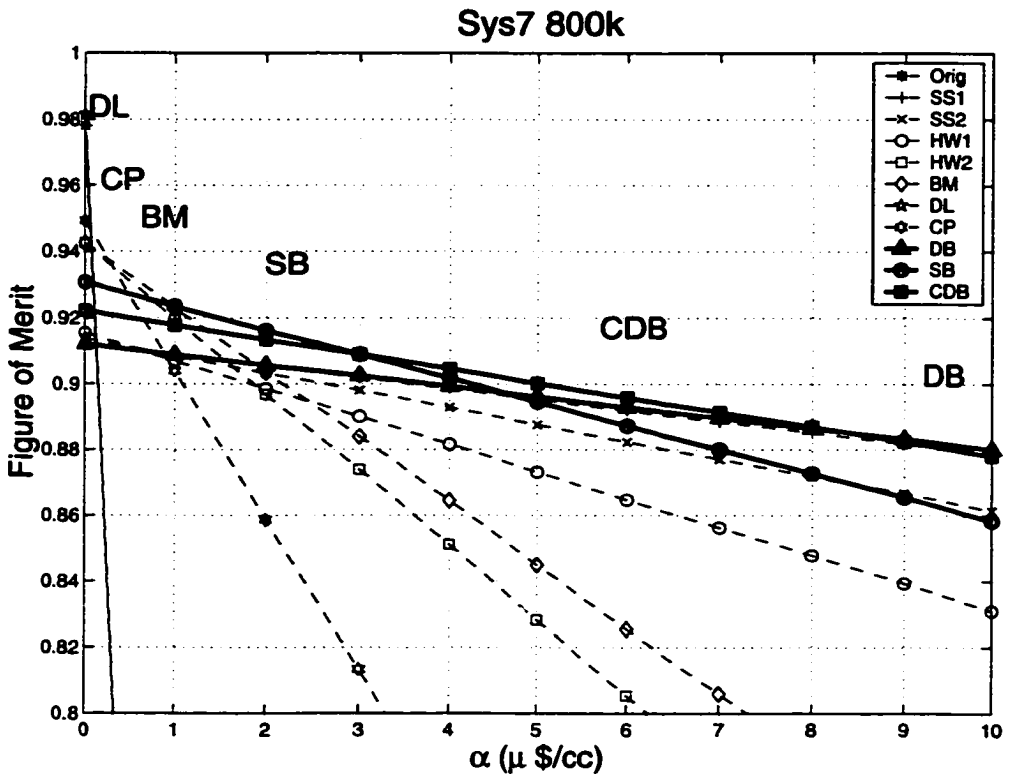


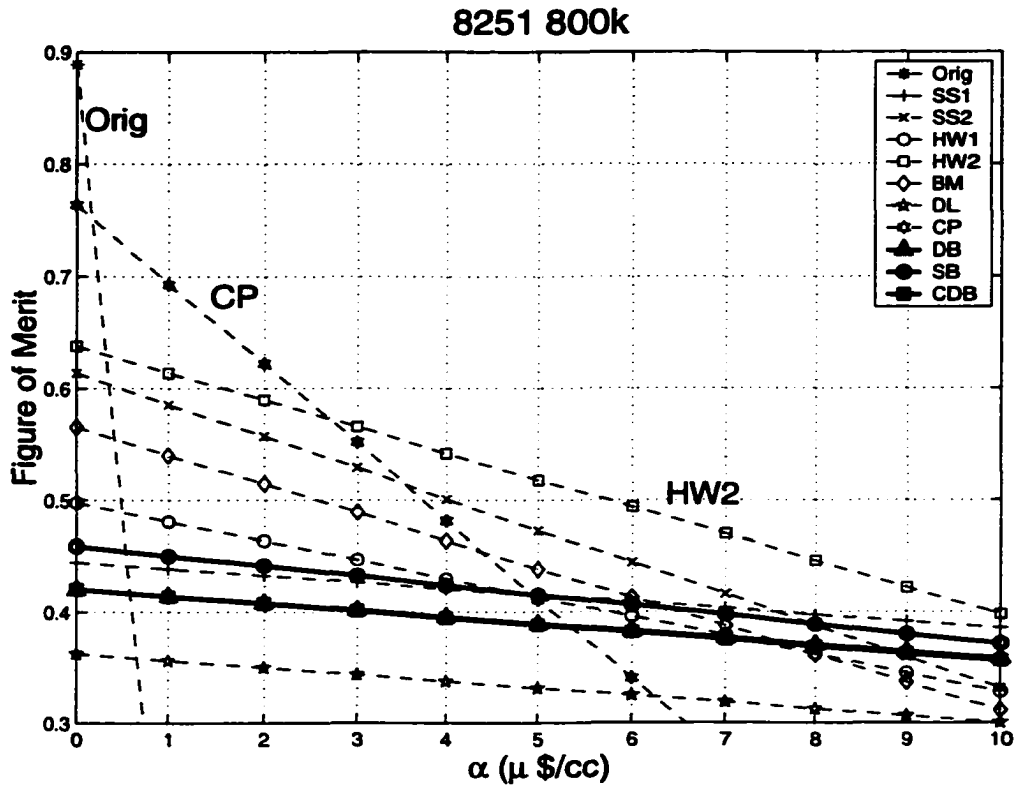


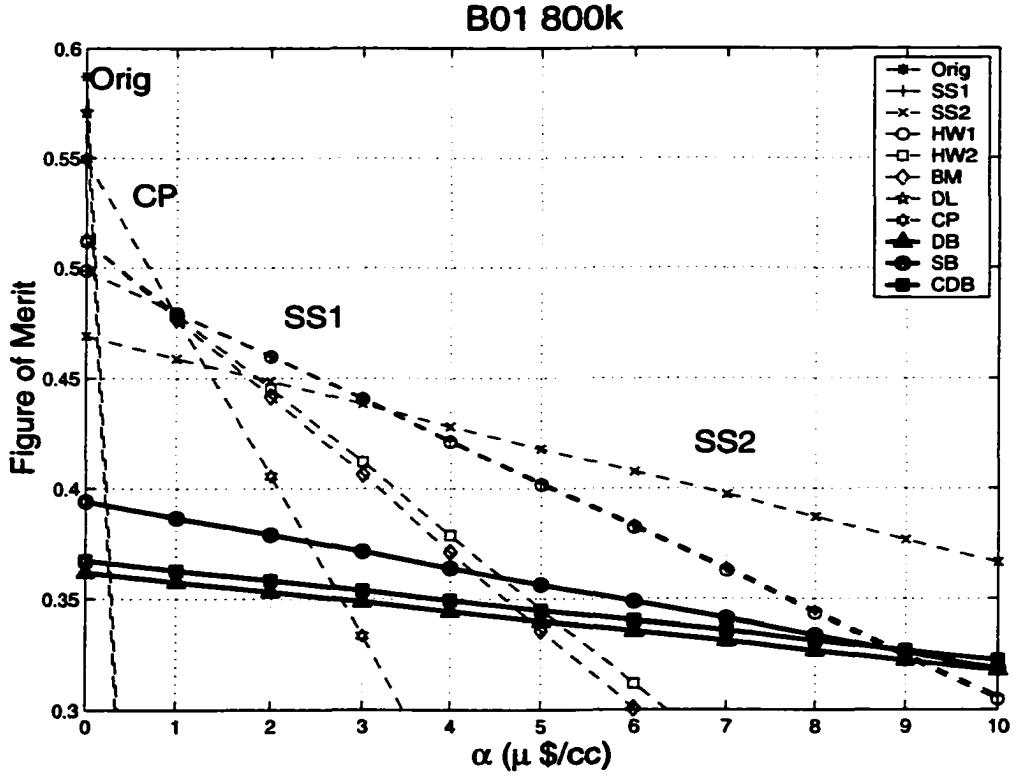


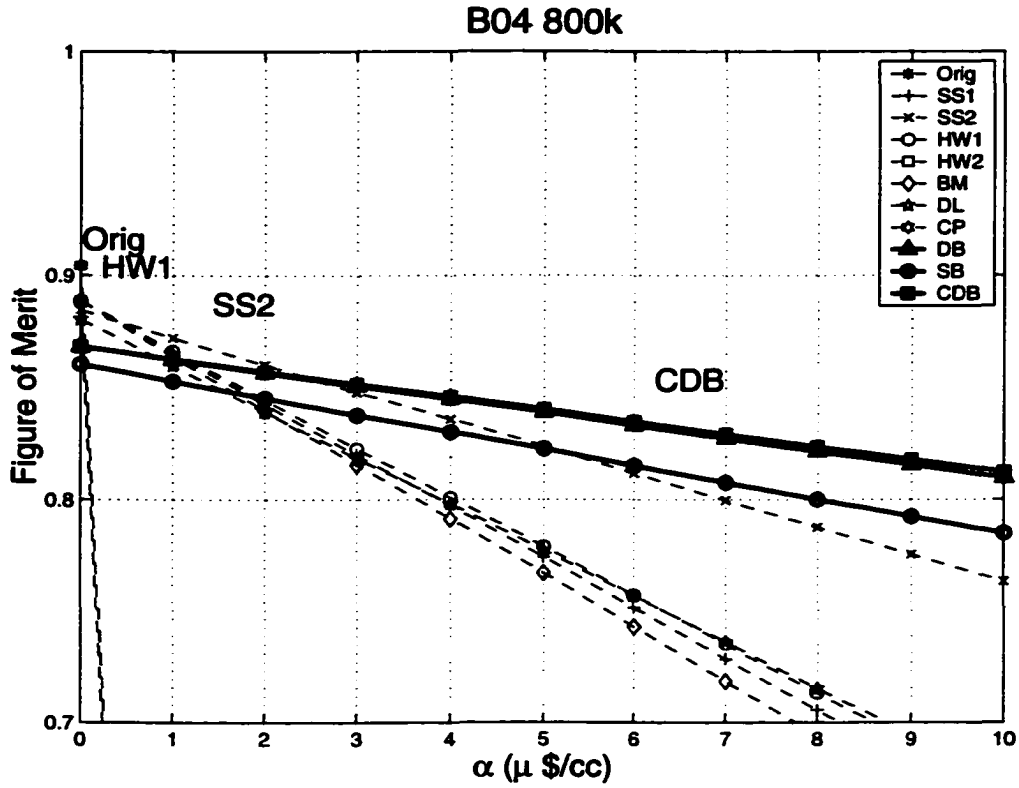
Appendix D

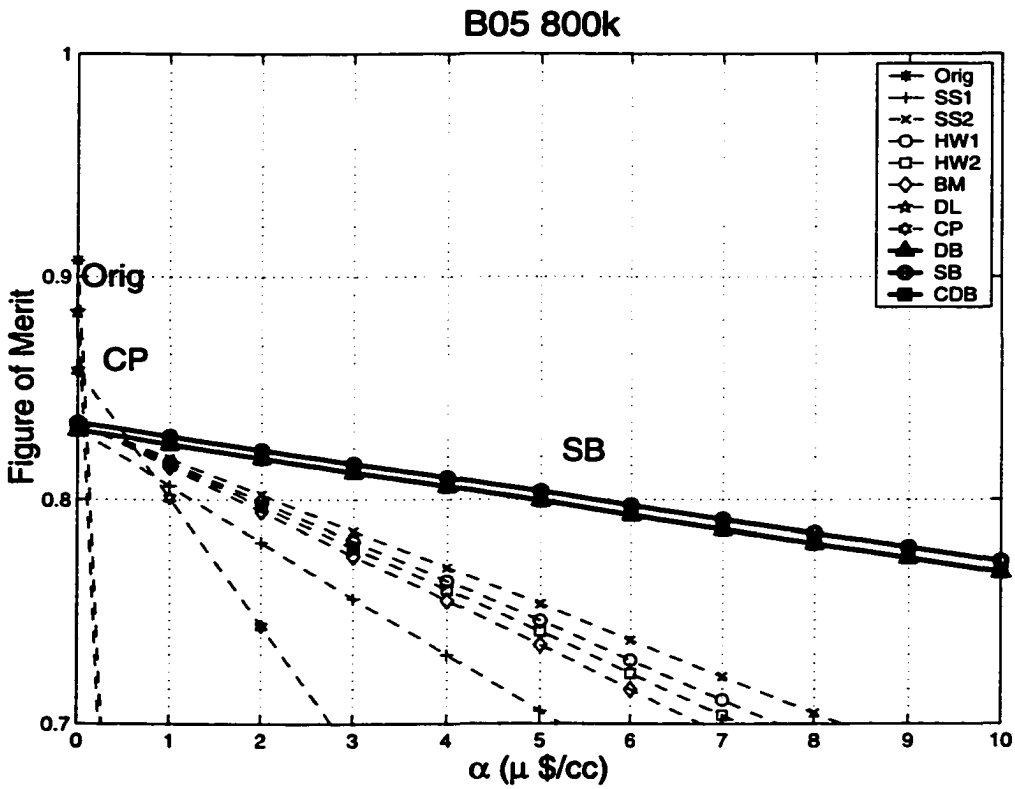
FIGURE OF MERIT EXPERIMENTS (800K SETUP)

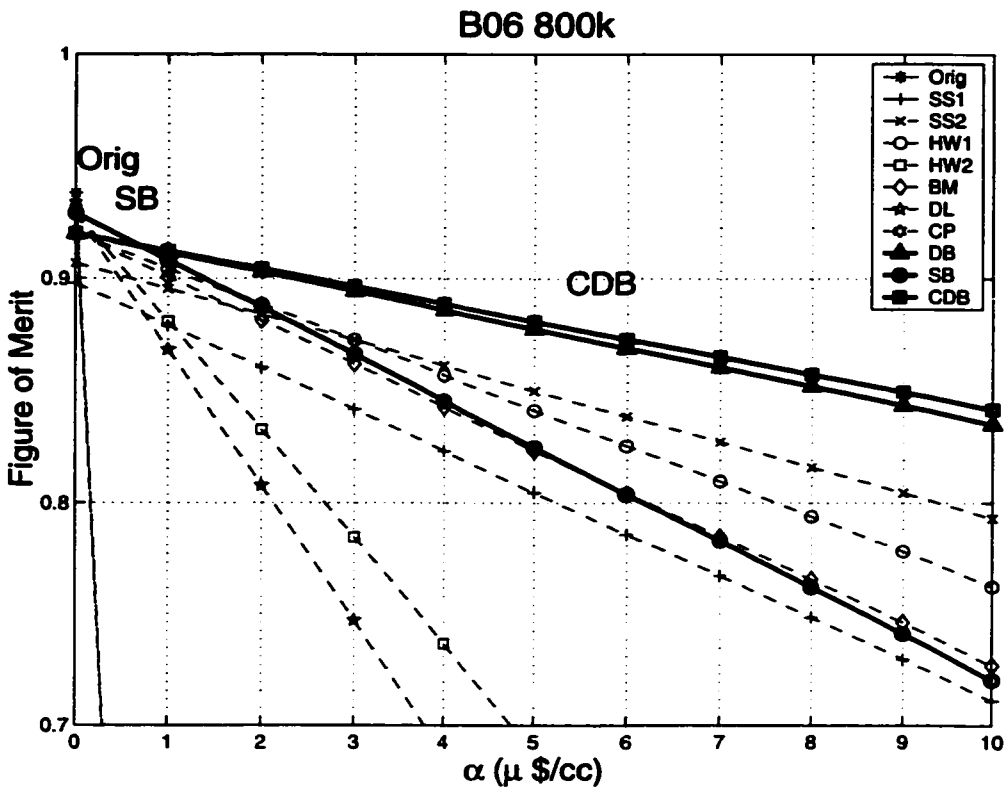


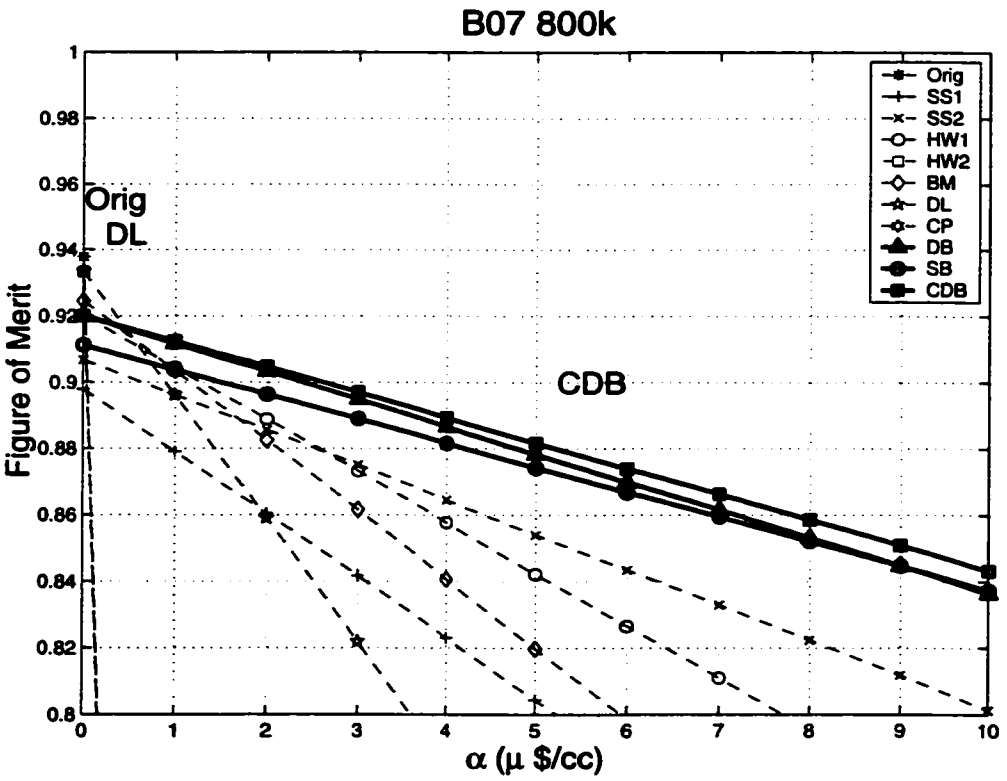


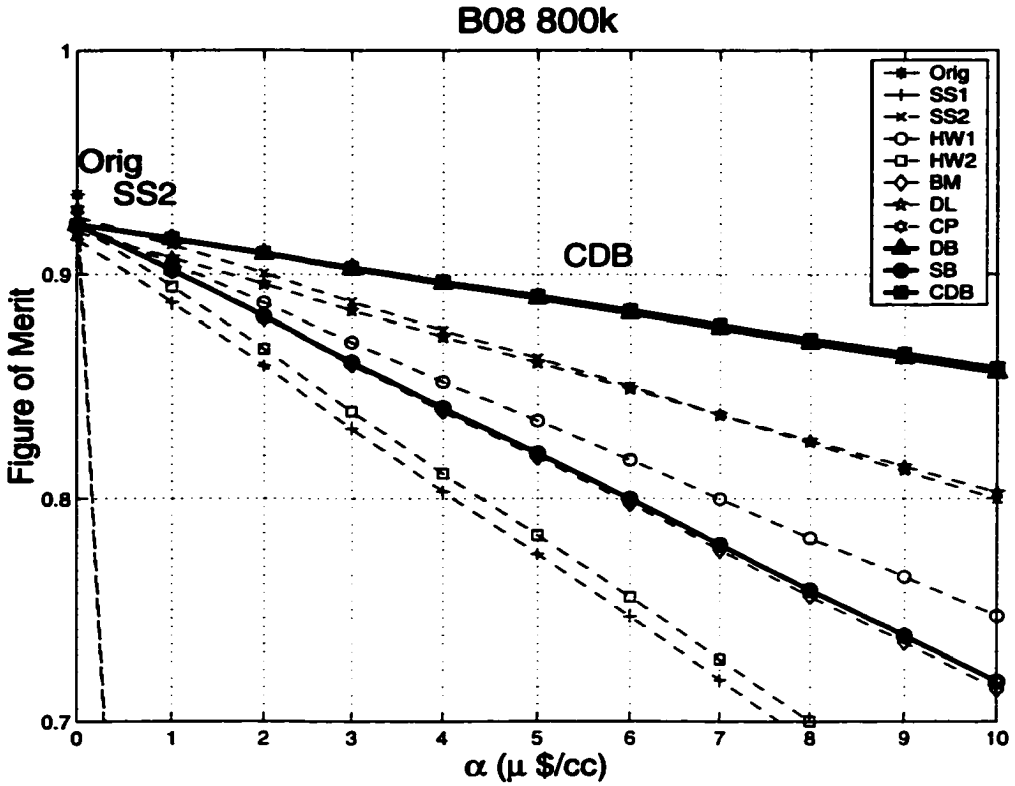


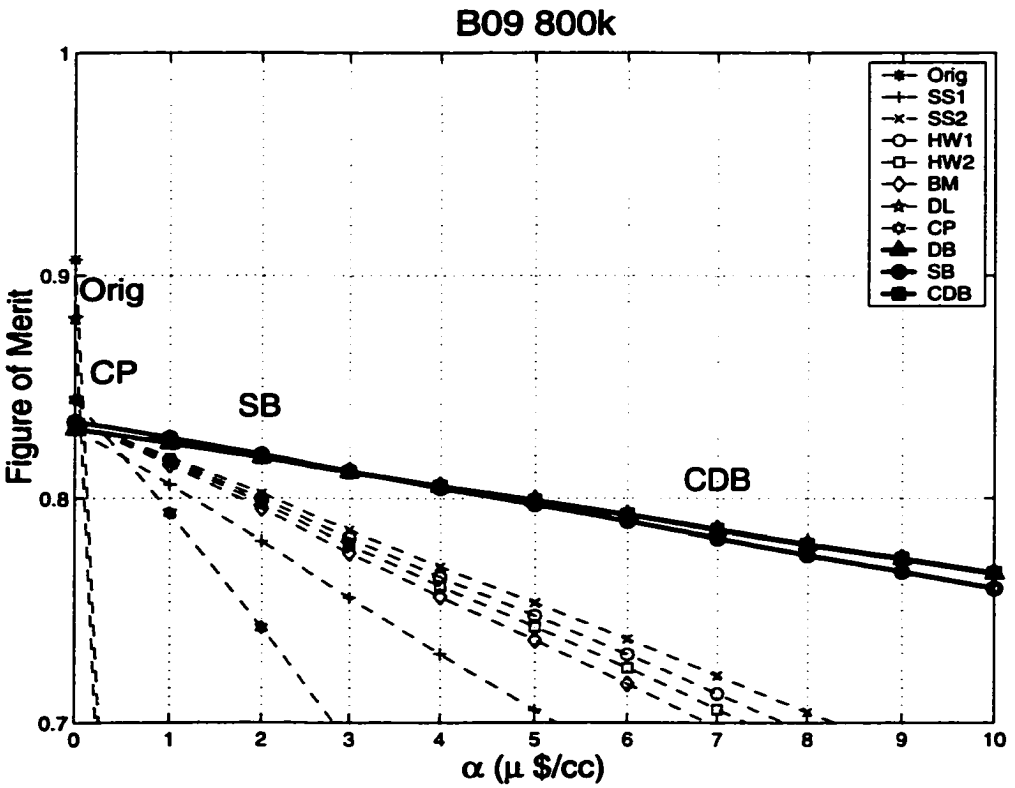


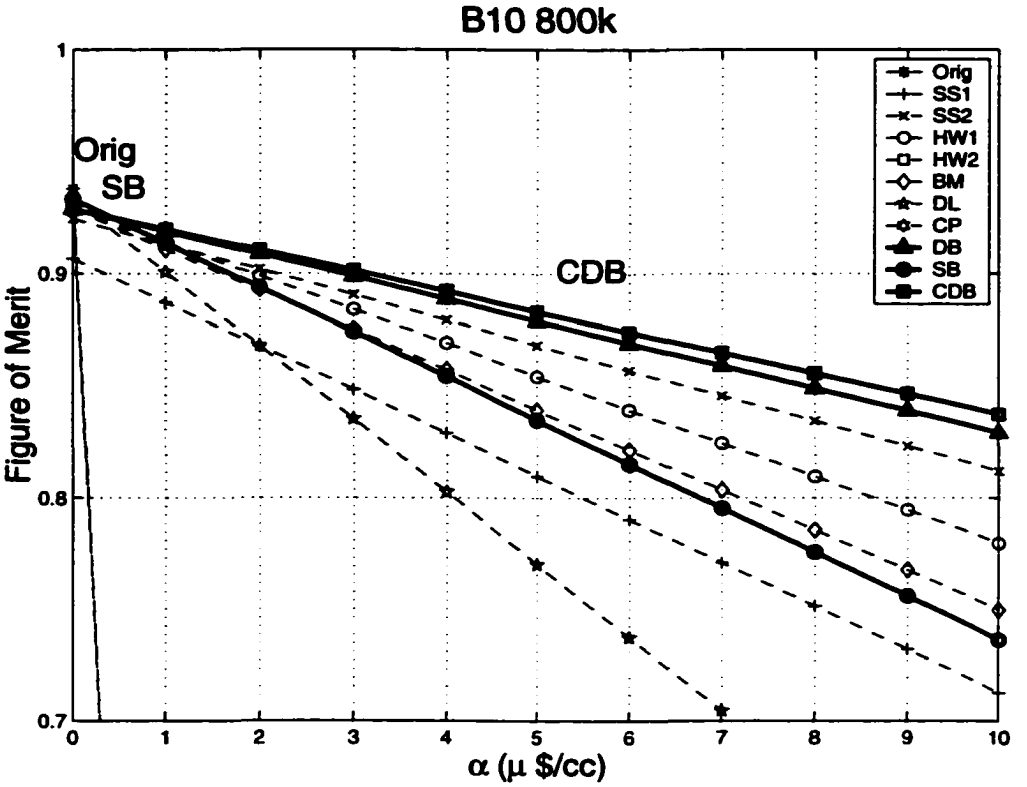


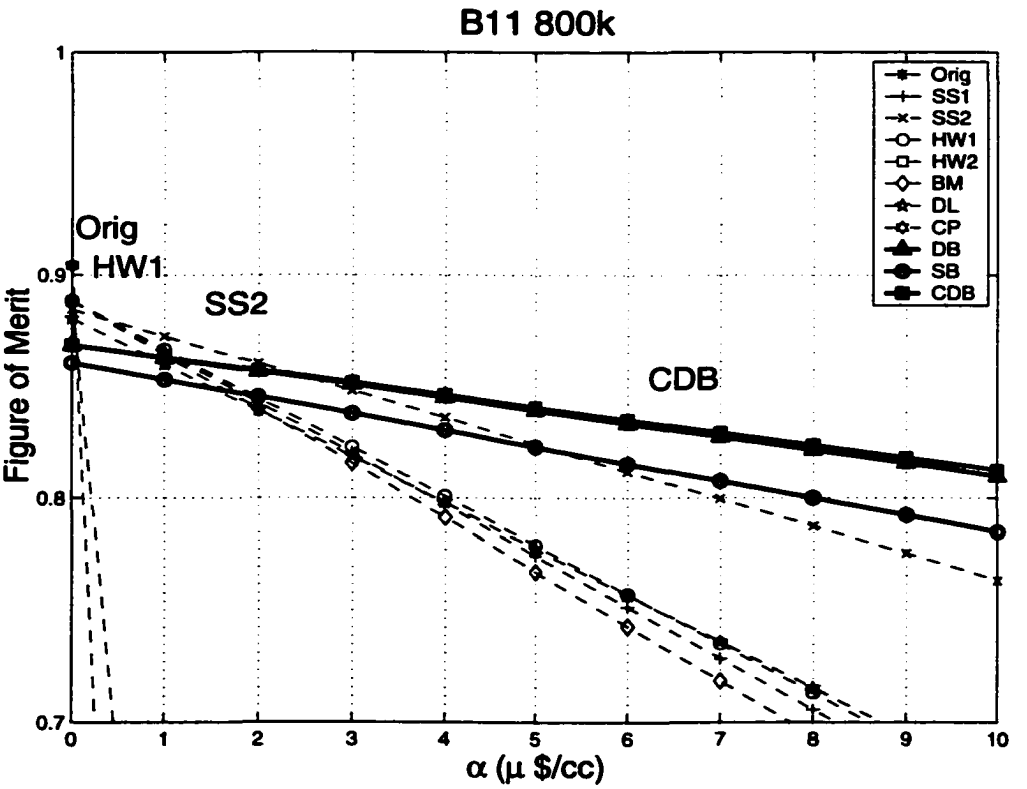


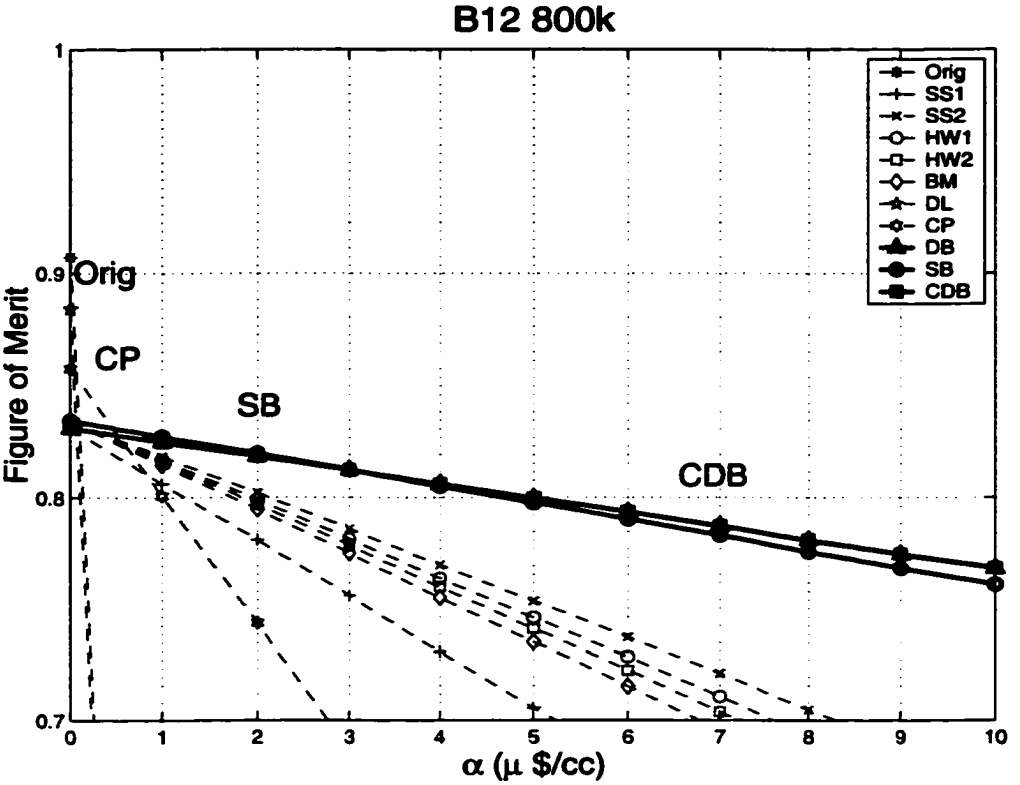


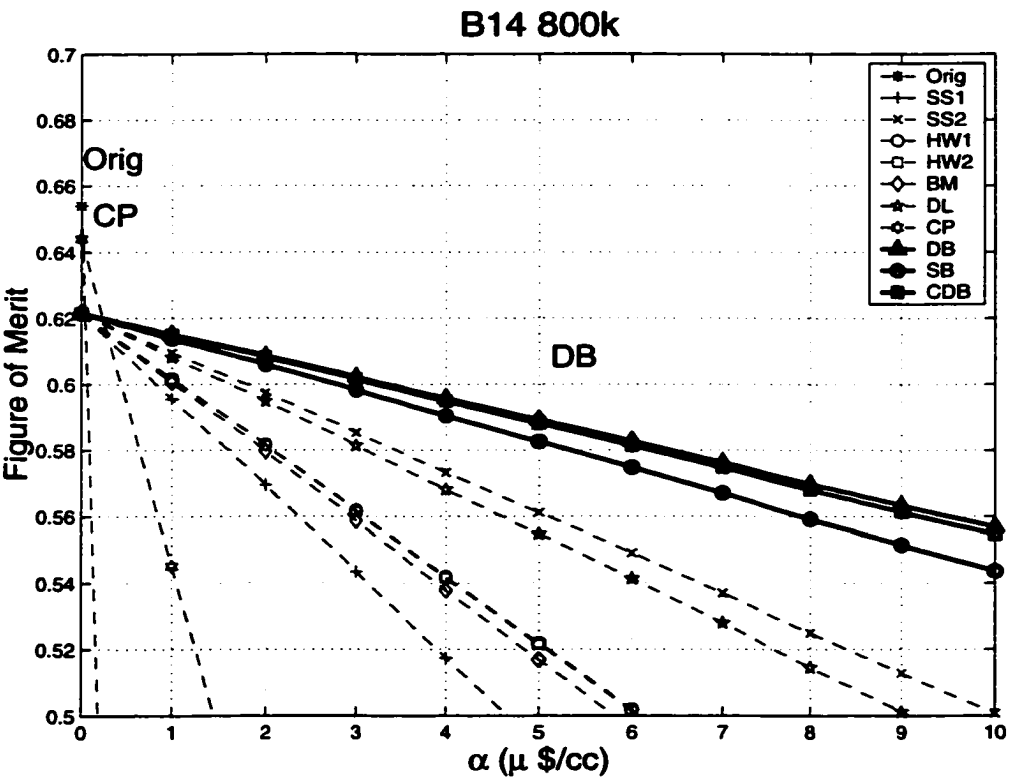


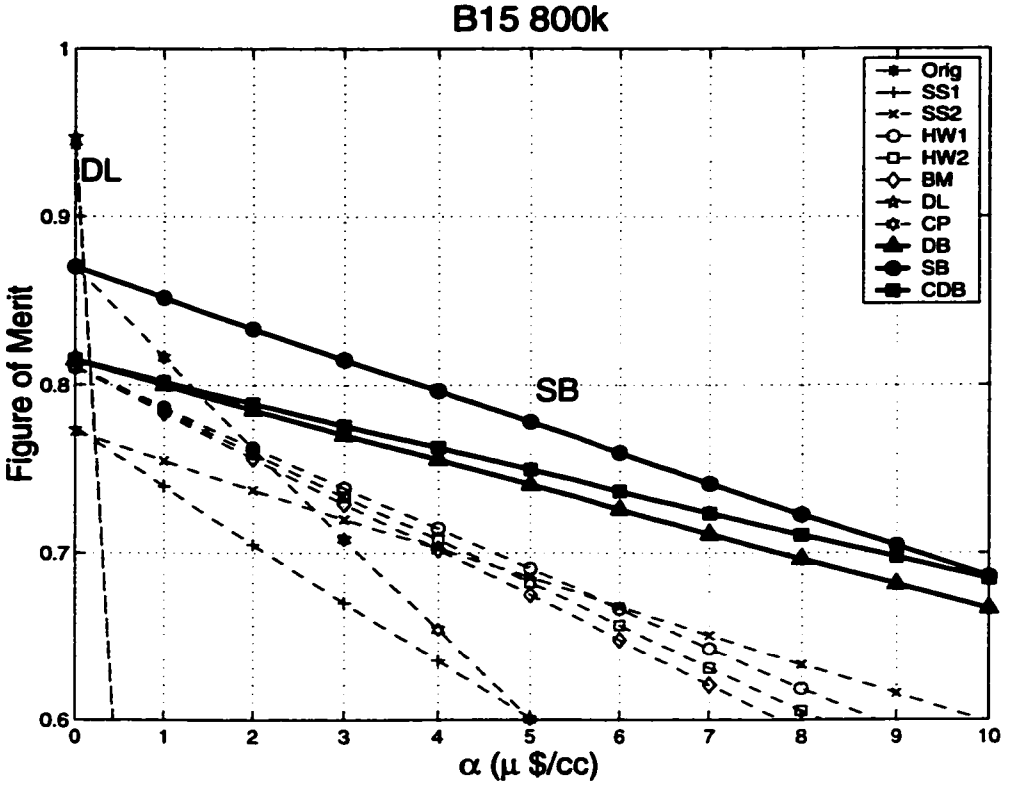












Appendix E

DEGREE OF INEFFICIENCY

Model	Orig	SS1	SS2	HW1	HW2	BM	DL	CP	DB	SB	CDB
Sys7	3774.25	244.36	97.35	17.78	449.09	27.83	27.83	158.51	12.98	59.98	12.13
8251	3889.57	415.62	161.84	52.00	48.82	53.27	493.25	768.47	6.93	16.74	10.33
B01	3889.94	512.47	272.53	21.03	33.58	37.98	179.02	449.81	13.28	19.74	13.23
B04	3890.11	406.36	201.01	32.11	40.46	42.06	272.28	141.78	5.51	3.76	5.51
B05	3900.68	411.05	187.01	29.00	122.55	43.95	19.50	152.35	2.55	6.80	3.90
B06	3906.30	548.92	232.67	13.06	13.71	23.83	331.03	389.68	11.33	22.13	1.13
B07	3887.44	513.28	273.34	49.13	19.89	40.74	180.13	300.62	13.54	20.60	13.59
B08	3929.12	440.29	215.45	30.18	41.33	44.83	37.09	148.01	12.41	47.06	0.01
B09	3889.57	415.62	161.84	52.00	48.82	53.27	493.25	768.47	6.93	16.74	10.33
B10	3890.03	512.56	272.62	50.06	20.77	39.62	179.10	254.89	13.37	19.83	13.32
B11	3933.54	524.45	241.70	60.73	61.13	65.68	37.00	993.57	0.37	62.72	1.02
B12	3792.55	738.22	324.02	25.45	25.85	30.40	3786.27	310.63	15.96	5.06	1.56
B14	3569.61	82.68	110.33	29.19	36.04	32.73	195.67	211.32	31.68	28.65	53.26
B15	2630.66	22.60	60.17	17.00	19.10	21.90	32.40	266.39	0.49	5.39	0.79
Average	3769.53	413.46	200.85	34.19	70.08	39.86	447.42	379.61	10.52	23.94	10.01

Degree of Inefficiency for the 80k-Setup

Model	Orig	SS1	SS2	HW1	HW2	BM	DL	CP	DB	SB	CDB
Sys7	3838.08	22.70	7.41	23.55	79.88	89.27	3854.17	235.61	85.26	68.85	80.36
8251	3938.09	68.31	18.66	63.57	3954.03	75.46	65.22	3938.11	3.32	19.79	2.05
B01	3896.94	98.65	50.81	58.18	63.13	69.09	3920.12	231.61	4.85	0.69	4.80
B04	3943.35	76.71	31.26	39.97	192.61	57.69	255.12	3947.81	3.63	56.53	0.31
B05	3944.13	77.49	27.92	39.75	3948.59	62.17	134.22	3948.59	3.70	7.70	0.30
B06	3954.59	115.44	27.73	55.64	106.96	71.92	29.65	3961.37	1.11	70.26	0.10
B07	3892.61	94.32	46.49	52.19	57.16	63.10	3919.11	210.53	1.05	2.72	0.91
B08	3945.46	73.59	15.06	28.99	3949.92	43.96	113.26	3949.93	4.25	48.46	0.09
B09	3938.09	68.31	18.66	63.57	1999.85	75.46	65.22	3938.11	3.32	19.79	2.05
B10	3893.64	95.35	47.51	54.88	59.83	65.79	3916.82	228.31	1.29	3.24	1.22
B11	3935.32	98.08	28.31	67.56	67.91	72.55	34.76	440.35	0.28	6.88	1.34
B12	3835.97	177.64	93.11	87.82	96.15	103.47	3831.74	178.31	37.71	0.38	28.71
B14	3654.39	120.79	63.32	123.02	18.04	97.56	204.56	124.99	146.70	120.69	147.58
B15	2625.74	7.06	14.77	29.18	74.06	57.35	2622.35	179.50	6.29	7.86	2.19
Average	3802.60	85.32	35.07	56.28	1047.72	71.77	1640.45	1822.37	21.63	30.99	19.43

Degree of Inefficiency for the 800k-Setup

Appendix F

INTERRUPTION PREDICTION ERROR VS WINDOW SIZE

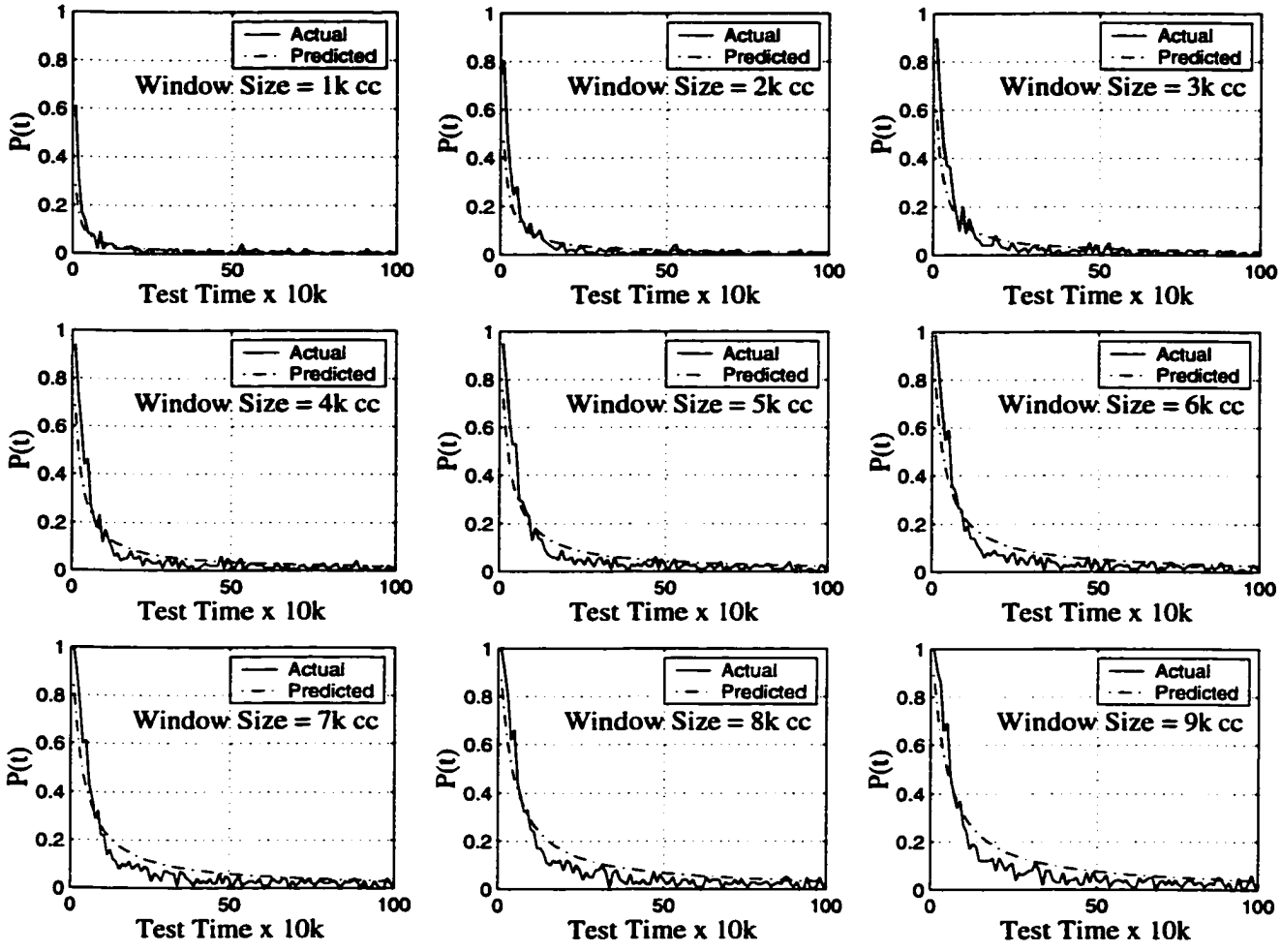


Figure F.1: Interruption Error for B01

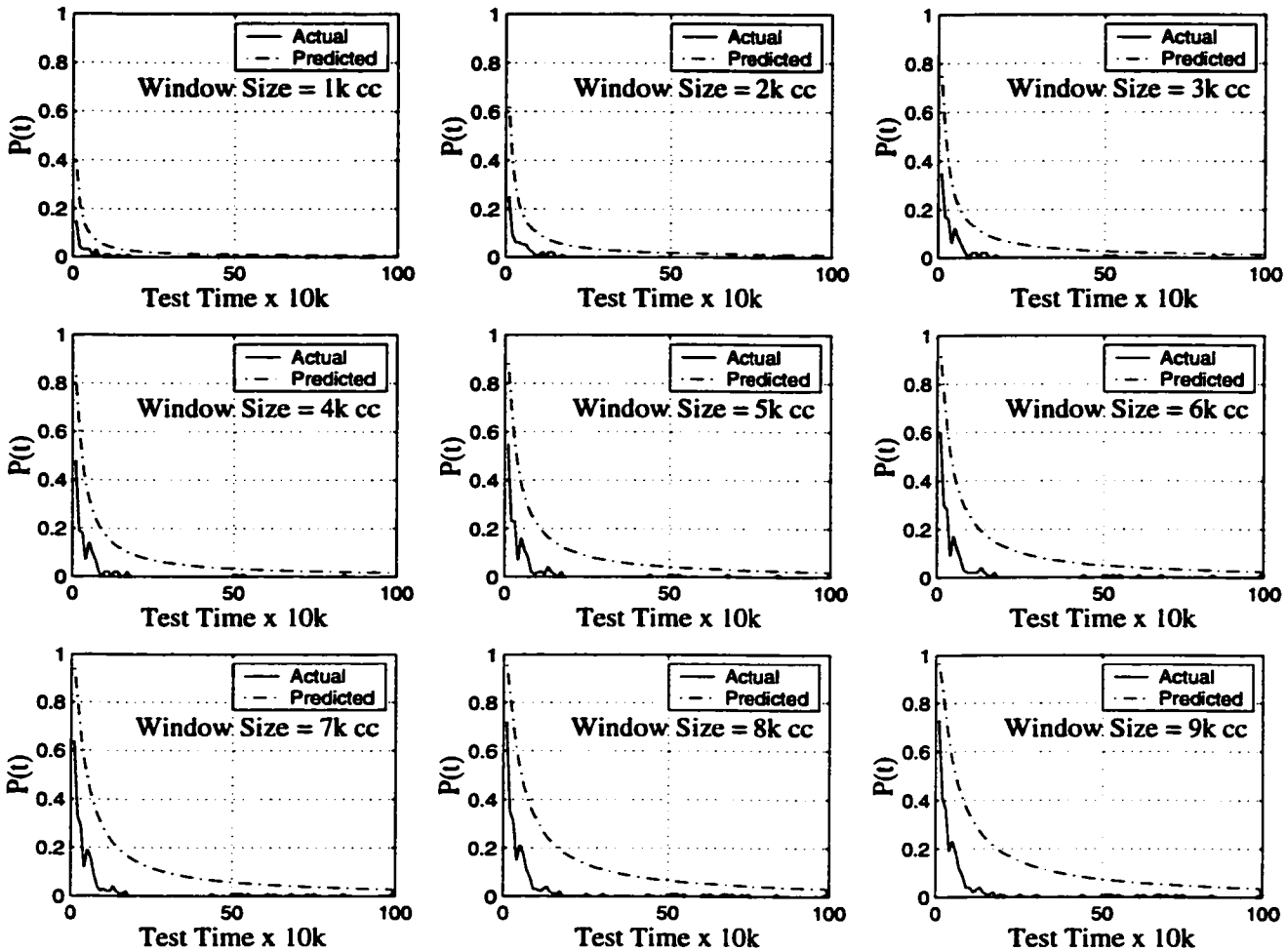


Figure F.2: Interruption Error for B04

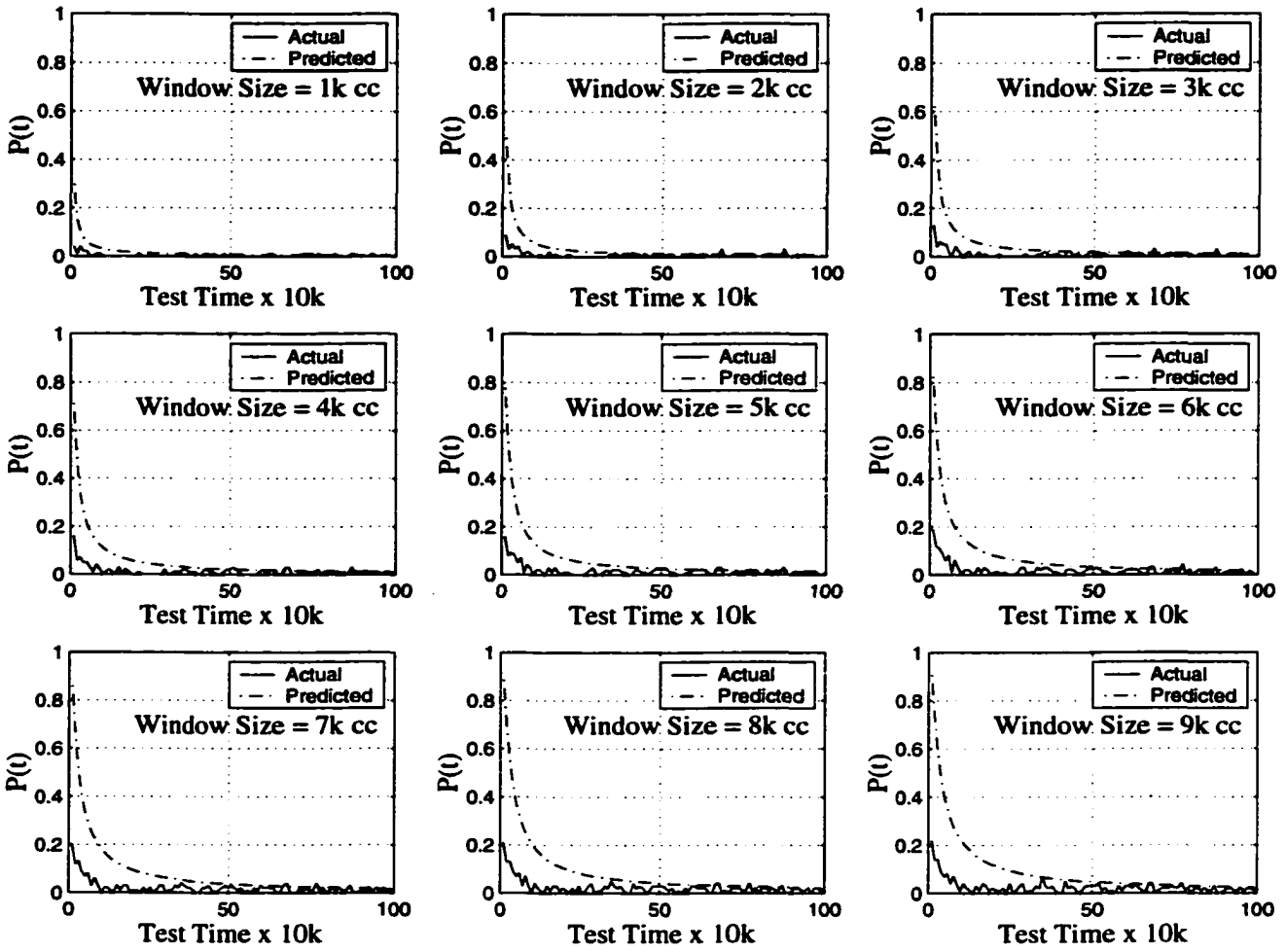


Figure F.3: Interruption Error for B05

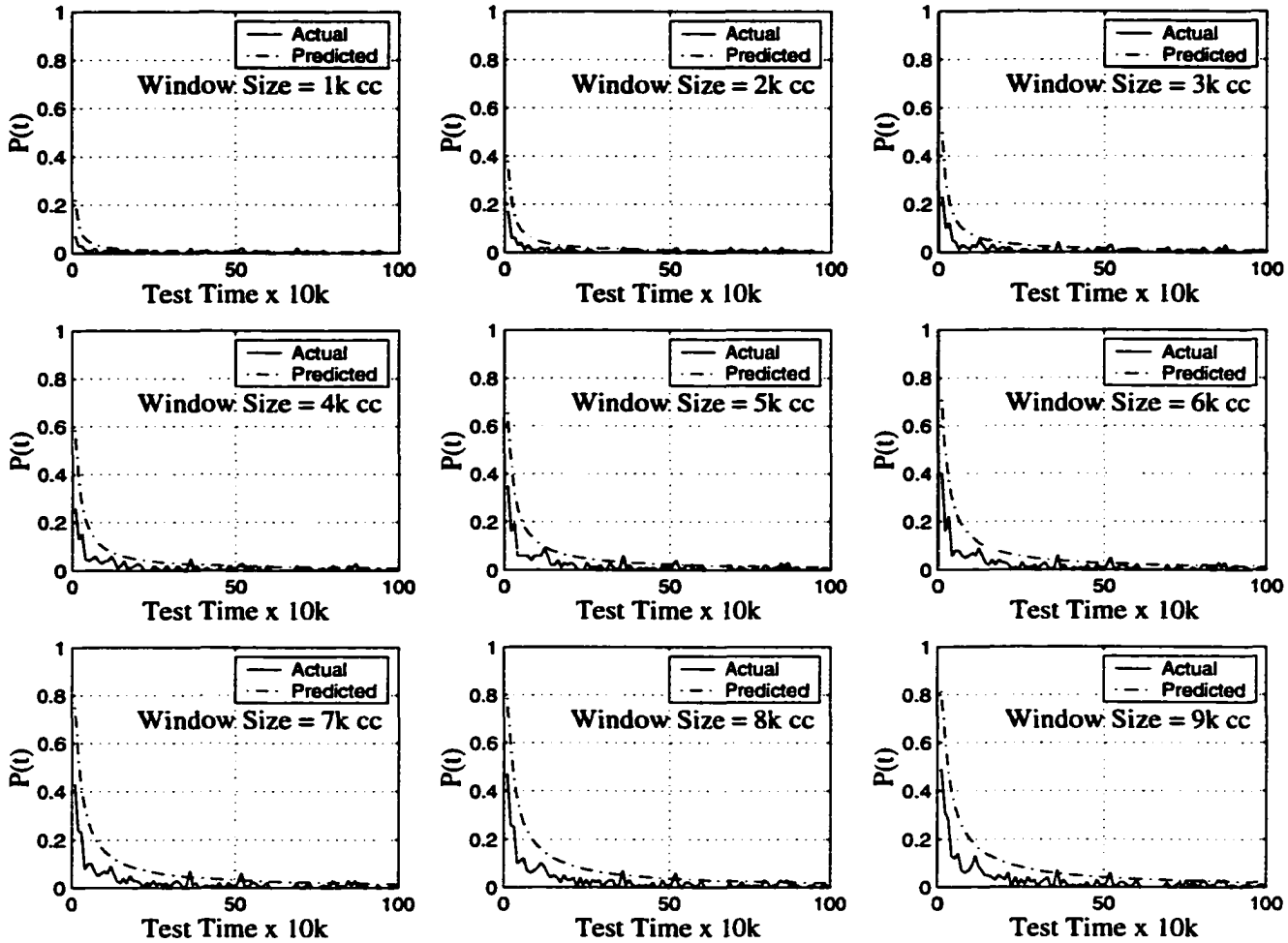


Figure F.4: Interruption Error for B06

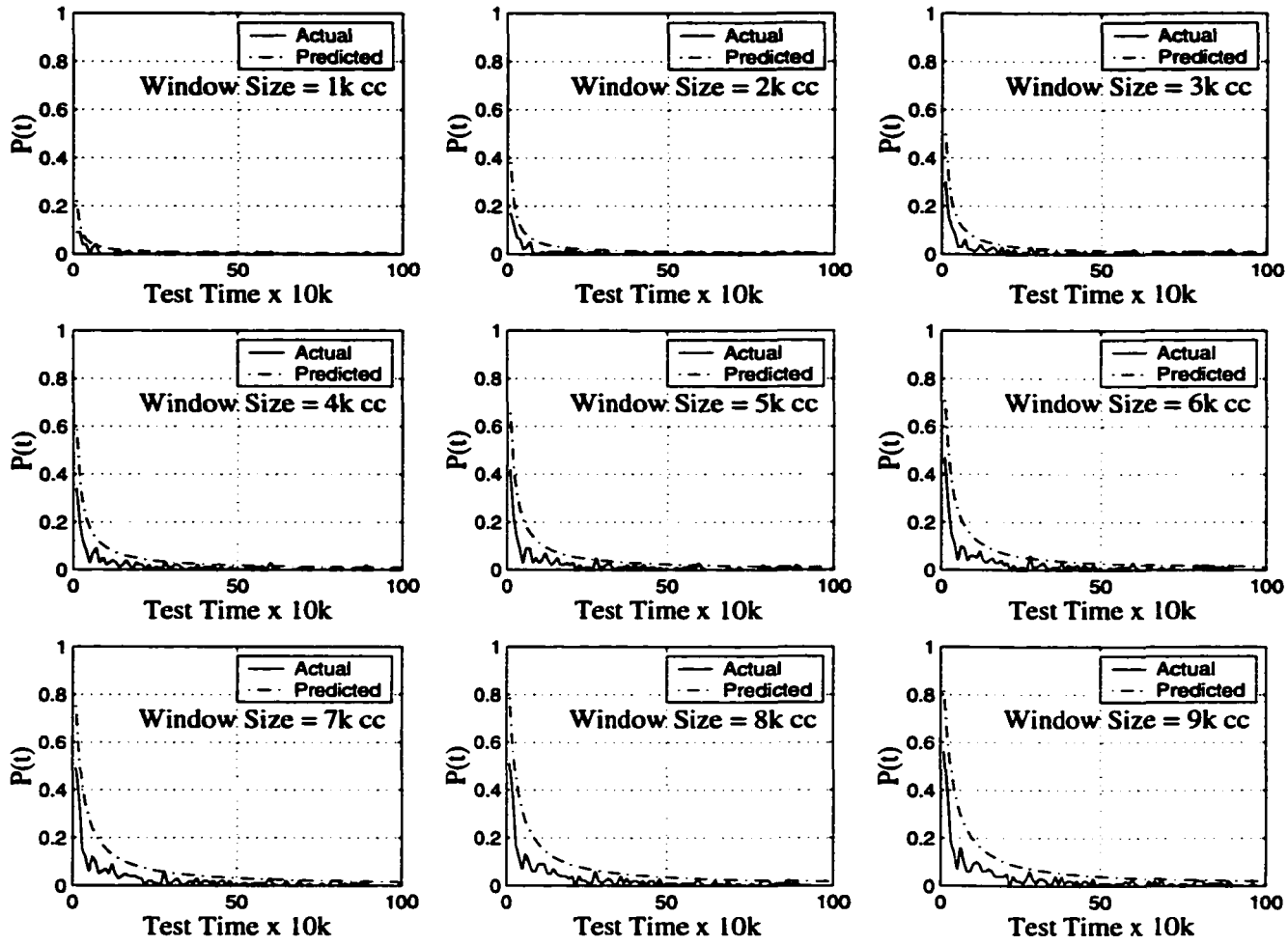


Figure F.5: Interruption Error for B07

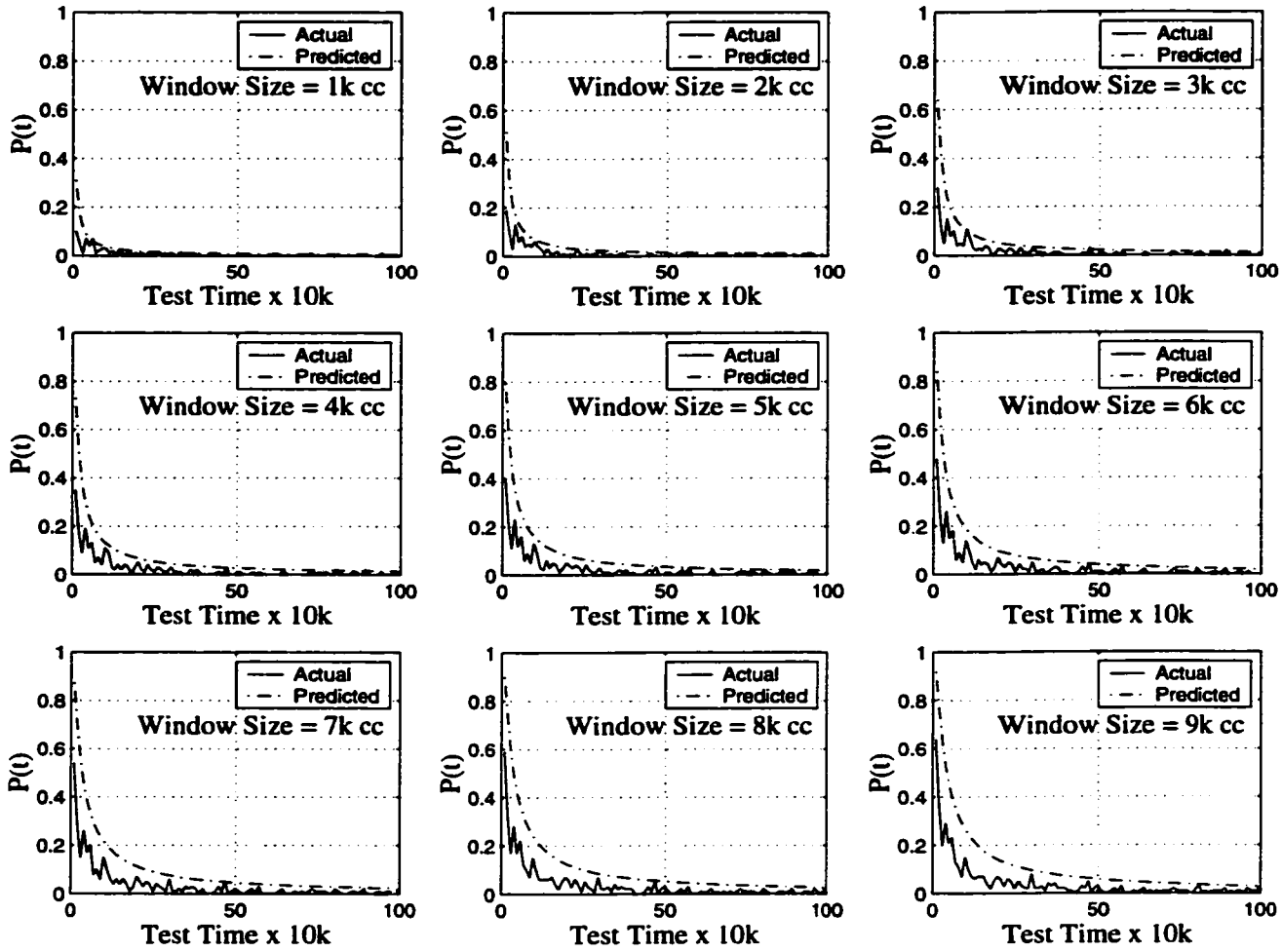


Figure F.6: Interruption Error for B08

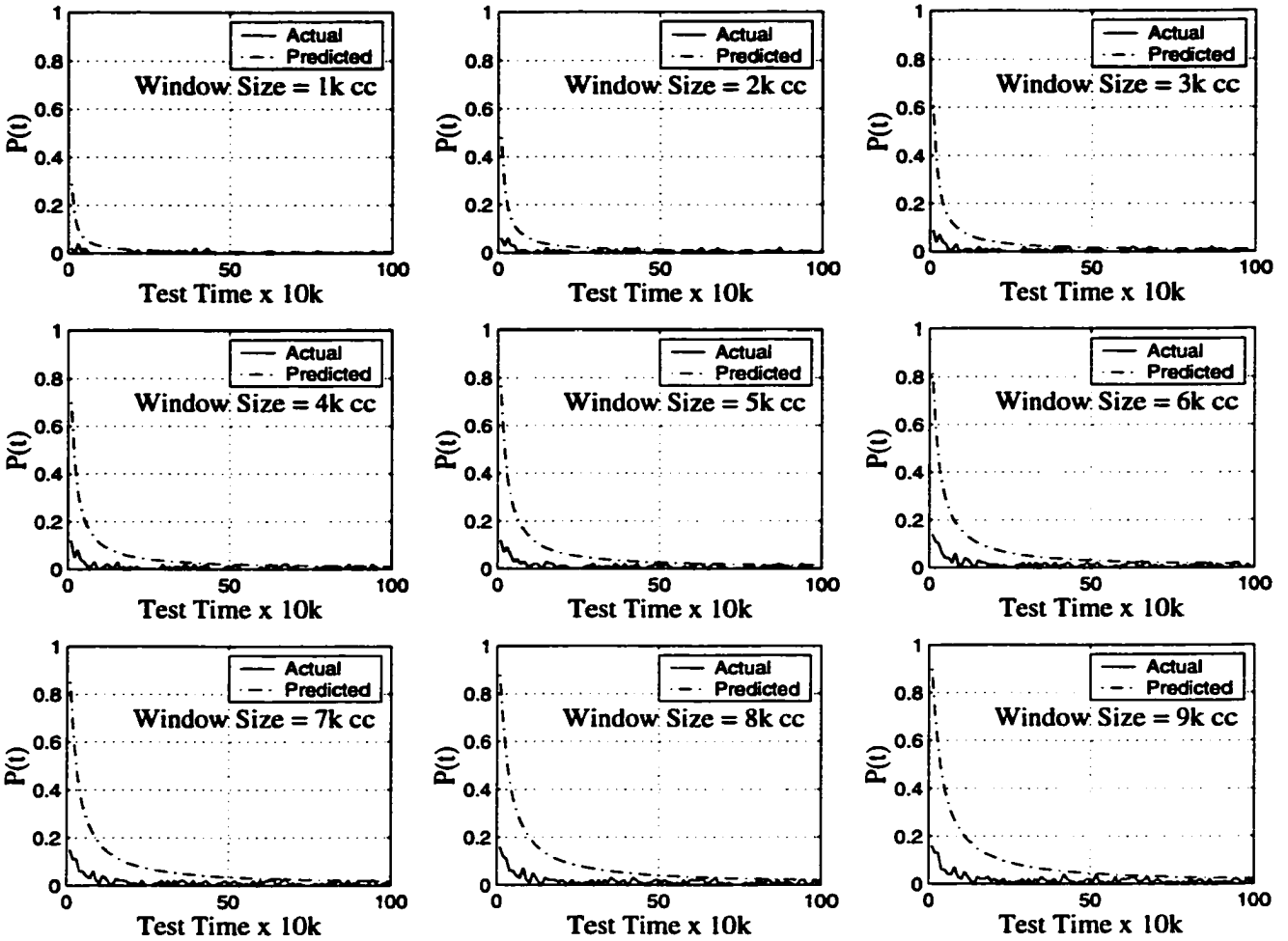


Figure F.7: Interruption Error for B09

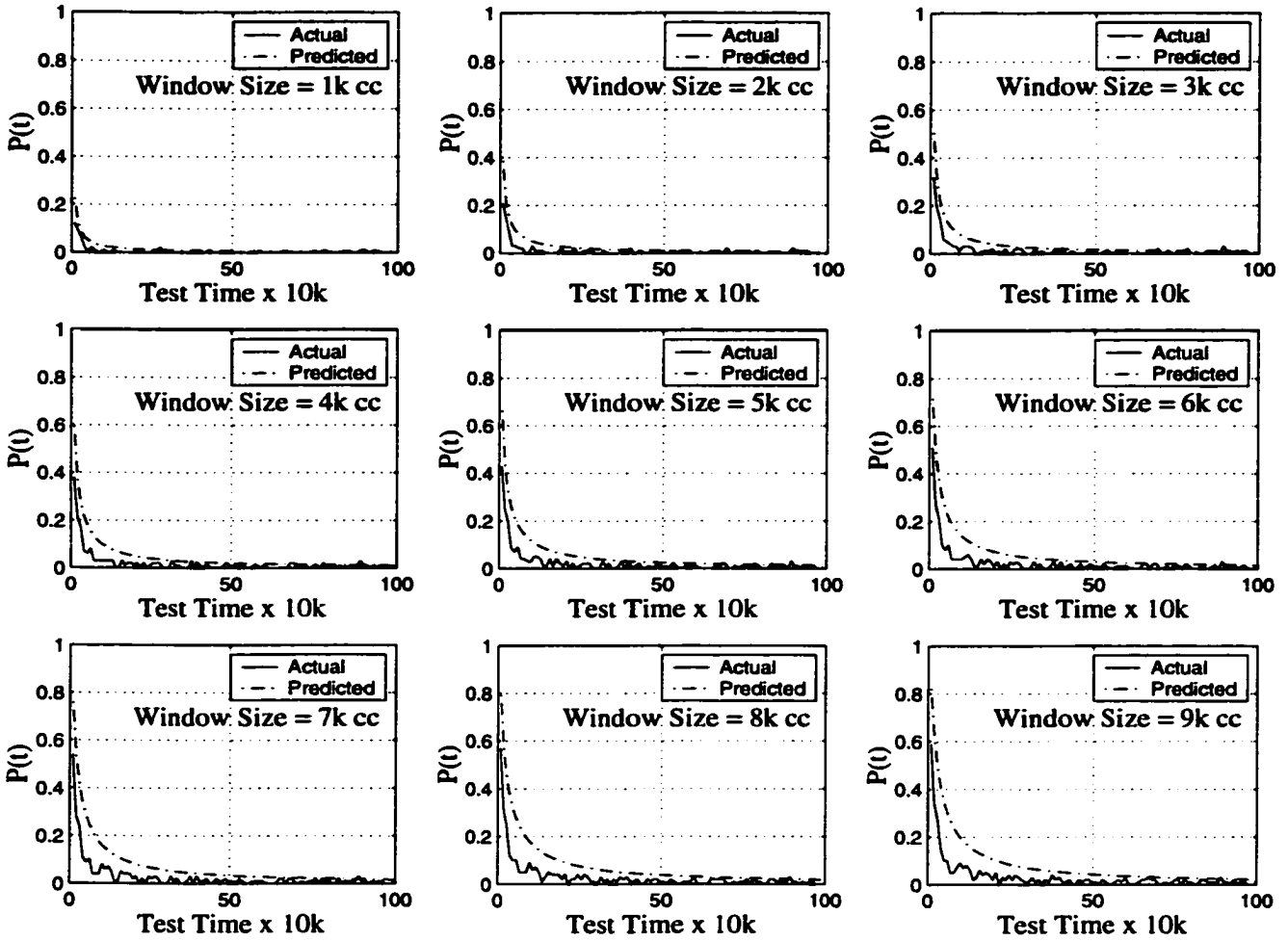


Figure F.8: Interruption Error for B10

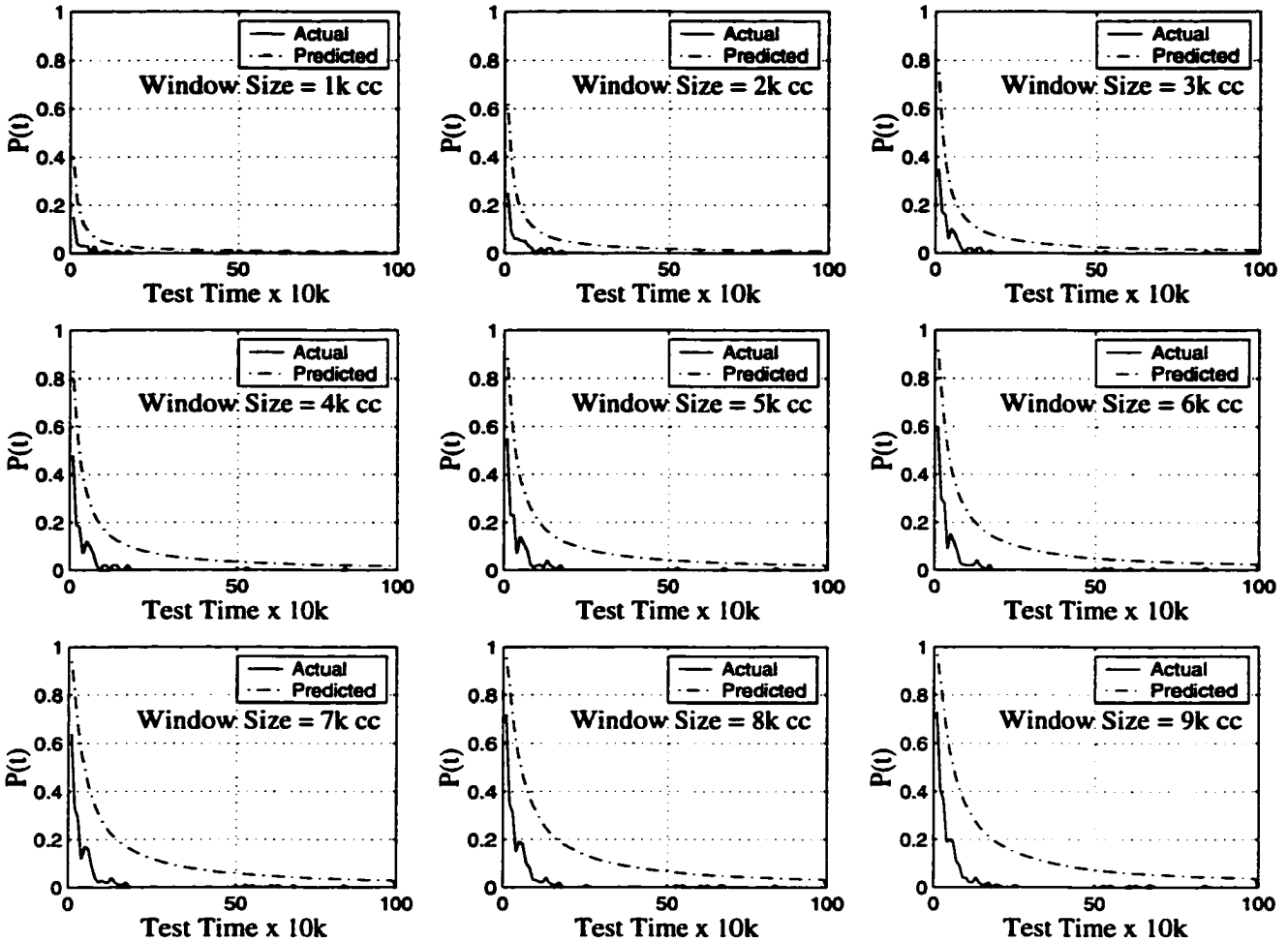


Figure F.9: Interruption Error for B11

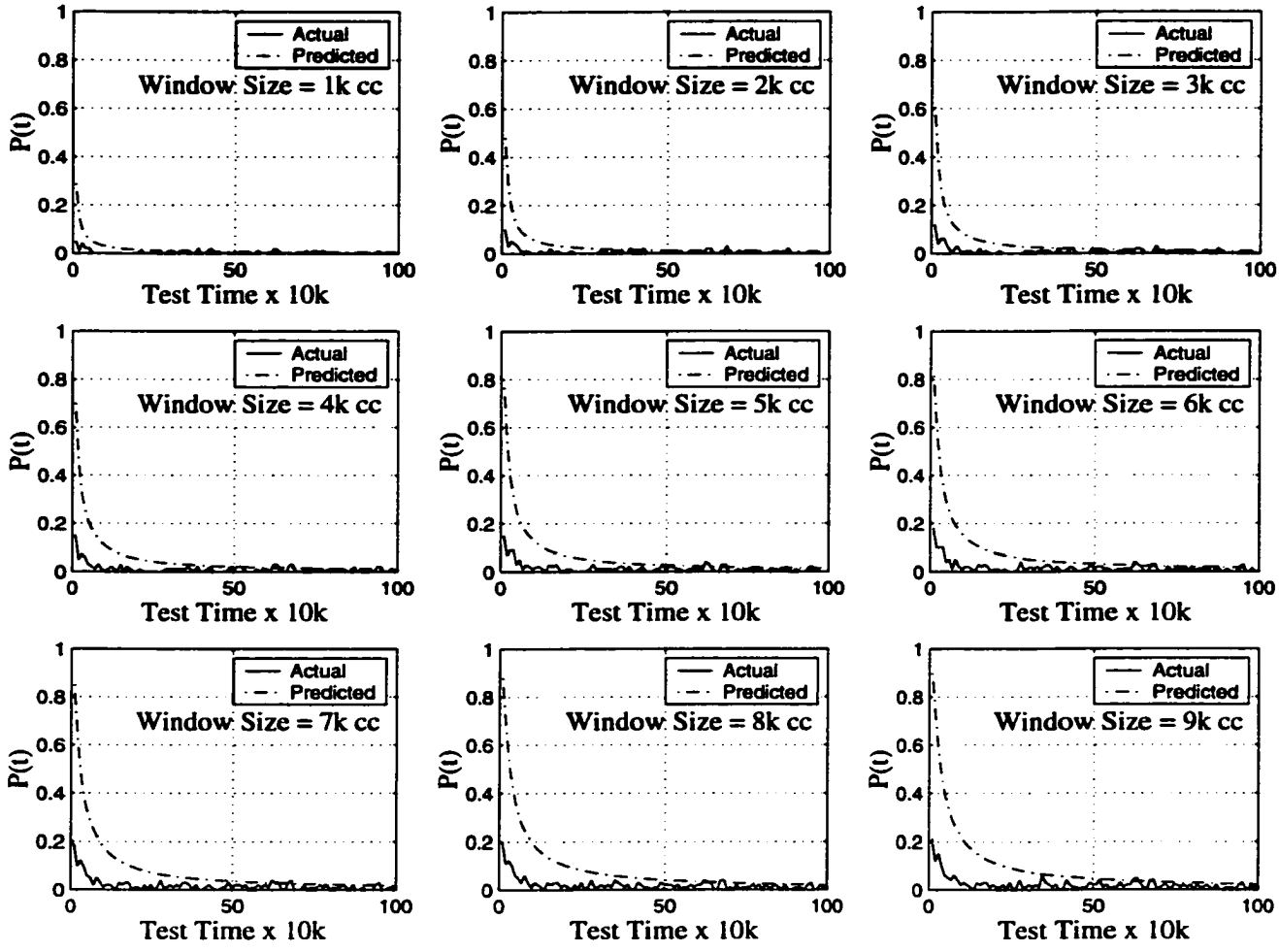


Figure F.10: Interruption Error for B12

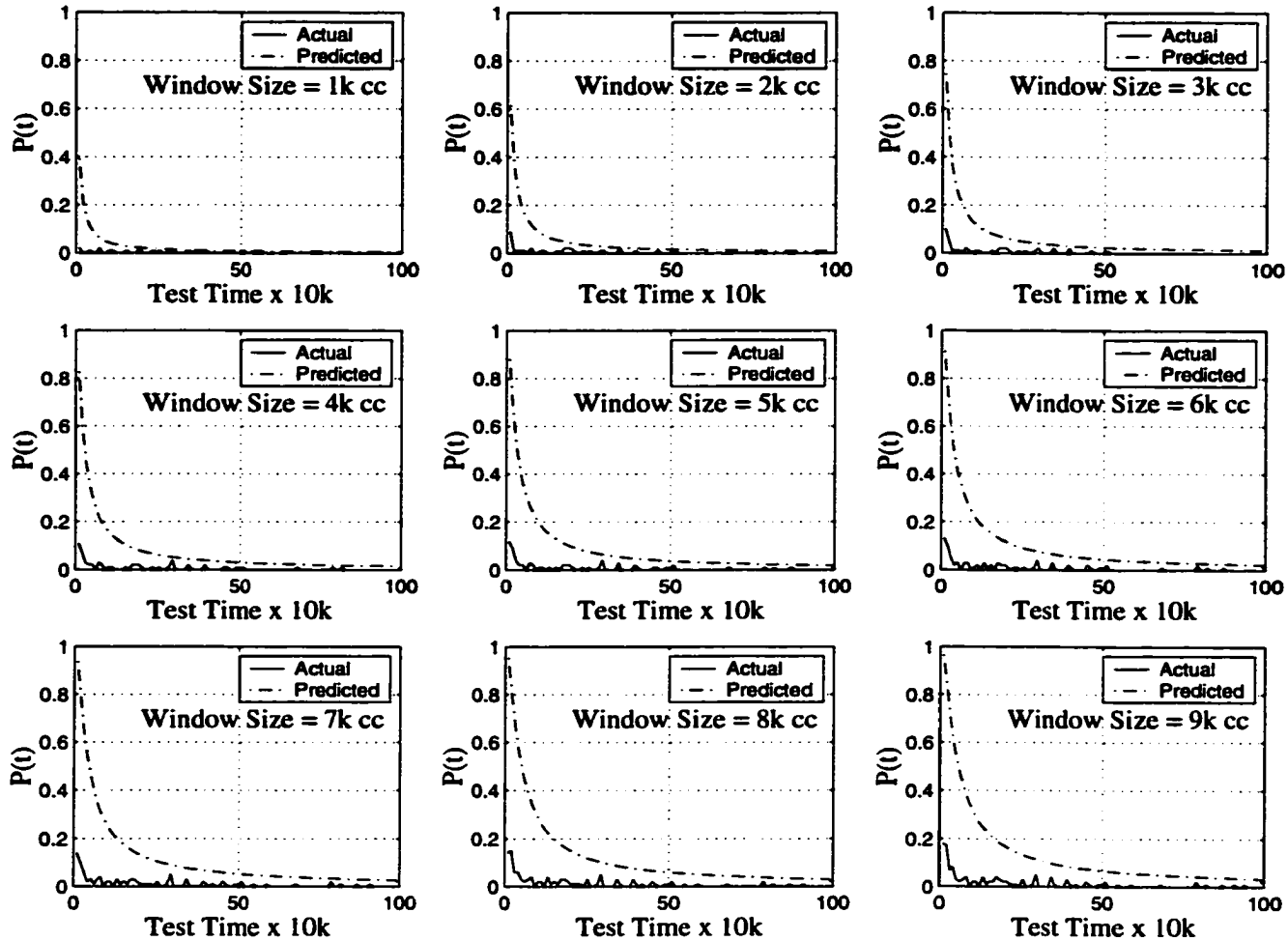


Figure F.11: Interruption Error for B14

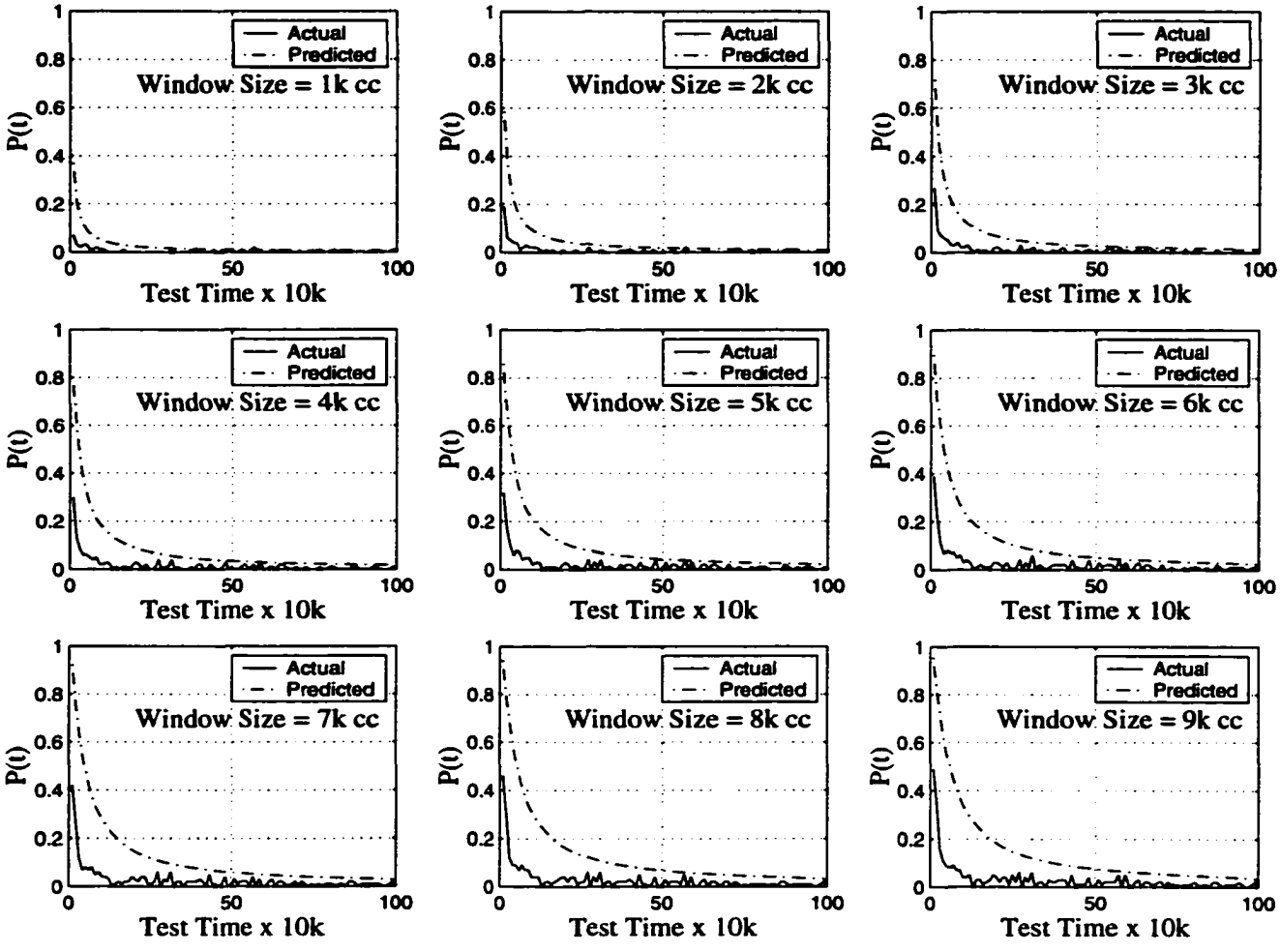


Figure F.12: Interruption Error for B15

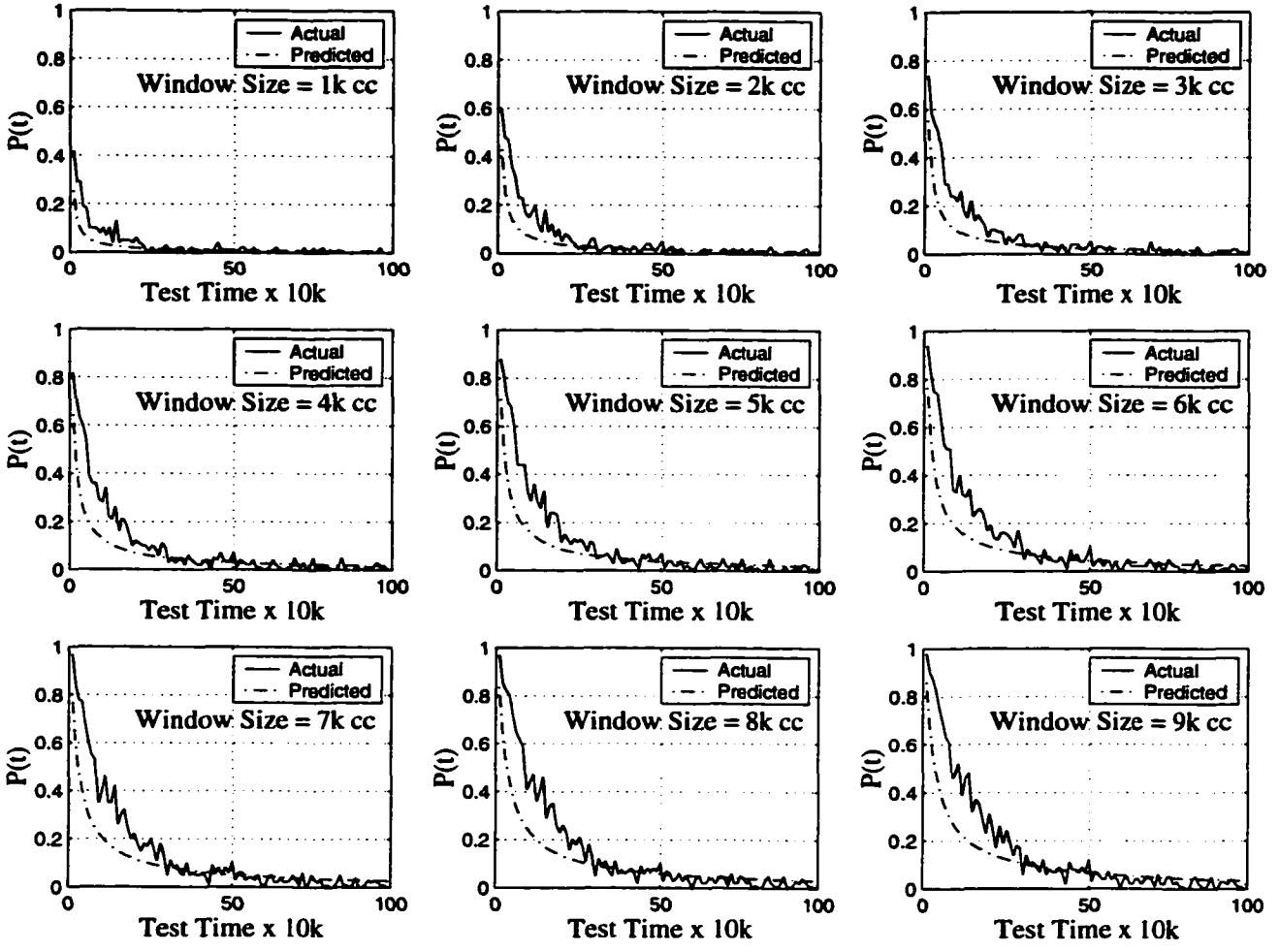


Figure F.13: Interruption Error for 8251

Appendix G

EXPECTED WAITING TIME ERROR VS WINDOW SIZE

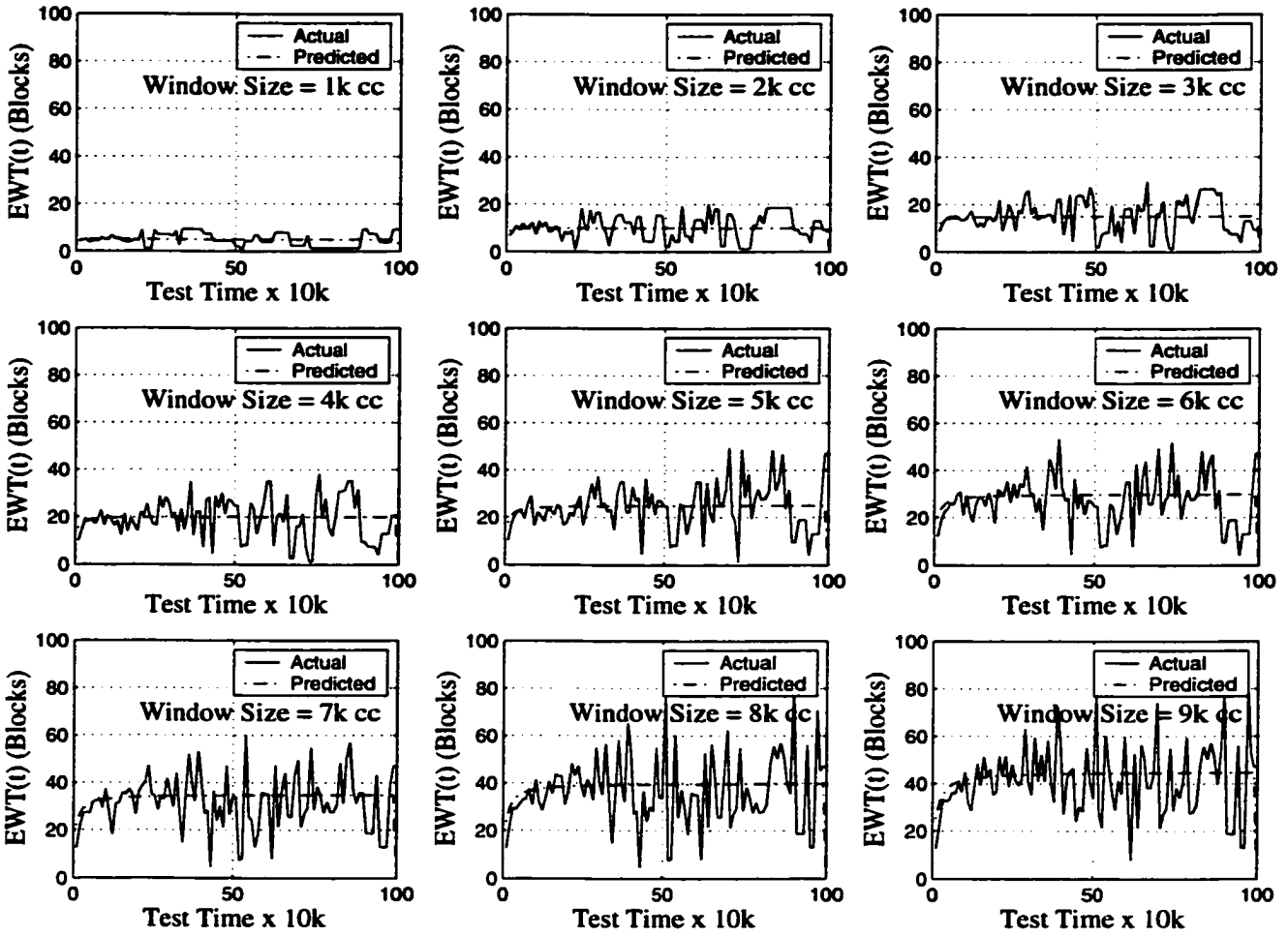


Figure G.1: EWT Error for B01

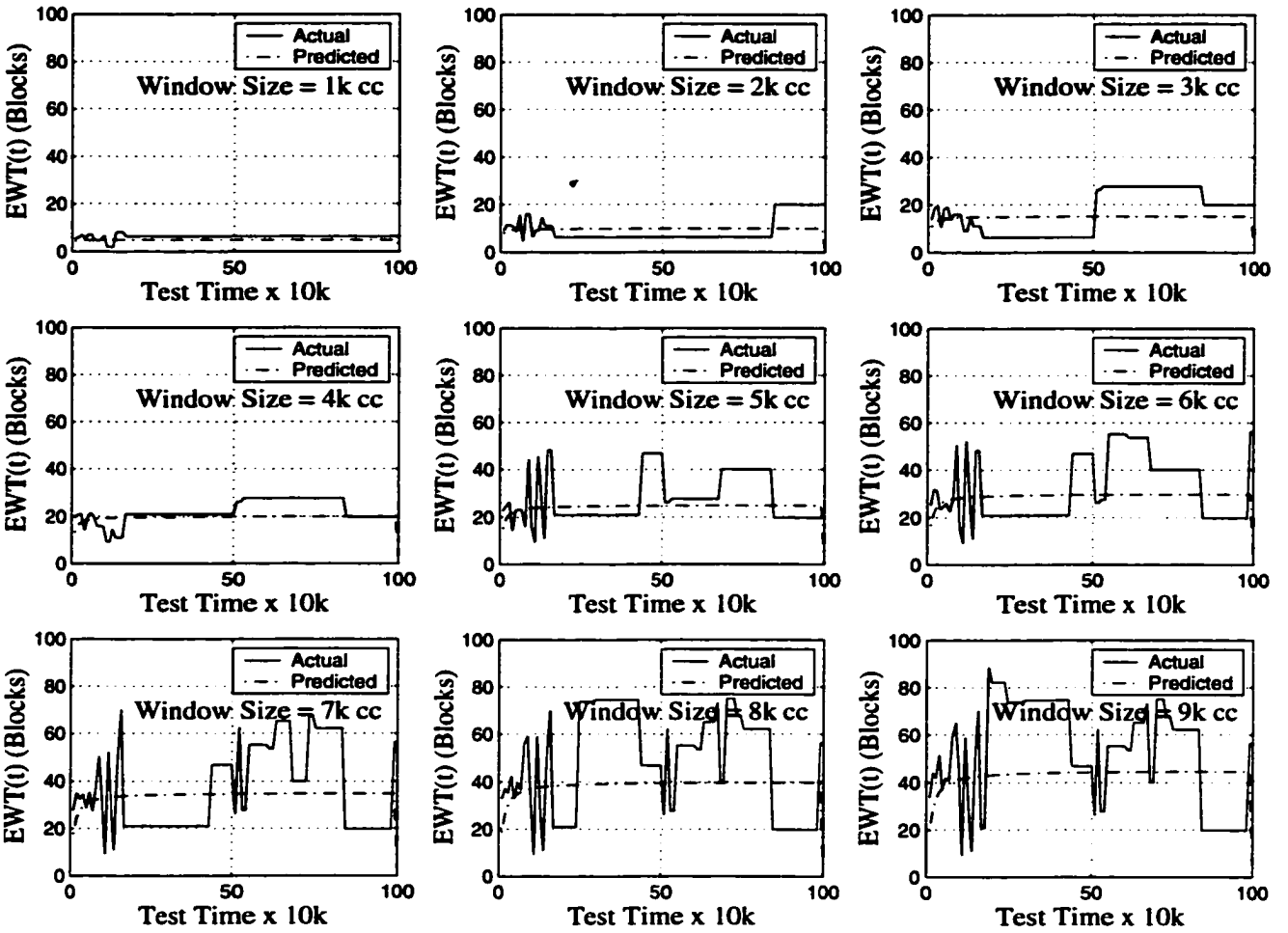


Figure G.2: EWT Error for B04

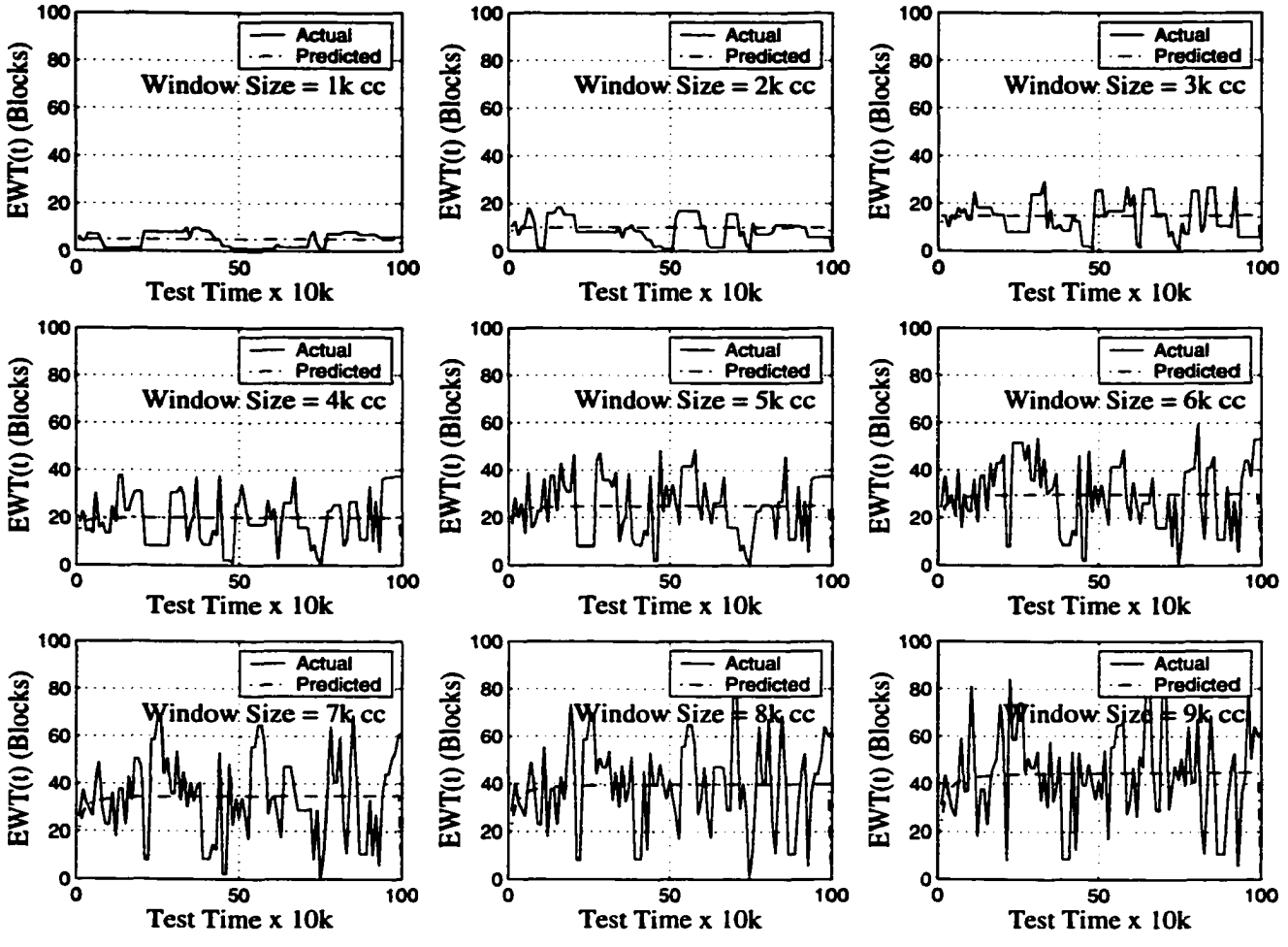


Figure G.3: EWT Error for B05

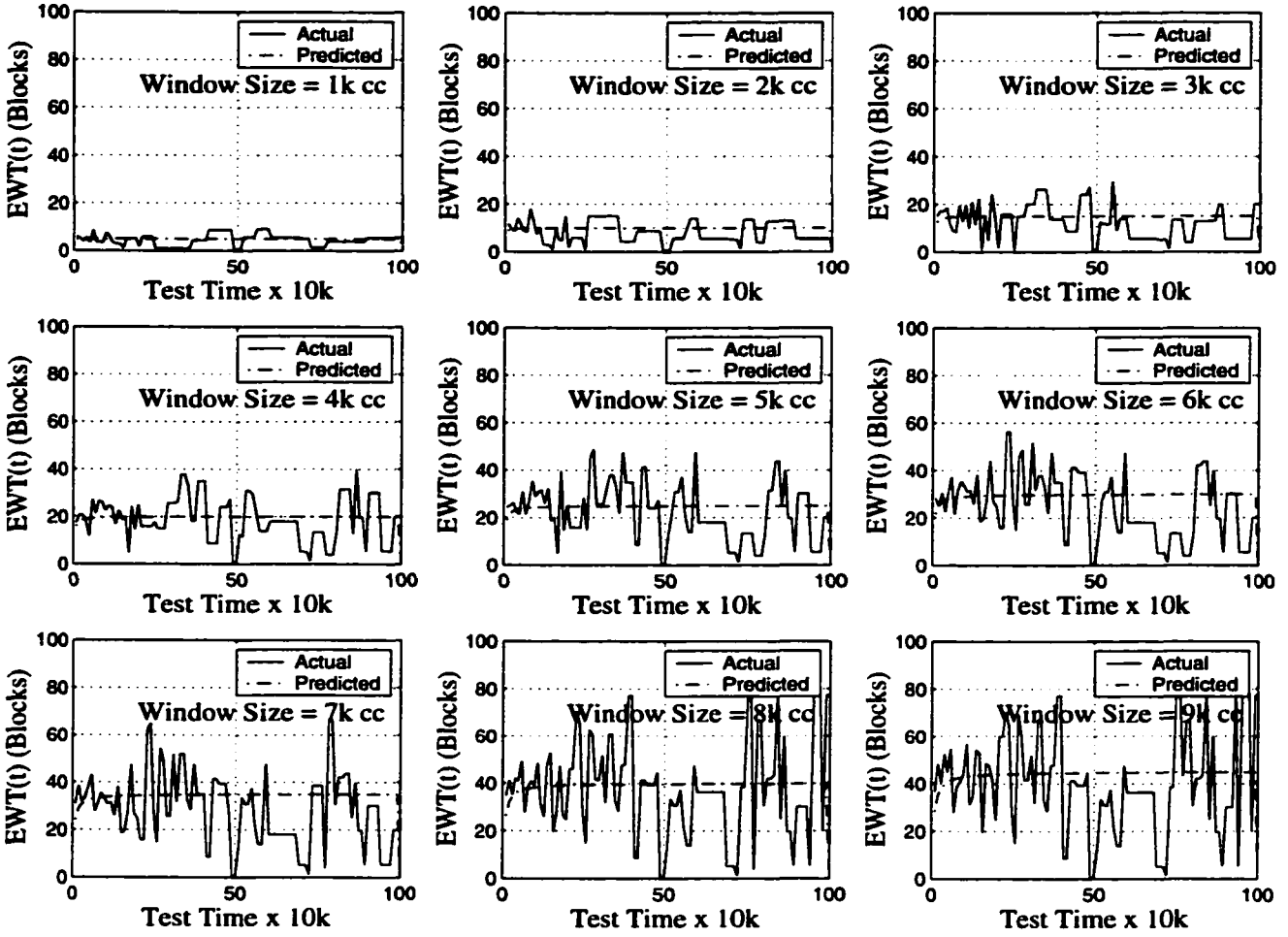


Figure G.4: EWT Error for B06

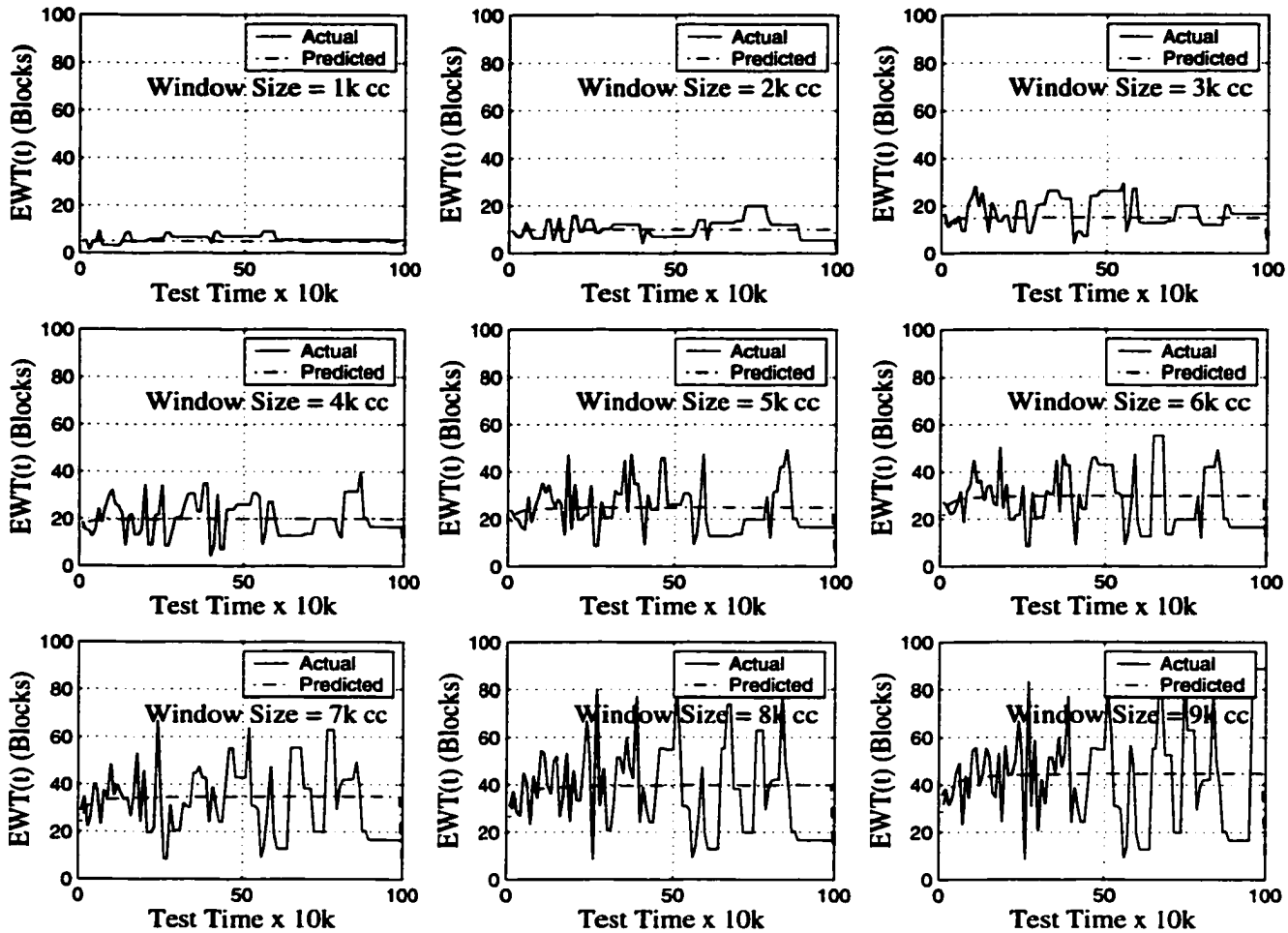


Figure G.5: EWT Error for B07

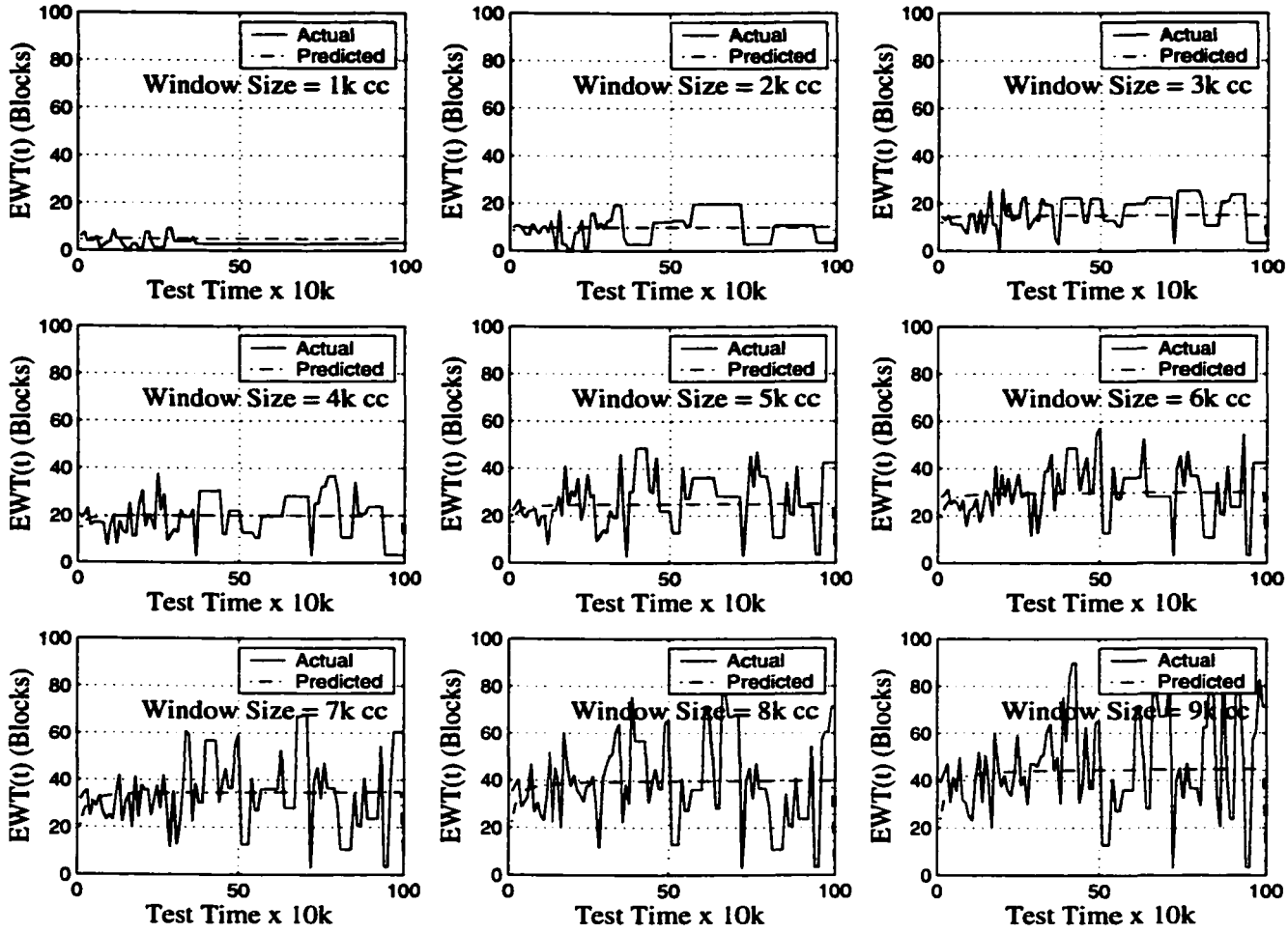


Figure G.6: EWT Error for B08

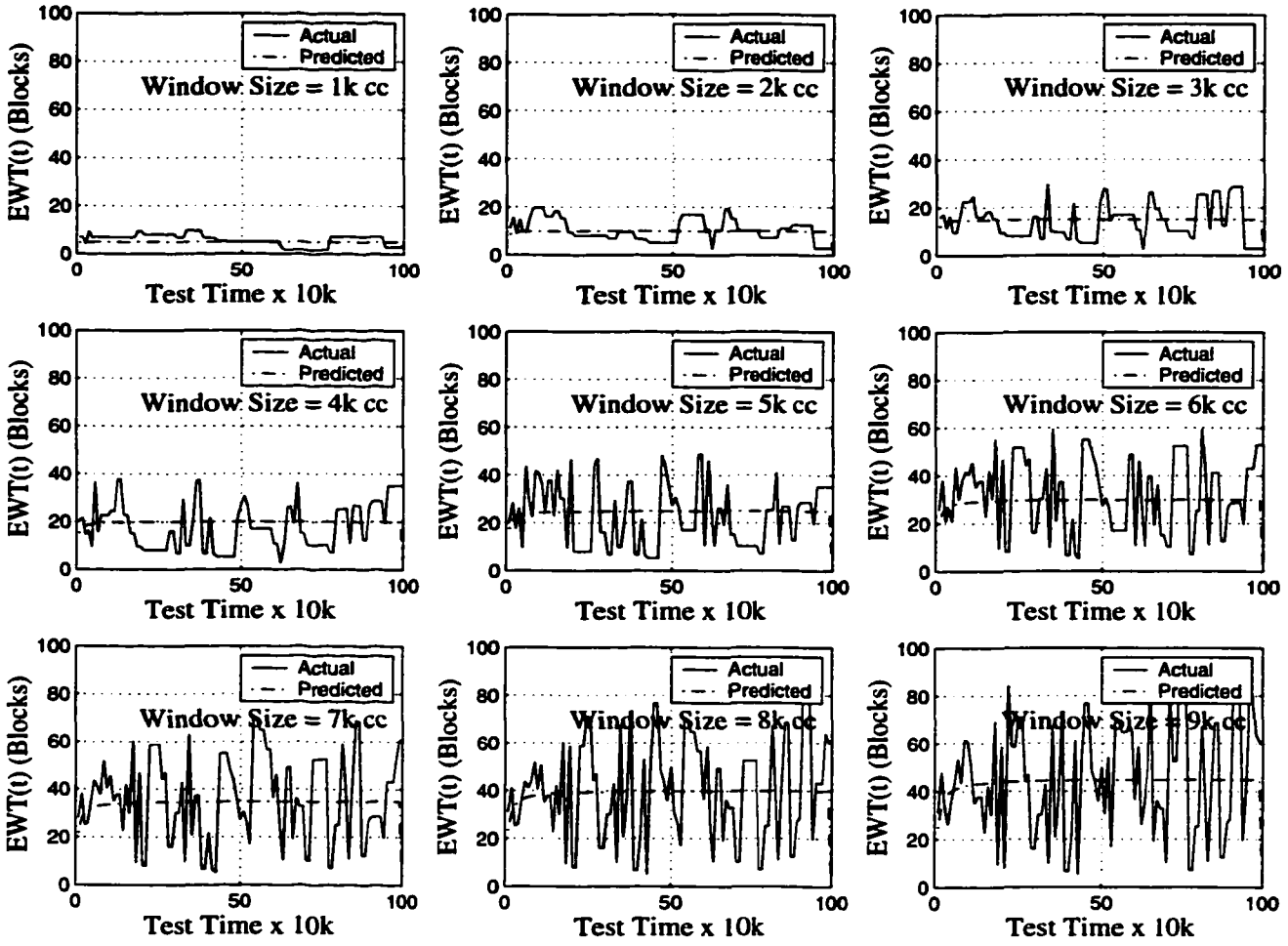


Figure G.7: EWT Error for B09

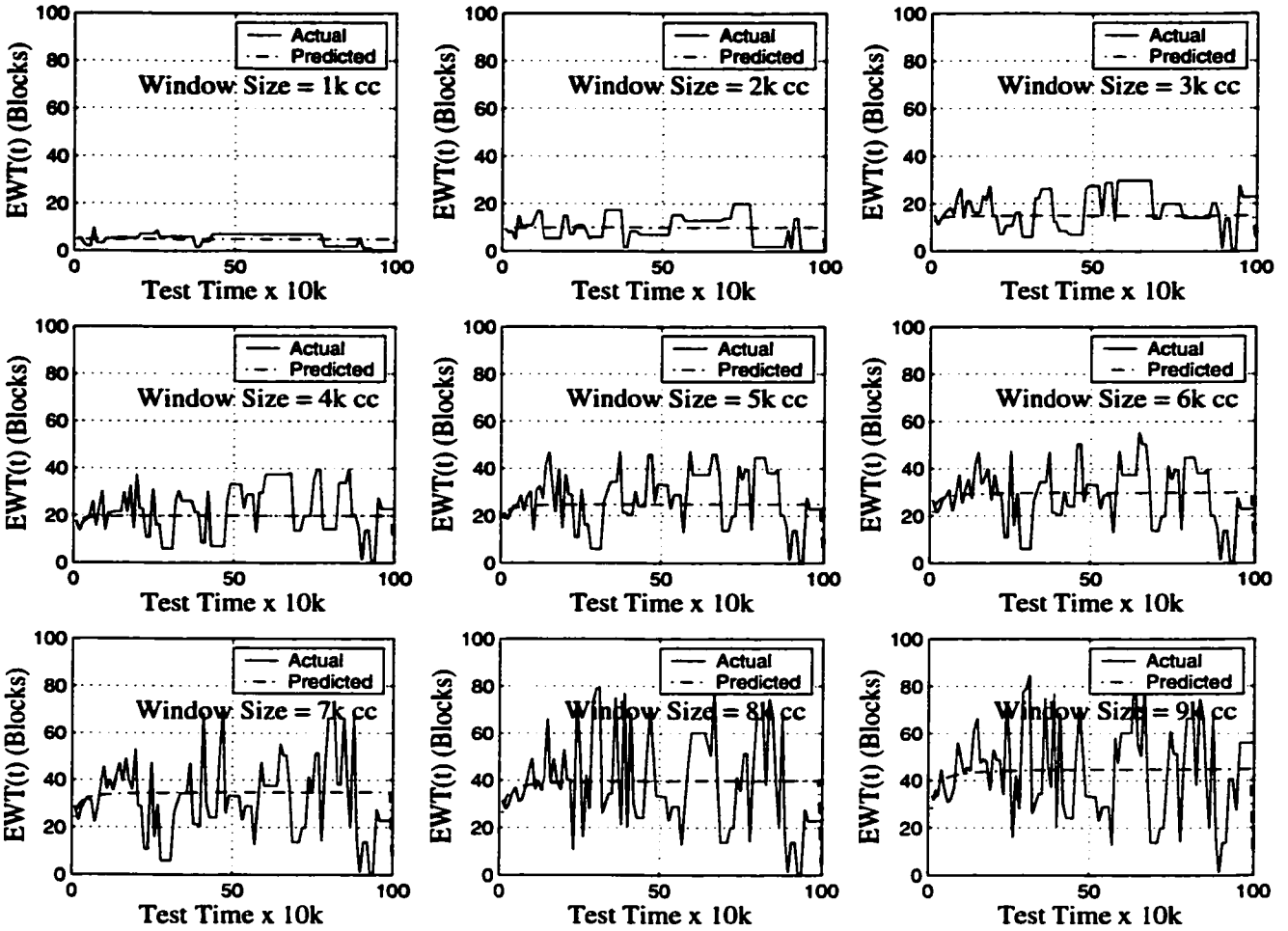


Figure G.8: EWT Error for B10

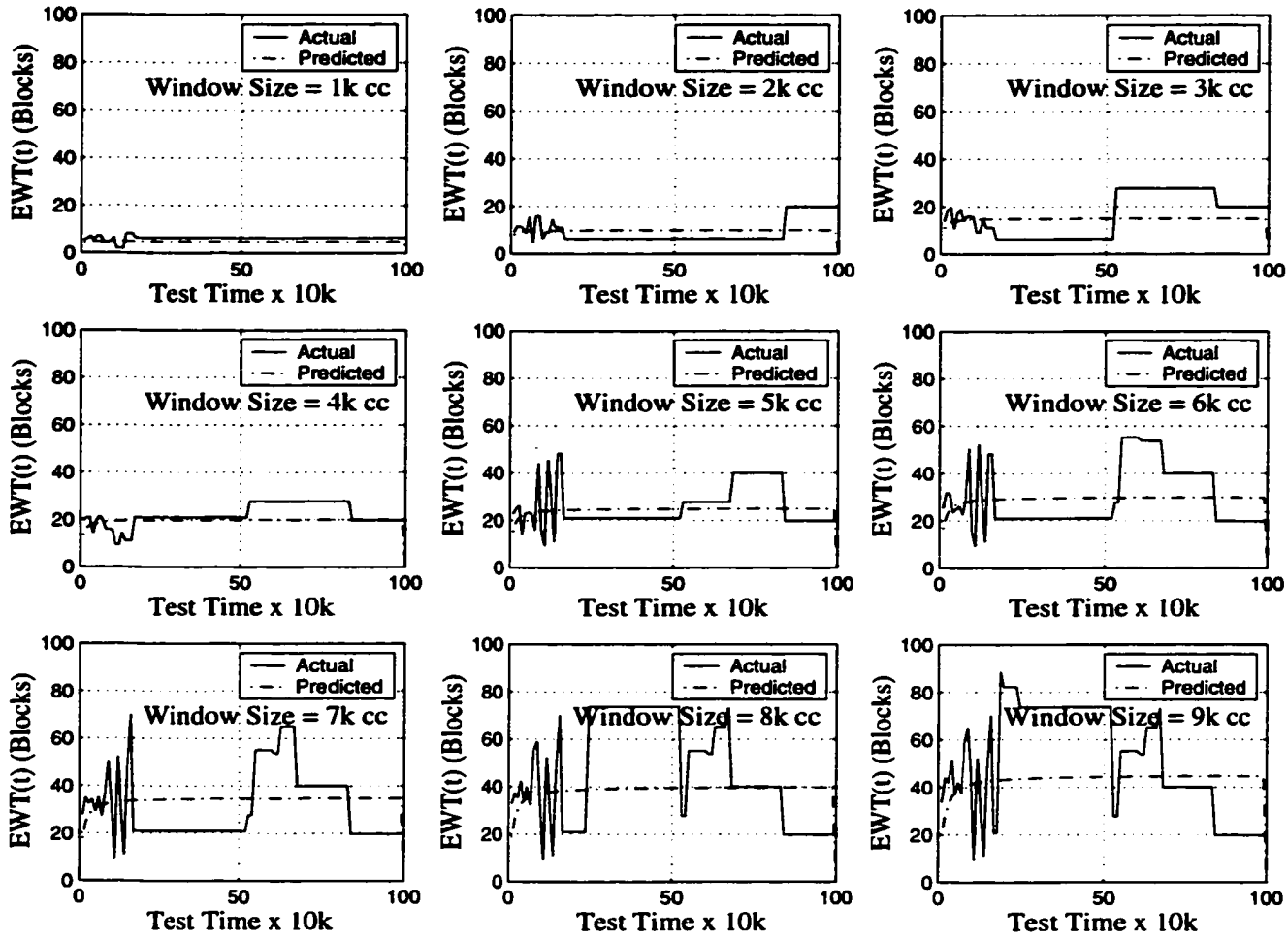


Figure G.9: EWT Error for B11

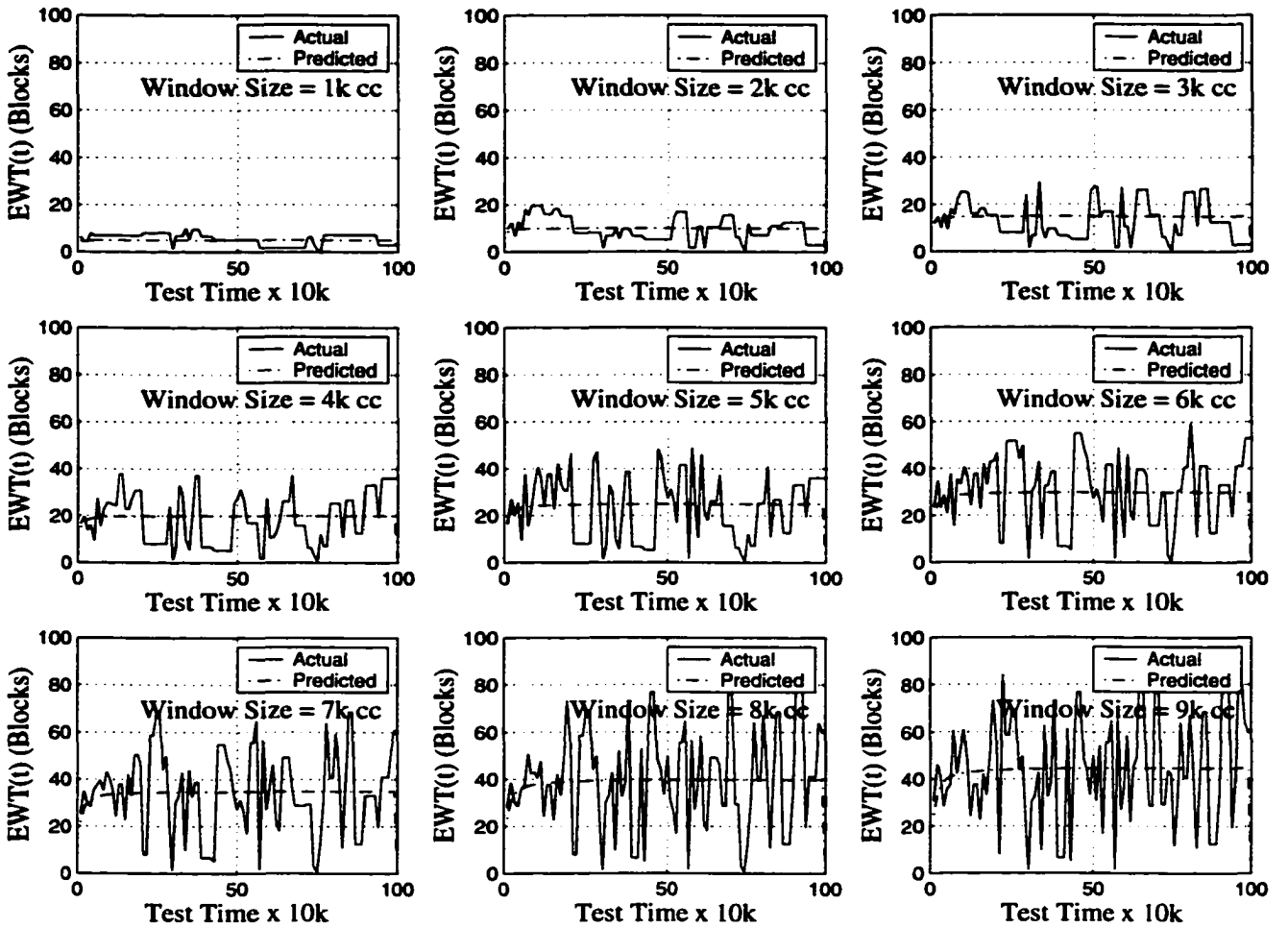


Figure G.10: EWT Error for B12

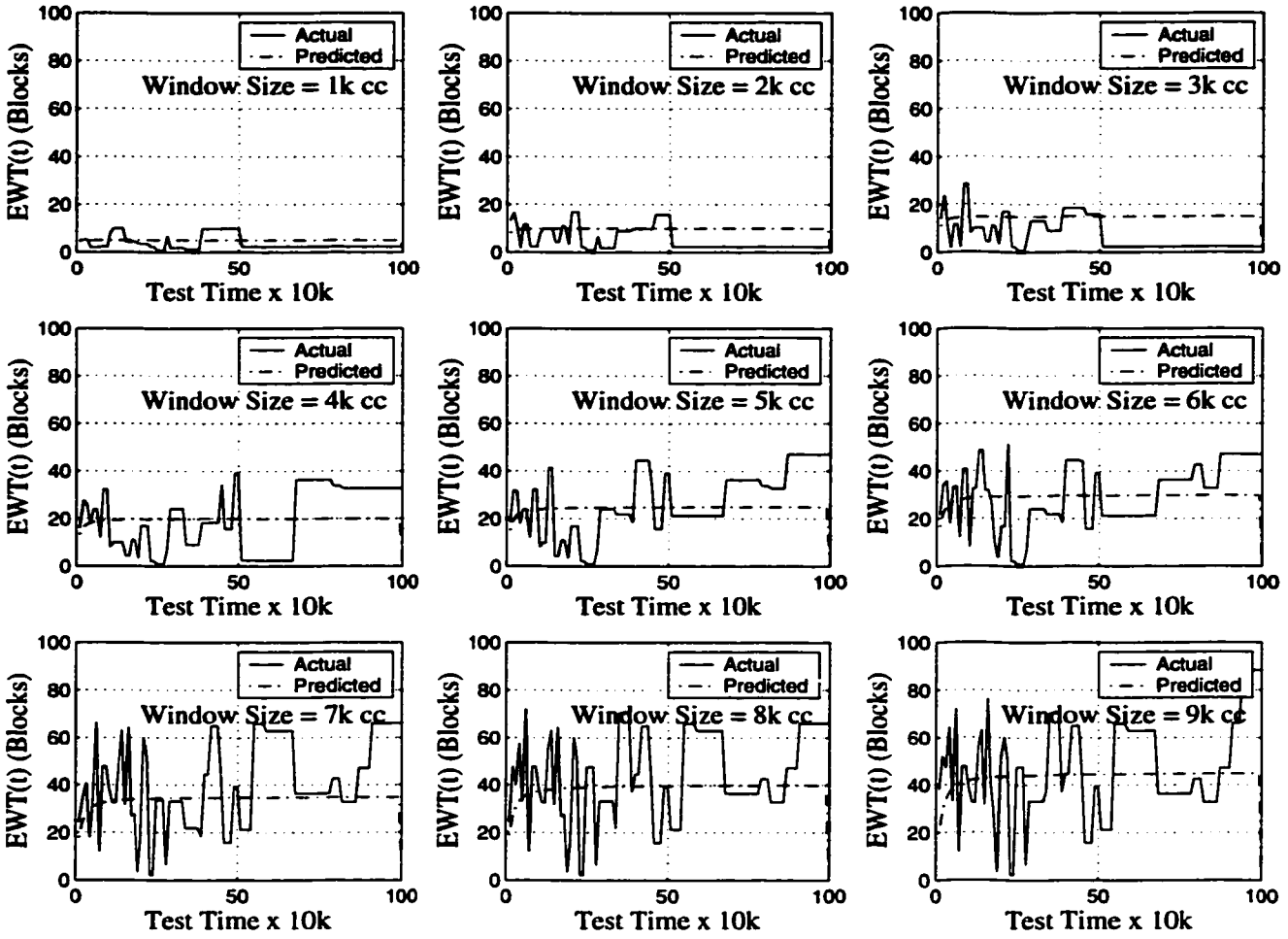


Figure G.11: EWT Error for B14

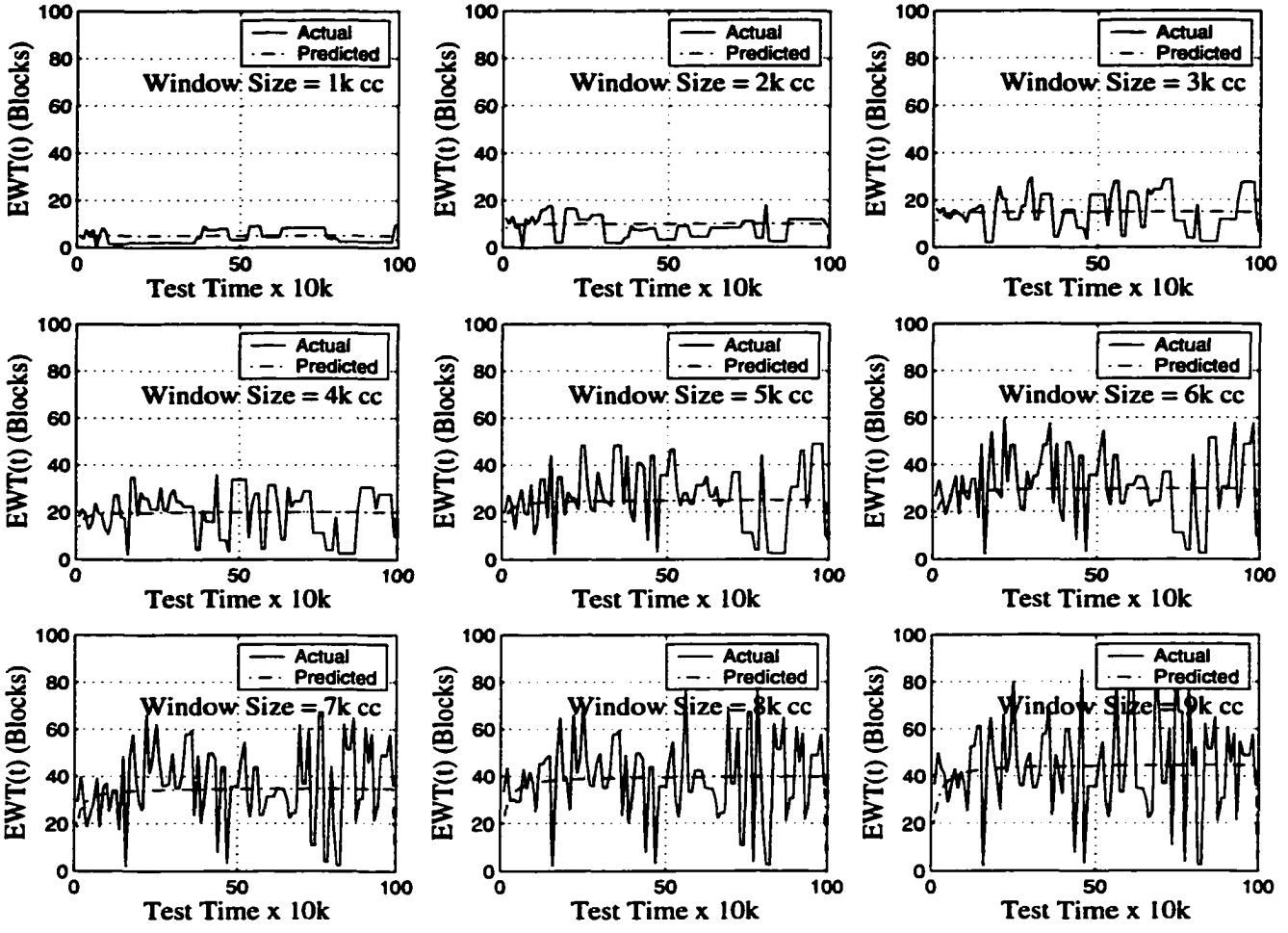


Figure G.12: EWT Error for B15

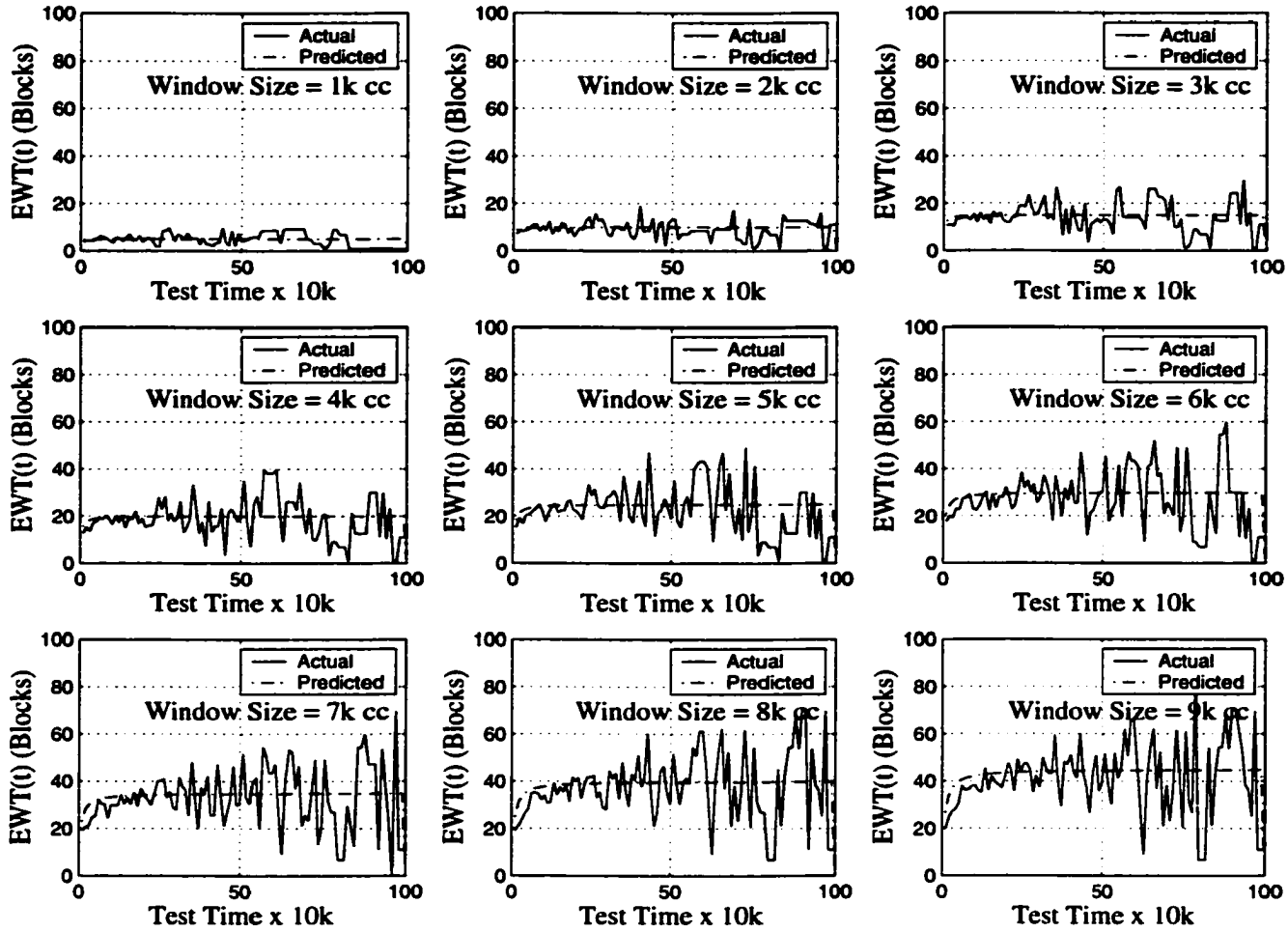


Figure G.13: EWT Error for 8251