AN ABSTRACT OF THE THESIS OF

David T. Hu for the degree of Master of Science in Chemical Engineering presented on September 11, 2003. Title: Fault Probability and Confidence Interval Estimation of Random Defects seen in Integrated Circuit Processing

Redacted for privacy

Abstract Approved:_

Milo D. Koretsky

Various methods of estimating the fault probabilities based on defect data of random defects seen in integrated circuit manufacturing are examined. Estimates of fault probabilities based on defect data are less costly than those based on critical area analysis and are potentially more reliable because they are based on actual manufacturing data. Due to limited sample size, means of estimating the confidence interval associated with these estimates are also examined. Because the mathematical expressions associated with defect data- based estimates of the fault probabilities are not amenable to analytical means of obtaining confidence intervals, bootstrapping was employed.

The results show that one method of estimating the fault probabilities based on defect data proposed previously is not applicable when using typical in-line data. Furthermore, the results indicate that under typical fab conditions, the assumption of a Poisson random defect distribution gives accurate fault probabilities. The yields as predicted by the fault probabilities estimated from the limited yield concept and kill ratio and those estimated from critical area simulation are shown to be comparable to actual yields observed in the fab. It is also shown that with in-line data, the *FP* estimated for a given inspection step is a weighted average of the fault probabilities of the defect mechanisms operating at that inspection step.

Four bootstrapped based methods of confidence interval estimation for fault probabilities of random defects are examined. The study is based on computer simulation of randomly distributed defects with pre-assigned fault probabilities on dice and the resulting count of different categories of die. The results show that all four methods perform well when the number of fatal defects is reasonably high but deteriorate in performance as the number of fatal defects decrease. The results also show that the BCA (*bias -corrected and accelerated*) method is more likely to succeed with a smaller number of fatal defects. This success is attributed to its ability to account for change of the standard deviation of the sampling distribution of the FP estimates with the FP of the population, and to account for median bias in the sampling distribution.

Fault Probability and Confidence Interval Estimation of Random Defects seen in Integrated Circuit Processing

by

David T. Hu

A THESIS

submitted to

Oregon State University

In partial fulfillment of
the requirements for the
degree of

Master of Science

Presented September 11, 2003
Commencement June 2004

Master of Science thesis of David T. Hu presented on September 11, 2003.

APPROVED:

Redacted for privacy

Major Professor, representing Chemical Engineering

Redacted for privacy

Chair of Department of Chemical Engineering

Redacted for privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for privacy

David T. Hu, Author

ACKNOWLEDGEMENTS

I would like to thank the following people for making the completion of this thesis possible.

- Dr. Milo Koretsky, for his support, encouragement, and tireless guidance while I was a student in the department and during the writing of this thesis.
- Manu Rehani, of LSI Logic, for introducing me to this topic and for his expert mentorship during my internship.
- Loan Pham, my wife, for her unwavering love and support of me while raising our newborn son, Patrick.
- Daniel Hu, my brother, who made sure I did not get discouraged when the going got tough.
- Finally, to my parents, whose unconditional love made everything possible.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

NOMENCLATURE

a	a distance in the sampling distribution
a_c	acceleration constant in BCA method
A_{Die}	die area
α_A	cluster factor of defect type A
b	a distance in the sampling distribution
B	bias of estimator
B_R	bootstrap estimate of bias
β_o	bias constant in BCA method
CI	confidence interval
DD_A	defect density of defect type A
FP	Fault probability
FP_{true}	FP parameter of population
FP_A	one of the three values of FP_{true} assigned in each run
FP_B	one of the three values of FP_{true} assigned in each run
FP_C	one of the three values of FP_{true} assigned in each run
FP_{LY}	FP estimated using the limited yield equation with the yield given by the kill ratio equation
FP_{Ross}	FP estimated using the Ross method
FP_{samp}	point estimate of FP_{true}
FP_{sim}	FP estimated using critical area analysis
$FP*$	bootstrap estimate of FP_{samp}
$h(x)$	probability density function of defect size x
IC	integrated circuit
ISEF	island etch
KR	kill ratio
λ	average number of fatal defects per die
LY_A	limited yield of defect type A
$m(\,)$	a transformation function
M1EF	metal 1 etch
M1MD	metal 1 mask
N	number of dice with one defect
n	fatal defects per unit area
N_F	number of failed dice with one defect
$\hat{\phi}$	transformed estimator
ϕ	transformed parameter
P$\{E\}$	probability of event E
POLF	poly etch
R	total number of bootstrap estimates
σ_ϕ	standard deviation of $\hat{\phi}$
s	estimate of standard deviation

NOMENCLATURE (Continued)

SD	standard deviation
$\hat{SD}_{boot}$	standard deviation of the bootstrap-estimated FPs
$\hat{SD}_{act}$	estimate of the SD of the actual sampling distribution of FP_{samp} based on actual values of FP_{samp}
SL2R	salicide formation
SPCE	spacer etch
T	total number of dies sampled
t	point estimate of θ
t^*	bootstrap estimate of t
t^*_α	100α percentile of bootstrap estimates
T_i	total number of dice with defect type i on them
T_G	total number of good dice sampled
T_{Gi}	total number of good dice with defect type i on them
TN1T	Ti-Nitride deposition 1
θ	population parameter
$\hat{\theta}$	point estimator of θ
V_R	sample variance of the bootstrap estimates
$\overline{x}$	estimate of the mean
Y_{DLi}	defect limited yield
Y_R	random yield
Y_S	systematic yield
z_α	standard normal variable having cumulative probability α

Fault Probability and Confidence Interval Estimation of Random Defects seen in Integrated Circuit Processing

1. INTRODUCTION

1.1 Overview of the Thesis

Because of the importance of yield estimation in integrated circuit (IC) manufacturing, the research in this area has been extensive. Most of the research, however, has focused on estimation of fault probabilities based on critical area analysis. The goal of this thesis is to come up with a comprehensive approach based on measured defect data to find the fault probabilities of individual steps in an IC manufacturing line. Estimates of fault probabilities based on defect data are less costly than those based on critical area analysis and are potentially more reliable because they are based on actual manufacturing data instead of just the die layout. Chapter 2 compares the various methods of estimation of fault probabilities using actual defect data to discover which method is most reliable. A way of accounting for defect clustering in the estimation is also discussed.

As a consequence of the typically limited sample size of the defect data used for estimating fault probabilities, means of assessing the uncertainty associated with these estimates are also examined. This uncertainty is best expressed as a confidence interval. Because the mathematical expressions associated with defect data- based estimates of the fault probabilities are not amenable to analytical means of obtaining confidence intervals, bootstrapping was employed. Chapter 3 examines different bootstrap based methods of obtaining confidence intervals for these estimates of fault probability.

Finally, based on the results of estimating the fault probabilities from the actual defect data, recommendations for achieving more accurate and complete detection of defects occurring on the manufacturing line will be made. Recommendations for the sample size needed for accurate CI estimates are also given based on the results from bootstrapping.

A simple analogy to basic engineering statistics summarizes the goals of this thesis. To obtain an unbiased estimate for the mean of a normal population, we would use

$$\bar{x} = \sum_{i=1}^{n} x_i / n .$$

To obtain its *1-2*α*%* confidence interval, we have:

$$\bar{x} \pm \left(s / \sqrt{n}\right) \cdot t_{\alpha},$$

where s is the sample standard deviation, n is the number of observations in the sample, and t_{α} is the α percentile of the t distribution. The goal of this thesis is to obtain the analogous expressions for the fault probability of the population of a certain class of defects *based on defect data* measured in the IC fab.

1.2 Integrated Circuit Yield Analysis

The estimation of yield is critical for economically efficient production of integrated circuits. In a newly implemented process, many yield detractors need to be identified. This can only be done if methods of estimating the yield loss from these yield detractors have been implemented. When the process is mature, even small improvements in yield can greatly increase profitability. It has been estimated that in a typical fab with an output of 20,000 wafers per month, if yield is improved by even 2%, profit could go up by about $10 million per year. The reason for this steep increase in profit is that once a process is mature the cost of manufacturing ICs is approximately constant whatever the yield. [1]

The total yield at wafer probe is typically viewed as made up of two components: the systematic yield, and the random yield. The systematic yield loss is due to 1) design problems, such as a design not meeting minimum spacing rules, 2) process steps not meeting specifications, such as photo-mask misalignment and under etching, and 3) faulty testing procedures. These are usually issues in the early development of the IC manufacturing process. Once a process is mature, the main yield loss is due to the random defects.

Random defects are caused by random events, such as particle deposition, that occur during the fabrication process. They mostly stem from the processing equipment and processing material and can cause the formation of features on the die not intended in the design layout. If they occur in a critical region, or critical area, of the die, they will cause opens or shorts, or some other type of fatal defect, and cause the chip to fail. [2] A scanning electron micrograph of a random defect causing a short is shown in Figure 1.1.

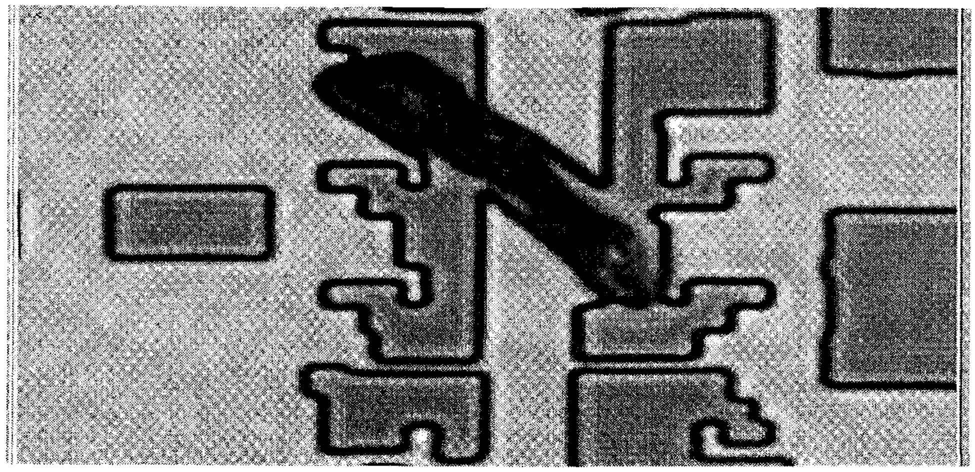


Figure 1.1. Scanning electron micrograph of a random defect causing a short between metal lines

Each type of random defect has a probability of causing a fault associated with it. This probability, the fault probability (*FP*), is simply the ratio of the critical area associated with that particular defect type to the total area of the chip. It is a function of parameters associated with both the defect itself and the layout of the die: the size and type of the defect, and the circuit geometry. [2]

To predict random yield the values of the fault probabilities associated with each defect type must be established. In general, there are two ways to establish these fault probabilities. One is by analyzing existing defect data and inferring the fault probabilities through models of the defect distribution. This method is sometimes referred to as "data mining" because it depends heavily on the ability to extract information from the database associated with the defect maps of the particular defect type. The second method is to simulate the defect distribution by means of Monte Carlo techniques on the die layout, and determine the fault probabilities from the number of failed circuits. This method is referred to as critical area analysis [3]. An example of a die layout with the critical areas for bridges and breaks determined is shown in Figure 1.2. A bridge is the unintentional linking of two layers, while a break is the unintentional break in a layer, where the layer can be conductive, such as metal or polySi, or nonconductive, such as the field oxide separating the active regions of a transistor.

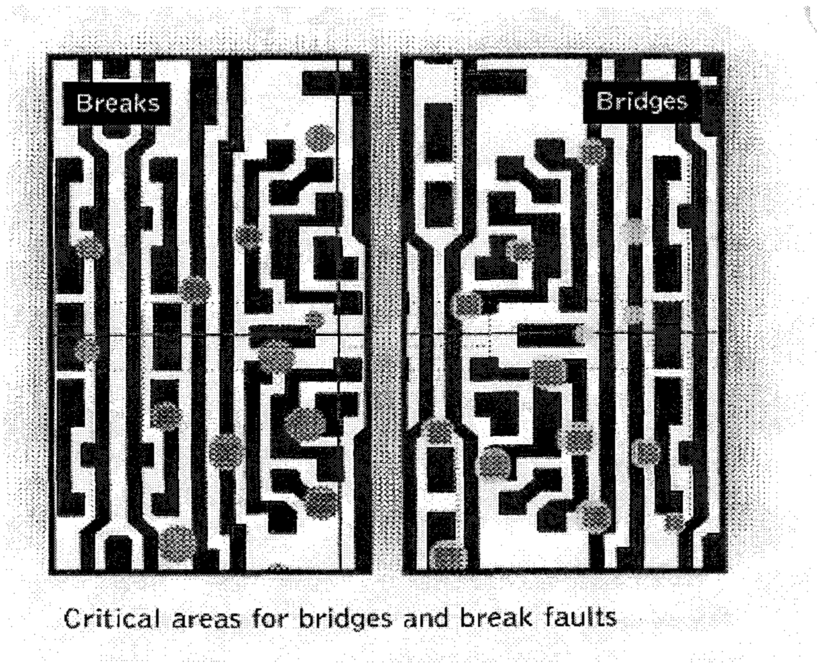


Figure 1.2. Example of critical area analysis done by HPL Inc. in which the critical areas for bridges and breaks have been determined

1.3 Defect Limited Yield

The defect limited yield for a particular defect type can be defined as the yield that would result if that particular defect type were the only defect present. In actual manufacturing, of course, there is usually more than one defect type, and, assuming the defect types occur independently of one another, the overall yield can be computed by:

$$Y = Y_S \cdot Y_R = Y_S \cdot Y_{DL_1} \cdot Y_{DL_2} \cdot Y_{DL_3} \cdot \ldots = Y_S \cdot \prod_i Y_{DL_i} \qquad (1\text{-}1)$$

where Y_s is the systematic yield, Y_R is the random yield, and Y_{DL_i} , i=1,2,3,... are the defect limited yields for defect types i. [4] For example, if no other defects are present other than defect type 1, and the systematic yield is 1, then $Y = Y_{DL_1}$; i.e., the resulting yield would be equal to the defect limited yield for defect type 1, and is limited by the yield loss due to this defect.

To estimate Y_{DL_i} , a distribution model for the defects is assumed. All the different yield equations result from different assumptions of the distribution of the defect density. Assuming the defects are distributed such that they have equal probability of occurring anywhere on a wafer, it can be shown from a binomial probability model that the probability of finding n fatal defects in a unit area, e.g., a chip area, A_{Die}, of region of constant defect density D is given by,

$$p(n;\lambda) = \frac{(\lambda)^n e^{-\lambda}}{n!} \qquad (1\text{-}2)$$

where

$$\lambda = FP \cdot A_{Die} \cdot D \qquad (1\text{-}3)$$

and *FP* is the fault probability. Since the fault probability may be defined as the portion of defects which are fatal, λ represents the average number of fatal defects per chip. [5] Equation (1-2) is the simplest distribution to assume and is known as the *Poisson* distribution. Defining the random yield for a particular type of defect, Y_{DLi}, as the probability of zero fatal defects, n=0, Equation (1-2) gives,

$$Y_{DL_i} = e^{-\lambda_i} \qquad (1\text{-}4)$$

As the die area is known, the determination of the fault probability and the defect density for each defect type per layer is the primary task in estimating the random yield.

1.4 Process steps and some common defects

Typically there are more than 200 steps in the manufacture of an integrated circuit. Figure 1.3 presents a flow diagram of twenty-seven major process steps that are involved in the fabrication of a typical device; however, the last eight steps are repeated for each metal layer grown. Additionally, the location of the inspection steps relative to these process steps are shown. Seven inspection steps are shown: ISEF, POLF, SPCE, SL2R, TN1T, M1MD, M1EF. The defect data we collected, which are

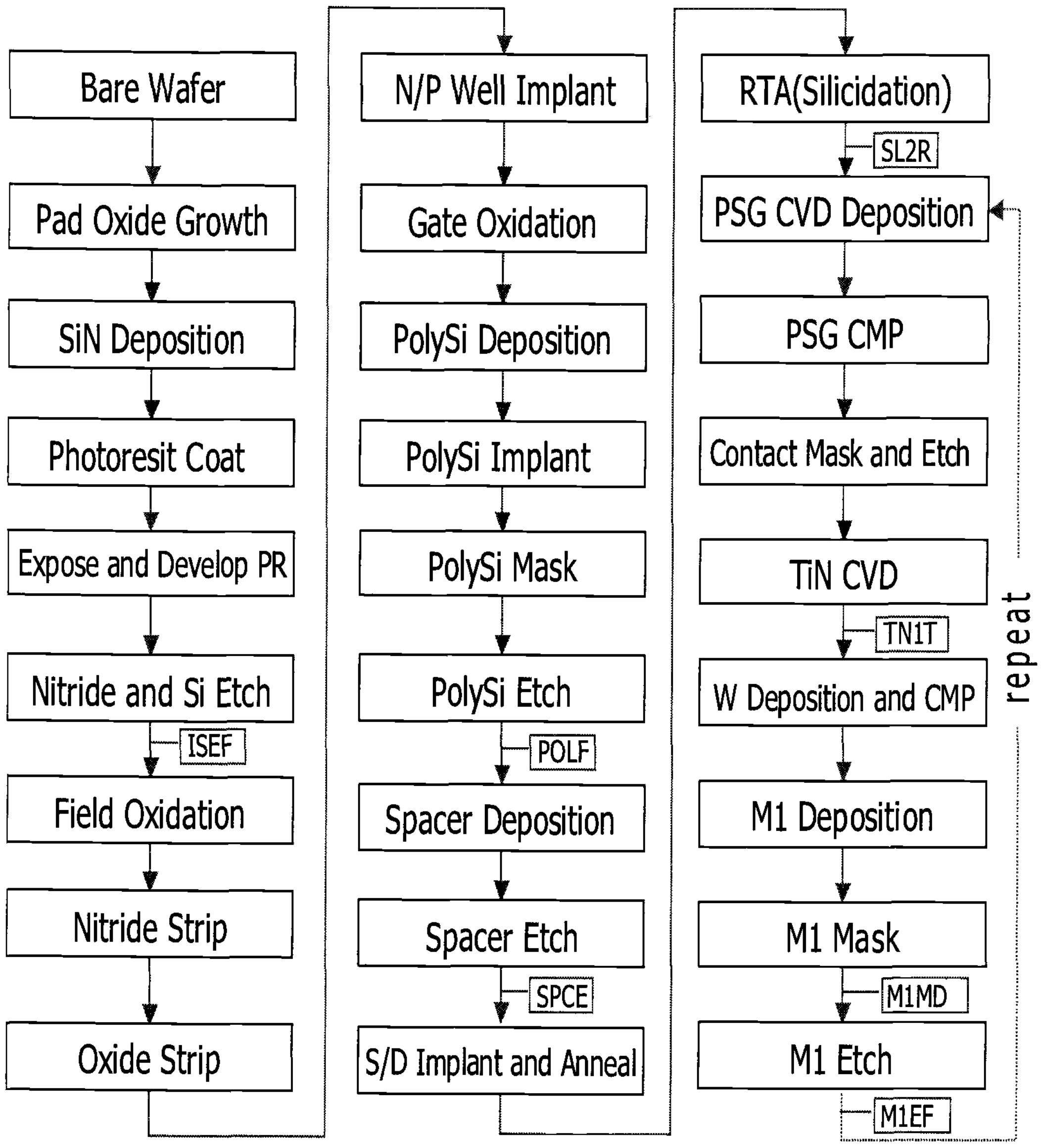


Figure 1.3 Process flow chart of device fabrication and location of the various inspection steps

presented in Chapter 2, were gathered using these inspection steps. The processes shown allow a device containing transistors with metal layers on top to be constructed from a bare silicon wafer. Figure 1.4 shows a cross- sectional schematic of a single transistor after the second metal layer has been etched. Once all the metal layers are built, the passivation layers are deposited and etched for the bond pads. Finally, the chips are tested and their pass/fail status recorded at electrical sort.

In our scheme, classification of the defects is based on the process step just preceding the inspection step at which the defects were detected. Table 1.1 labels the inspection steps by identifying the process steps immediately preceding them. The defects are named to correspond to the inspection steps at which they were detected. Figure 1.5 shows schematics of the defect mechanisms that can occur during device fabrication. Figure 1.5a illustrates an active bridge and an active bridge. An active bridge is formed when two active regions are connected due to an unintentional break in the field oxide. An active break is formed when the field oxide encroaches upon an active region that should be free of field oxide. As the cross-section shows, these defects appear after island-etch. Figure 1.5b illustrates a poly bridge and a poly break, which manifest after polysilicon etch. A poly bridge refers to the unintentional linking of two polySi layers, and a poly break refers to the unintentional break in a polySi layer. Likewise, Figure 1.5c shows defects occurring during metalization. A metal bridge is the unintentional linking of two metal layers, while a metal break is the unintentional break in a metal layer. A poly- metal short is formed when there is a

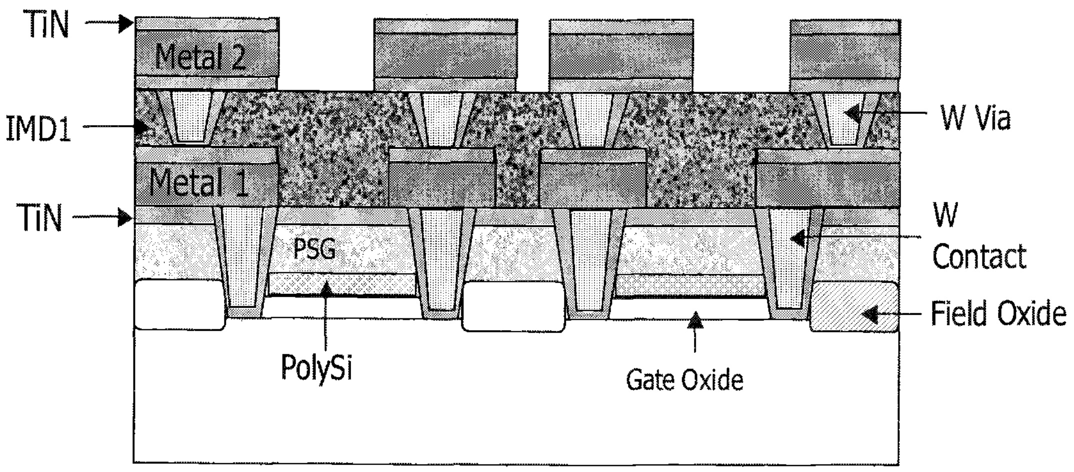
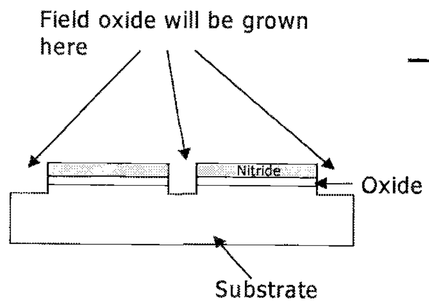
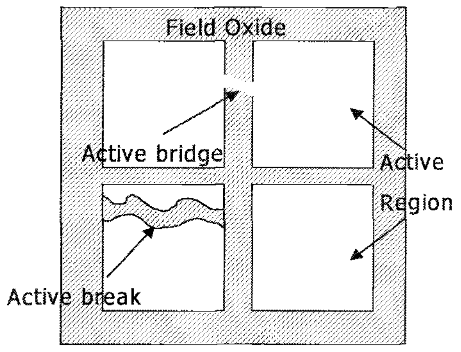


Figure 1.4 Cross-section of a transistor at the end of metal 2 etch

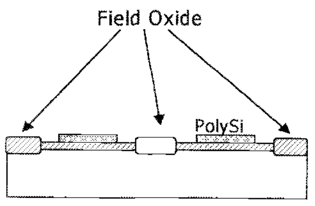


Cross section of device after island etch

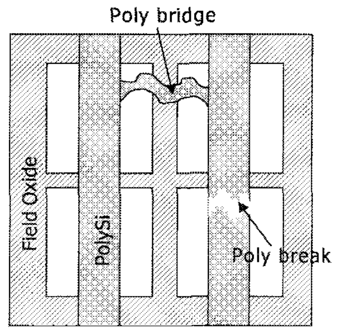


Top view of device after island etch, field oxidation, and nitride and oxide strip

a

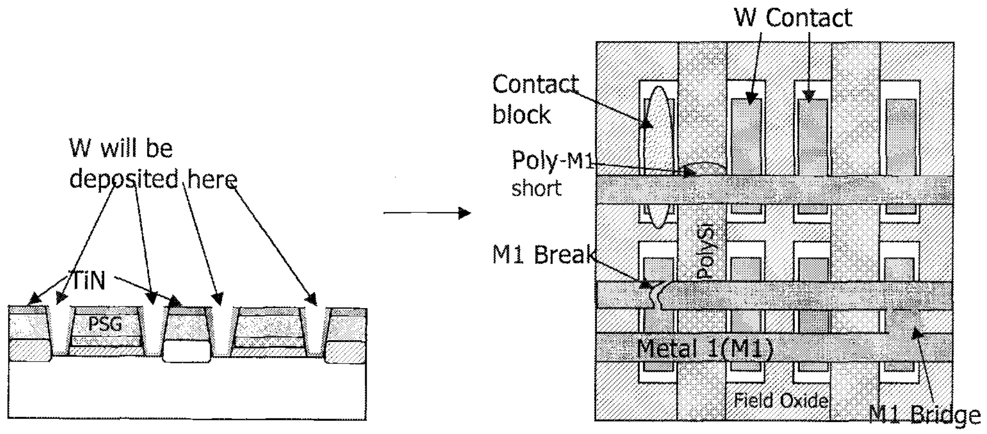


Cross section of device after polysilicon etch



Top view of device after polysilicon etch, spacer deposition, spacer etch, and salicide formation (RTA)

b



Cross section of device after TiN CVD

Top view of device after TiN CVD, W deposition, CMP, M1 deposition, M1 mask, and M1 etch

Figure 1.5 (a-c) (Continued) Schematics of a device at various representative process steps. In each of parts a to c the planar schematic shows the possible defects that can occur between the two process steps shown.

conductive link between the polySi layer and a metal layer. For layers above the metal 1 layer the interconnections between the metals are called via, instead of contacts, and the defect that causes the contact or via to be interrupted is called a contact block, or via-block, respectively.

Table 1.1. Inspection steps examined for defects and the process steps immediately preceding them.

Inspection Step	**Process step immediately preceding it**
ISEF	Island etch: Nitride and Si etch that define the active regions
POLF	Poly etch: PolySi etch
SPCE	Spacer etch
SL2R	Rapid thermal anneal (RTA) that forms salicide layer on PolySi
TN1T	First TiN deposition prior to W deposition and contact formation
M1MD	Metal 1 mask
M1EF	Metal 1 etch
TN2T	Second TiN deposition prior to W deposition and via formation
M2MD	Metal 2 mask
M2EF	Metal 2 etch

1.5 Bootstrapping

The estimation of fault probabilities based on analysis of defect maps produces only point estimates of fault probabilities. A method to come up with range estimates based on this method, i.e., confidence intervals (*CI*), has not yet been reported. Standard formulae exist for estimation of the confidence interval for only a limited number of parameters. For point estimates of *FP* based on measured defect data only, no such analytical procedures exist because the defect map estimated *FP* is complicated in terms of the underlying data structure and point estimation function. Bootstrapping can overcome this difficulty because it can estimate the CI as long as the procedure for obtaining the point estimate from a sample is known.

The basic idea of bootstrapping is very simple. The sample is used as an approximation for the parent population itself. An estimate of the sampling distribution of the estimator of a given parameter can then be achieved by random sampling with replacement from the sample. The resulting bootstrap estimates for the parameter then approximate the actual sampling distribution of the estimator and can be used as a basis to estimate confidence intervals for that parameter. By means of a simple computer algorithm this procedure of re-sampling can be automatic and relatively quick.

The success of bootstrapping as a means of estimating *CI* can be seen by its flourishing application in almost all scientific disciplines. These include the biological sciences [6-8], physical sciences [9], engineering disciplines [10-11], and the social and behavioral sciences [12-13]. This research hopes to add to this history of success by applying the bootstrap to the *CI* estimation of *FP*s of random defects seen in semiconductor processing.

1.6 Outline of the thesis

Chapter 2 describes how fault probabilities were estimated using binomial statistics and kill ratios based on defect data. The resulting fault probabilities are examined and compared with those based on critical area analysis. The estimated random yields (based on Equation (1-1)) are also compared with actual random yields for certain lots from the fab. The optimal method to estimate fault probabilities is determined by the method that gives random yields consistent with the actual random yields. The fault probability for an inspection step is shown to be an average of the different defects seen at the inspection step. Finally, a simulation is done to show the effects of clustering on the estimation of fault probability.

Chapter 3 discusses four methods of confidence interval estimation based on bootstrapping. A simulation based on randomly generated defects, with pre-assigned fault probabilities, distributed on wafers is performed to compare these four methods. The resulting confidence interval estimates are then evaluated based on the proportion of confidence intervals that actually capture the true value of the fault probabilities. Chapter 4 presents the conclusions and recommendations for further study.

2. FAULT PROBABILITY AND KILL RATIO ESTIMATION BASED ON ANALYSIS OF DEFECT DATA

2.1 Introduction

In the literature, the term kill ratio (*KR*) is sometimes used interchangeably with the term fault probability (*FP*). In this study, we distinguish the two. Like *FP*, *KR* can be used to estimate the defect limited yield; therefore, the *KR* as well as *FP* was estimated. The calculations were based on three months of defect data collected on the fabrication line at LSI Logic in Gresham, OR. In this sample set, more than one hundred thousand random defects were detected by optical inspection tools placed after specific process steps on the fabrication line, such as etching or deposition.

In addition to classifying the defects based on the process steps, we also classified the defects by increments of size. The size bins are categorized from *Sz1* to *Sz10*, the bin *Sz1* representing all defect sizes from 0 to less than 1 micron, in diameter, bin *Sz2* from 1 to less than 2 microns, etc, up to bin *Sz10*, representing those defect sizes 9 microns and greater. By defect data we mean the number of dice in a certain category based on criteria such as the type of defect(s) detected on the die and its pass or fail status at probe. For example, T_{GA} is the total number of good dice with defect type A, T_A is the total number of dice with defect type A, T_G is the total number of good dice inspected, and T is the total number of dice inspected for defect type A.

After classifying all the defects based on the above method, the *FP* and *KR* were calculated. Two methods were used to estimate the *FP* from the defect data. One method was based on isolating the dice with only one defect, and counting the total number of dice with a particular defect and the number of failed die for the same particular defect. [14] If N dice with only one defect type, A, are counted, and N_F of them fail, the estimated FP for this defect would be:

$$FP_A = N_F/N \tag{2-1}$$

This method was proposed by Ross, and the estimate based on Equation (2-1) is also referred to as FP_{Ross}. [14]

The second method used to estimate FP from the defect data is based on the kill ratio. A kill ratio can be defined as the ratio of the increased probability that a die will fail due to a particular defect type A present on it, to the probability that the die will not fail if that particular defect A is not present:

$$KR_A = \frac{P\{R/A\} - P\{R/A^c\}}{P\{G/A^c\}} \tag{2-2}$$

[4], where R represents the event that a die is rejected, or fails electrical test, G is the event that the die is good, or passes electrical testing, A is the event that defect type A is present on the die, and A^C is the event that defect type A is not present on the die. If one were also to define the FP_A in the same terms used to define KR_A, we would have:

$$FP_A = P\{R/A_{onlyone}\} \tag{2-3}$$

where $A_{onlyone}$ is the event that a die has only one defect, of type A. In other words, FP_A can be defined as the probability of a die failing when only one defect, of type A, is present. It is straightforward to show that *FP* and *KR* are not, in fact, the same, and estimate different probabilities. For example, if the failure rate is zero when defect type *A* is not present, $P\{R/A^c\}=0$, $P\{G/A^c\}=1$, and Equation (2-2) becomes,

$$KR_A = P\{R/A\} \qquad (2\text{-}4)$$

Comparing Equations (2-3) and (2-4), it is clear that the *KR* will be greater than *FP*, because the probability of failure for a die with at least one defect must be greater than that for a die with just one defect. However, if the defect density of defect type *A* is low, and there is no clustering of defects, most of the die that have any defects on them would have only *one* defect type *A*. In that case, Equation (2-4) would approximate Equation (2-3). Therefore, if the defect density is not too high, the *KR* for a particular defect type may offer a good approximation to the *FP*.

2.2 Estimating Fault Probability based on Kill Ratio

In addition to serving as upper limits to the *FP*, estimating *KR* is of value because it can be used to estimate the defect limited yield, from which the *FP* may be inferred. Based on Equation (2-2), one can show that *KR* of defect type A can be estimated by the following:

$$KR_A = 1 - \frac{\dfrac{T_{GA} - T_G}{T_A - T}}{\dfrac{T_G - T_{GA}}{T - T_A}} \qquad (2\text{-}5)$$

[15].

It can be shown from basic probability theory that the limited yield for defect type A can be computed as follows:

$$LY_A = 1 - P\{A\} \cdot KR_A \tag{2-6}$$

[15]. From this expression, it can be further shown that:

$$LY_A = \frac{T_G(T - T_A)}{T(T_G - T_{GA})} \tag{2-7}$$

[15]. Equations (2-5) to (2-7) are based on the assumption that the defects have the same constant probability of occurring on any dice on the wafer; i.e., the defects have a Poisson distribution. Thus we can equate Equations (1-4) and (2-7) to obtain an expression for FP_A based on the defect data:

$$FP_A = -\frac{\ln\left(\frac{T_G(T - T_A)}{T(T_G - T_{GA})}\right)}{DD_A} \tag{2-8}$$

where DD_A is the number of defects of type A per die.

The negative binomial equation has been shown as representative of the actual distribution of random defects on a wafer in the fab setting because it accounts for defect clustering [5]. The negative binomial equation is given by:

$$p(n) = \frac{\Gamma(\alpha_A + n)}{n!\Gamma(\alpha_A)} \frac{(DD_A / \alpha_A)^n}{(1 + DD_A / \alpha_A)^{n+\alpha_A}} \tag{2-9}$$

where α_A is the cluster factor that determines how clustered the defects of type A are, and *p(n)* is the probability of having *n* defects of type A on a die. Substituting *n*=0 into the negative binomial equation, we have,

$$p_A(0) = \frac{1}{(1 + \overline{DD}_A / \alpha_A)^{\alpha_A}} \tag{2-10}$$

If we know the spatial probability distribution function of defects on a wafer, p_A, we can estimate T_A as,

$$T_A = T(1 - p_A(0)) \tag{2-11}$$

where $p_A(0)$ is the probability a die will not have any defects of type A on it. Substituting Equation (2-10) into Equation (2-11), and rearranging, we have,

$$\frac{1}{(1 + \overline{DD}_A / \alpha_A)^{\alpha_A}} = 1 - T_A / T \tag{2-12}$$

Equation (2-12) is a nonlinear equation for α which can be solved by numerical methods, such as a bisection search or the *Newton –Raphson* method. Thus, we can solve for α once we know *T*, T_A and DD_A. We can calculate the defect limited yield for the defect type A, again with the aid of the negative binomial equation, as

$$LY_A = \frac{1}{(1 + DD_A \cdot FP_A / \alpha_A)^{\alpha}} \tag{2-13}$$

Equation (2-13) does not assume that the defects follow a Poison distribution. Thus, it is not correct, strictly speaking, to equate Equation (2-7), or equivalently Equation (1-4), with Equation (2-13). However, under certain conditions, this equality is a good approximation, even if the distribution is clustered. To illustrate this point, Figure 2.1 compares the behavior of the limited yield under a Poisson distribution, i.e., Equation (1-4), and a distribution that may be clustered, i.e., Equation (2-13), for *FP*=0.02, versus defect density for varying values of α. We see that the agreement between these two equations lessens as the cluster factor decreases, i.e., as the distribution becomes more clustered, or as the *DD* increases. However, at a low *FP*, we see that at a higher *DD* and low cluster factor, the agreement is still quite good. From Figure 2.1 we see that for the *LY* for a clustered distribution with α=0.1 and a *DD* of more than one defect per die, the agreement with the *LY* for a Poisson distribution is better than 99%, when the *FP* is 0.02.

On the other hand, at high *FP*'s, this approximation quickly breaks down as the *DD* is increased. Figure 2.2 shows two sets of *LY*'s, one corresponding to *FP* = 0.02, the same set used in Figure 2.1, and one to *FP* = 0.33. The *LY* curves for *FP*= 0.02 are superimposed on the top curve in this figure, while the *LY*'s set for *FP*= 0.33 are clearly separated, dramatically illustrating the dependence of the approximation of Equations (1-4) and (2-13) on the *FP* value. From the above illustration, we see that at lower values of *FP*, *DD* per defect type below one defect per die, and α above 0.1, conditions typically seen in the fab, the values of the *LY* predicted by Equations (1-4) and (2-13) are comparable. Thus, under the conditions just given, the *FP* predicted by Equation (2-8) should be accurate, even with clustering of defects.

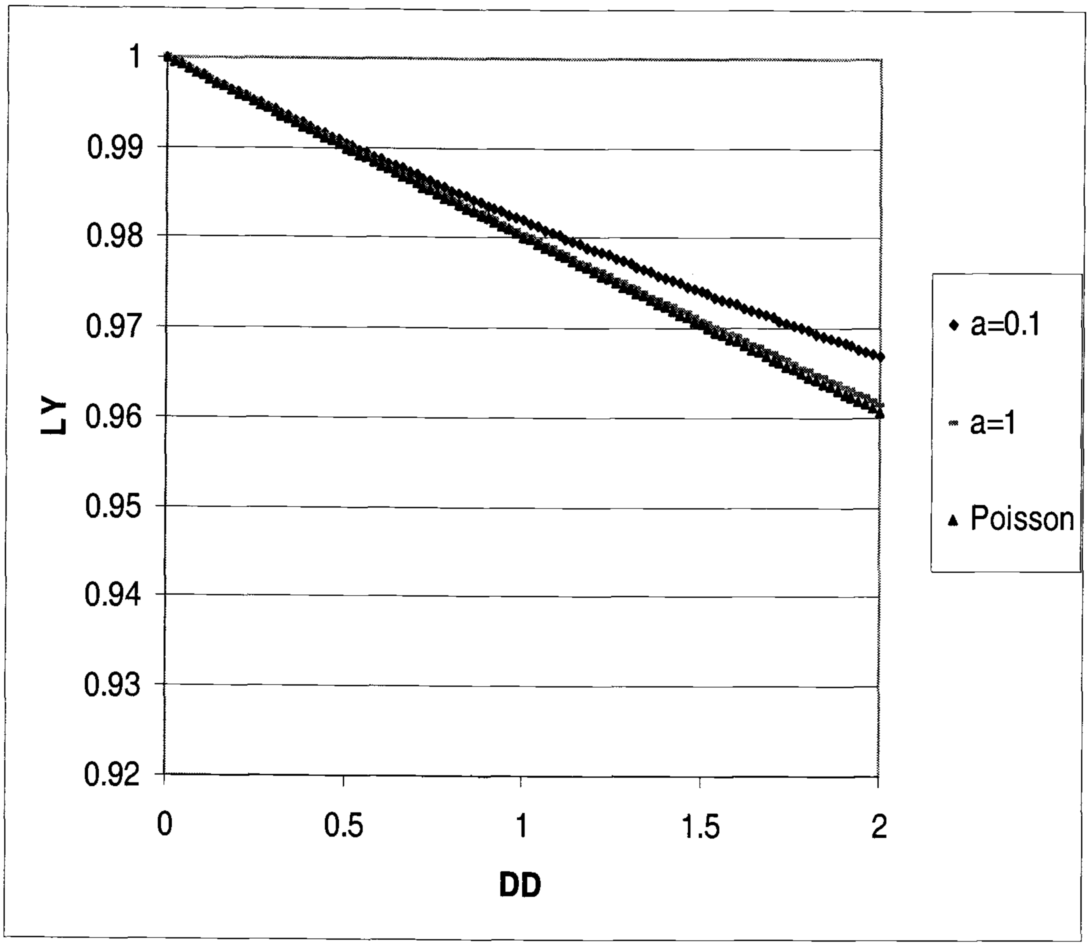


Figure 2.1 Comparison of *LY* versus DD for a Poisson distribution and clustered distributions at two different cluster factors, for FP=0.02

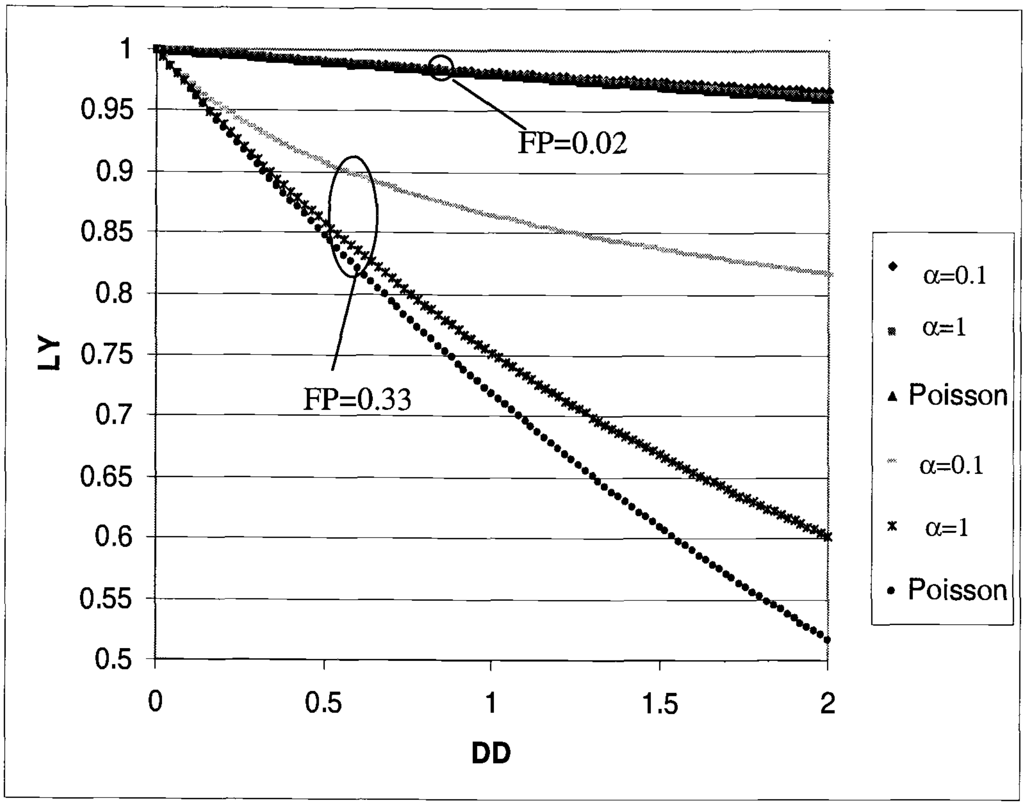


Figure 2.2 Comparison of *LY* versus *DD*, corresponding to *FP*=0.02, and *FP*=0.33. For each *FP* there is a Poisson distribution and two clustered distributions set at different cluster factors. The curves of *LY* at *FP*=0.02 are superimposed on the top line

2.3. Averaging of *FP* for all Defect Sizes

Finally, *FP* values were estimated from computer simulation based on the die layout for each layer, using a method known as critical area analysis (CAA). [16] Basically, in CAA, the computer randomly places defects of a certain type on the die layout; if the defect type were a conductor, for example, the critical area would be the area of the places where this defect type would cause a short. The estimate of the *FP* is then the ratio of the number of defects that landed on a critical area of the layout to the total number of defects generated. To facilitate comparison with the values of *FP* estimated from the defect data, the values of *FP* estimated by CAA simulation had to be averaged over the same size bins, or over all defect sizes. To obtain the average *FP* over a defect size range from the simulation results, we can start with the general expression for the expectation of a function of a random variable:

$$E[g(x)] = \int_{-\infty}^{\infty} g(x) f(x) dx \qquad (2\text{-}14)$$

[17], where *f(x)* is the probability density function of the random variable *x*. If we know *FP* as a function of size, *FP(x)*, where *x* is size, then the average *FP* over all possible defect sizes would be, per Equation (2-14):

$$\overline{FP} = \int_{0}^{\infty} FP(x) h(x) dx \qquad (2\text{-}15)$$

where *h(x)* is the probability density function of the defect sizes, also known simply as the defect size distribution. To determine the average simulated *FP* over a specific defect size range, x_{min} to x_{max}, we use the following expression,

$$\overline{FP}_{x_{\min}\ to\ x_{\max}} = \frac{\int_{x_{\min}}^{x_{\max}} FP(x)h(x)dx}{\int_{x_{\min}}^{x_{\max}} h(x)dx} \qquad (2\text{-}16)$$

The exact functional form of *h(x)* can be determined from defect monitors. However, it has been found that assuming a linear increase in *h(x)* up to a certain size, x_o, and a $1/x^3$ decrease above this size is an adequate approximation for most defect size distributions found in the fab. In most cases, x_o has been found to be much smaller than the minimum dimension of the device [18]. Once x_o is established, *h(x)* is determined by recognizing that the probability density function must satisfy the following relationship:

$$\int_0^\infty h(x)dx = 1 \qquad (2\text{-}17)$$

Assuming that

$$h(x) = ax \qquad \text{for } 0 \le x \le x_o \qquad (2\text{-}18a)$$

and

$$h(x) = b/x^3 \qquad \text{for } x_o \le x < \infty \qquad (2\text{-}18b)$$

[19], we have, by substitution of Equations (2-18) into Equation (2-17),

$$\int_0^{x_0} axdx + \int_{x_0}^{\infty} b/x^3 dx = 1 \qquad (2\text{-}19)$$

Since *h(x)* must be continuous,

$$ax = b / x^3 \qquad \text{at } x = x_o \tag{2-20}$$

Solving Equations (2-19) and (2-20) simultaneously, we have,

$$a = 1 / x_o^{\ 2} \tag{2-21a}$$

and

$$b = x_o^{\ 2} \tag{2-21b}$$

Thus,

$$h(x) = \{ \begin{array}{ll} x / x_0^{\ 2} & for\ 0 \le x \le x_0 \\ x_0^{\ 2} / x^3 & for\ x_0 \le x \le \infty \end{array} \tag{2-22}$$

Figure 2.3 shows an example of *h(x)* where x_o is assumed to be 0.1μm. Once we determine *FP(x)* and *h(x)* we can estimate the average *FP* over all defect sizes for any given defect type by numerically integrating Equation (2-15).

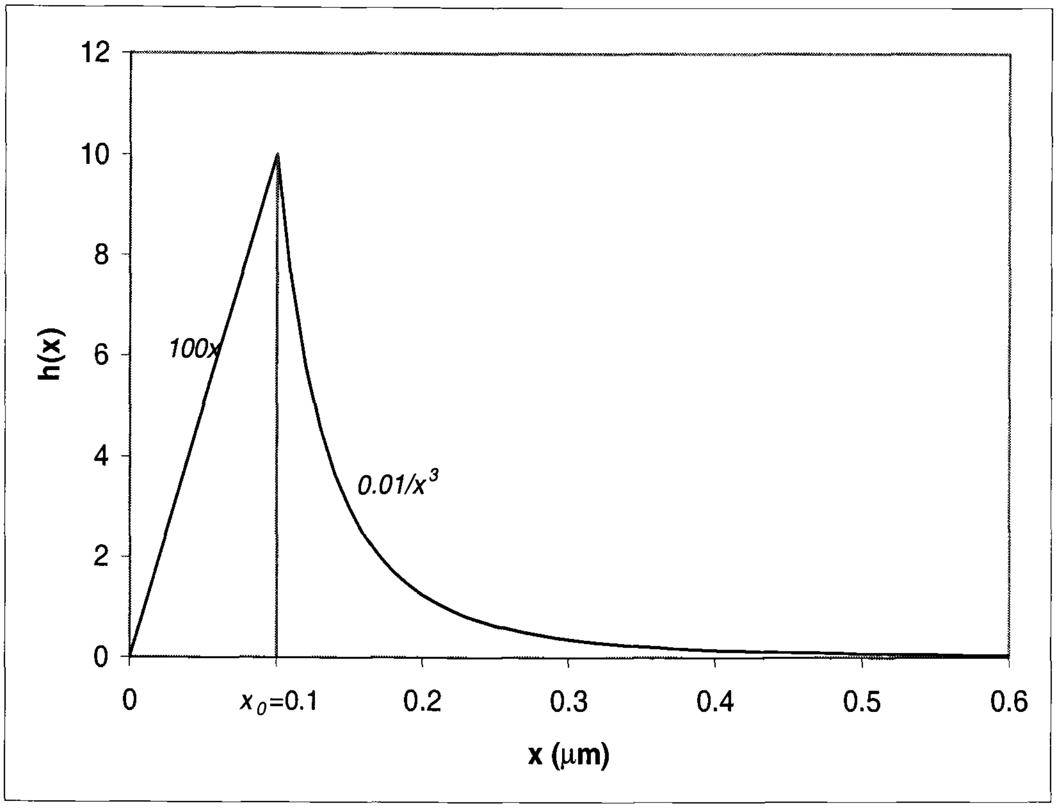


Figure 2.3 Defect Size Distribution, *h(x)*, where x_0=0.1 μm.

2.4 Results and Discussion

2.4.1 Comparison of Estimated Fault Probabilities

When the defects were sorted into the bin sizes, Sz1 to Sz10, greater than 99% of the defects fell into the Sz1or Sz2 bins. Thus the larger size bins would have too much statistical uncertainty associated with the FP estimates for meaningful comparisons. In light of this result, we did not use the size bins in our study. Figure 2.4 shows an example of the fault probability versus defect size curve, *FP(x),* for metal bridges, obtained through critical area analysis. Also shown in Figure 2.4 are the defect size distribution curve, *h(x),* and the *FP (x) * h (x)* curve, the area under which gives us the average fault probability for metal bridges.

Table 2.1 shows the defect density, cluster factor, α, and the counts of T_A, T_{GA}, T and T_G for each of the inspection steps. Table 2.2 shows the estimated FP_{LY} based on Equation (2-8) and the limited yield *LY* estimated by Equation (2-7), using the values shown in Table 2.1, for each of the eight inspection steps. These estimates assume that the inspection tools have no inspection errors that cause defects of type A to go undetected.

In order to compare the *FP* based on defect data (FP_{LY}) with the *FP* based on simulation (FP_{sim}), we must realize that the estimates of FP_{LY} represent an average of the values of the *FP* of different fault mechanisms. The different fault mechanisms by which the defects detected at a particular inspection step may cause a fault are shown

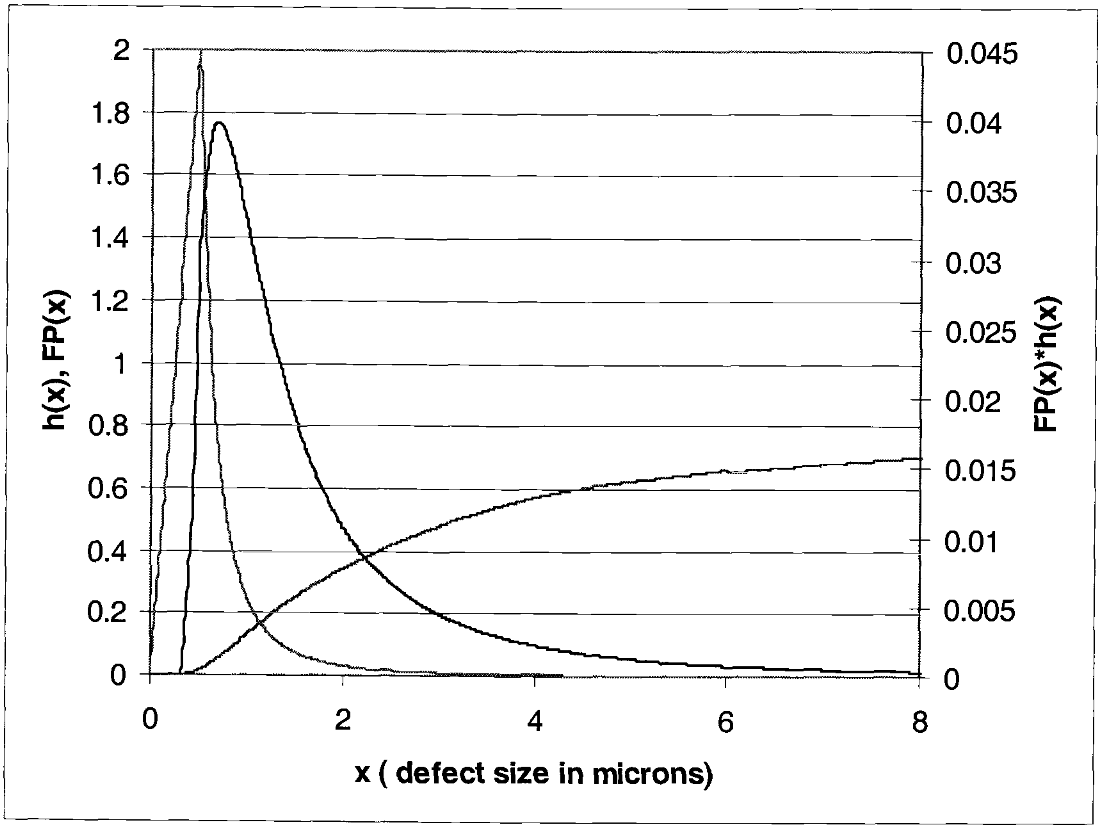


Figure 2.4 An example of *h(x)*, *FP(x)*, and the resulting *FP(x)*h(x)*. *FP(x)* is the fault probability for metal bridges and was obtained from CAA. The average fault probability for all defect sizes is the area under the curve *FP(x)*h(x)*.

Table 2.1 Estimated parameters and the *LY* and FP_{LY} for each of the eight inspection steps

Inspection Step	DD	α	T_A	T_{GA}	T	T_G
ISEF	0.537	0.407	4885	3764	16848	13266
M1EF	0.166	0.266	2206	1686	18252	14420
M2EF	0.169	0.375	2136	1560	16380	12573
M3EF	0.068	0.647	789	601	12636	9676
POLF	0.454	0.298	6653	5398	27612	21840
TN1T	0.071	0.118	1060	700	19656	15781
TN2T	0.091	0.186	835	613	11700	8903
TN3T	0.065	0.310	674	485	11700	8999

Table 2.2 Estimated *LY* and FP_{LY} for each of the eight inspection steps

Inspection Step	LY	FP_{LY}
ISEF	0.9913	0.0162
M1EF	0.9955	0.0270
M2EF	0.9928	0.0428
M3EF	0.9997	0.0052
POLF	1.0000	0.0000
TN1T	0.9900	0.1424
TN2T	0.9973	0.0297
TN3T	0.9961	0.0601

in Table 2.3. Table 2.3 also shows that the values of FP_{sim} for the different fault mechanisms can vary significantly. In general, the *FP* for a particular inspection step *l* can be calculated by the following expression:

$$FP_l = FP_x * t_x + FP_y * t_y + FP_z * t_z + \cdots \qquad (2\text{-}23)$$

where t_x, t_y, t_z are the fraction of defects detected at inspection step *l* that can cause a fault by defect mechanisms *x, y, z*, respectively. This expression will later be verified

Table 2.3 Possible fault mechanisms for the defects detected at various inspection steps compared with defect data estimated *KR* and *FP*

Inspection step	FP_{LY}	Simulated Fault Mechanism/ FP_{sim} at x_0=0.5 μm			
ISEF	0.0164	Active bridge	Active break	PolyM1 short	
		0.0136	0.0092	0.303	
M1EF	0.0272	M1 bridge	M1 break	M1M2 short	M1M2 via block
		0.0551	0.0741	0.416	0.0144
M2EF	0.0432	M2 bridge	M2 break	M2M3 short	M2M3 via block
		0.0613	0.093	0.358	0.0078
M3EF	0.0052	M3 bridge	M3 break	M3M4 short	M3M4 via block
		0.033	0.0626	N/A	0.0002
POLF	0.0000	Poly bridge	Poly break	PolyM1 short	Contact block
		0.0252	0.093	0.303	0.0293
TN1T	0.1486	M1 contact block	M1bridge	M1 break	PolyM1 short
		0.029	0.0613	0.0741	0.303
TN2T	0.0299	M1M2 via block	M2 bridge	M2 break	M1M2 short
		0.0144	0.0613	0.093	0.416
TN3T	0.0605	M2M3 via block	M3 bridge	M3 break	M2M3 short
		0.0078	0.033	0.0626	0.358

by probability arguments. FP_{LY} is assumed to be equivalent to FP_l in Equation (2-23). The fault mechanisms and fractions of each defect type seen at each inspection step must be determined by the use of test structures and failure analysis. [18] With the current classification scheme of the defect data we can only estimate the FP_l.

However, the fault mechanisms can be speculated based upon past experience with similar layouts.

In general, defects may cause a fault at a layer before the inspection step at which it is detected, or cause a fault at a subsequent layer. For example, at ISEF, not only can the active area be affected by extra field oxide -"bridging"- across the active area, but they can also be affected by missing oxide - "active break". In addition, defects detected at ISEF may cause the polySi and metal 1 formed at a subsequent step to be shorted together. Thus for the defects detected at the ISEF inspection step, there may actually be three fault mechanisms at work that can cause a die to fail. By a similar process of looking at the process steps that precede and follow each inspection step, we can deduce the possible fault mechanisms of the defects detected at a particular inspection step.

Once we have determined the process steps at which defects being simulated may occur, the inspection step at which these defects may be detected is established. For example, FP_{LY} for the TN1T inspection step represents the average FP for the defects detected right after the first TiNi deposition. The CAA estimated *FP* named "Contact" is based on the layout showing the location of contacts. TN1T is the inspection step occurring immediately after the contacts are etched and TiN is deposited, and before tungsten is deposited into the contacts, as shown in Figure 1.3. Therefore, we can assume that some of the defects detected at this inspection step are defects that could cause a contact to be blocked, as shown in Figure 1.5c. Similarly, M1 break, M1 bridge, and PolyM1 short are other possible defect mechanisms. Thus

FP_{LY} for the TN1T inspection step should be comparable to the average of the values of FP_{sim} for the contact block, M1 bridge and break, and PolyM1 short.

Using a value of x_0 =0.5 μm in Equations (2-15) and (2-22), the averages of the values of FP_{sim} for each inspection step are comparable to those of FP_{LY}. As can be seen from Equation (2-23), we can adjust the fractions of each defect type shown in Table 2.3 so that FP_{sim}=FP_{LY} at x_o=0.5 μm. For example, using a value of x_0 =0.5 μm, we can arbitrarily adjust the fractions of the FP_{sim} of each defect mechanism detected at TN1T to 25% for M1 contact block, 20% for M1 bridge, 16.4% for M1 break, and 38.6% for polyM1, so that the weighted average of the FP_{sim} of these defect types equals the FP_{LY} value of 0.149. Table 2.3 shows the values of the FP_{sim} at x_0=0.5 μm for each of the possible fault mechanisms of the defects detected at the inspection steps. The estimates of FP_{LY} for each of the inspection steps are also shown.

2.4.2 Comparison of Estimated Fault Probabilities with Yield

To evaluate the various methods of *FP* estimation, we use the estimated values of the *FP* for each inspection step to calculate random yields for wafers which have known yields and defect density for each inspection step. We will assume that FP_{sim} at x_o=0.5 μm is equivalent to FP_{LY}. For FP_{sim} we will use values based on x_o=0.1 μm and x_o=1 μm as well. The estimated yields are calculated using Equation (1-1), where each of the limited yields is calculated using Equations (1-3) and (1-4).

Table 2.4 shows each of the estimated random yields (Y_R) compared with the actual yields for 11 wafers.

Table 2.4 Random Yields, Y_R, calculated based on estimates of *KR*, FP_{Ross}, FP_{LY} and FP_{CAA}

Wafer Number	Actual Random Yield	Y_R estimated from...				
		FP_{Ross} (Eq. 2.1)	FP_{LY} (Eq. 2.8)	FP_{sim} (x_0=0.1 μm)	FP_{sim} (x_0=0.5 μm)	FP_{sim} (x_0=1 μm)
1	0.919	0.840	0.972	0.999	0.972	0.892
2	0.926	0.827	0.961	0.999	0.961	0.89
3	0.942	0.852	0.97	0.999	0.97	0.901
4	0.912	0.811	0.969	0.998	0.969	0.864
5	0.918	0.671	0.94	0.998	0.94	0.835
6	0.948	0.667	0.947	0.998	0.947	0.824
7	0.976	0.740	0.943	0.997	0.943	0.805
8	0.939	0.839	0.968	0.998	0.968	0.882
9	0.916	0.916	0.968	0.999	0.968	0.96
10	0.938	0.854	0.972	0.999	0.972	0.894
11	0.949	0.820	0.99	0.998	0.99	0.877

As we do not know if the eight inspection steps cover all the possible defect limited yields, we can only judge the success of a method of yield estimation by whether it is above or below the actual yield. The results of Table 2.4 show that the predicted yield based on FP_{Ross} significantly under predicts the yield, meaning that *FP* based on Equation (2-1) over-estimates the actual FP per inspection step. FP_{sim} at x_o=1.0 μm is seen to over estimate the true FP as well. Only the *FP* estimated from Equation (2-8), FP_{LY}, and FP_{sim} with x_o less than or equal to 0.5 give yield results that are consistent with the actual yields. For wafers 6 and 7, the random yields predicted by FP_{LY} and FP_{sim} at x_0=0.5 mm are slightly below the actual yields. This indicates that the true value of x_o may be less than 0.5 μm, and FP_{LY} may slightly over estimate the true FP. This value of x_o is greater than the critical dimension of the device, indicating that the assertion that x_o is usually found to be significantly less that the critical dimension may not apply in our case [18].

2.4.3 Sources of Error in FP estimation

The reason that the *FP* based on Equation (2-1) overestimates the true FP is that the defects detected at any of the inspection steps may not represent all the defects actually present on the die. There are two potential reasons that a defect may not be detected on any given wafer: 1) they are covered by a previous deposition (smaller defects go more easily undetected than larger ones); i.e., they occurred at a process step which was never inspected for defects, and 2) they are so similar to the surrounding layout in texture and topography that the inspection tool cannot distinguish them from the background.

It seems the first reason would be most responsible for contributing to the missed defects. This can be understood by realizing that not all the possible process steps at which defects may occur are examined for defects. In other words, there could potentially be more inspection steps that would allow all defects occurring on a wafer to be detected. Thus, although a particular die may show only one defect present on the final defect map, in fact it may contain defects from other steps that were not examined. Table 2.5 shows the possible ways by which defects can go undetected. The column with the title, "Cause of undetected defects" shows the layer under which defects that occur between two inspection steps may be buried and go undetected. For example, defects that occur between the ISEF (island etch) and POLF (poly etch) inspection steps may be covered under the polySi or nitride deposits when inspected at POLF, but defects that occur between the POLF and SPCE inspection steps have no place to be buried when inspected at SPCE.

Table 2.5 Defect Detection and causes of undetected defects

Inspection step	**Cause of undetected defects**
ISEF	Nitride (only defects< 1 μ)
POLF	PolySi (only defects<1 μ)
SPCE	Virtually no place to take cover
SL2R	Virtually no place to take cover
TN1T	PSG, Ti-Nitride (only defects< 1 μ)
M1MD	Metal1, Metal1 PR
M1EF	Virtually no place to take cover
TN2T	IMD1, Ti-Nitride only def < 1u
M2MD	Metal2, Metal2 PR
M2EF	Virtually no place to take cover

We would now like to see how it is possible to over estimate the FP_{LY}. Unlike the assumption behind the Ross method, the derivation of the limited yield based on the *KR* does not assume that there are no other defects present on the die. For example, for the FP_{LY} of TN1T, it does not matter whether there are defects from other process steps that are undetected, as long as those defects are not part of the defects that are classified as being part of TN1T. Defects seen at, or classified as ISEF, will not be seen at TN1T; however, this does not affect the estimation of the FP of the defects seen at TN1T. This is because a basic assumption of the kill ratio is that there *are* other defects types-in our case defects from other inspection steps-present. Thus the FP_{LY} is immune to defects hidden at other inspection steps. However, the FP_{LY} is *not* immune to the effects of defects that are undetected if these defects are the defects whose FP is being estimated. Thus for example, when the FP_{LY} of TN1T is estimated, defects not detected at TN1T that should be detected because they are classified as defects that belong to TN1T will affect the estimation of FP_{LY} of TN1T.

Before we further explore the effects of missing defects on the estimation of FP_{LY}, it would be wise to verify that Equation (2-8), which is only true when the defects are Poisson distributed, can be used to give accurate estimates of FP. Equation (2-8), strictly speaking, should only be used when there are no clustering of defects. To get an estimate of the error we are introducing when using the equation, we can compare the limited yields estimated by Equations (1-4) and (2-13) for the values of *FP*, defect density, and cluster factor we estimated for each inspection step. They should be comparable if clustering is negligible. The results are shown in Table 2.6. All the errors are below 0.01 percent except for TN1T, which has a relatively high estimated FP_{LY} of 0.15, approximately, so that we would expect a less accurate approximation. Even so, the approximation is still less than 0.05 percent off.

Table 2.6 Comparison of limited yields (*LY*) predicted by Equations (1-4) and (2-13) and the percent error of the *LY* for the estimated values of *DD, FP* and α.

Inspection Step	*DD*	FP_{LY}	*LY* (estimated from Eq. 1-4)	*LY* (estimated from Eq. 2-13)	*LY* % Error
ISEF	0.5373	0.0164	0.9912	0.9913	0.0074
MIEF	0.1657	0.0272	0.9955	0.9955	0.0003
M2EF	0.1693	0.0432	0.9927	0.9928	0.0088
M3EF	0.0678	0.0052	0.9996	0.9997	0.0053
POLF	0.4538	0.0000	1.0000	1.0000	0.0000
TN1T	0.0707	0.1486	0.9895	0.9900	0.0456
TN2T	0.0909	0.0299	0.9973	0.9973	0.0014
TN3T	0.0654	0.0605	0.9961	0.9961	0.0049

Thus, we can be sure that any significant estimation error in FP_{LY} will not come from neglecting defect clustering. Besides, we know from the results in Table 2.4 that it is more probable that the FP_{LY} over estimates the true FP. It will later be shown that when clustering is ignored, the FP_{LY} will under estimate the true FP. Most likely, errors come from missing defects that the inspection tools are assumed able to detect, and from misclassifying the dice. Let *a* be the probability of not counting a die as T_A when it does contain defects of type A. This inspection error rate *a* is a measure of the die misclassification rate. An estimate of *a* would be:

$$\hat{a} = 1 - \frac{T_{A_o}}{T_A} \qquad (2\text{-}25)$$

[15], where T_{A_o} is the number of dice with defect type A observed and T_A is the actual number of dice with defect type A. We can define a measure of the rate of missing defects as the capture rate *c*:

$$\hat{c} = \frac{DD_{obs}}{DD_{act}} \tag{2-26}$$

where DD_{obs} is the observed defect density and DD_{act} is the actual defect density. Introducing these two error terms into Equation (2-8), we have

$$FP_A = -\frac{\ln\left(\frac{T_G(T(1-\hat{a}) - T_{Ao})}{T(T_G(1-\hat{a}) - T_{GAo})}\right)}{DD_{Ao} / \hat{c}} \tag{2-27}$$

where T_{Ao} and T_{GAo} are the observed counts of T_A and T_{GA}.

Using Equation (2-27), with $\hat{a}$ =0.05, and $\hat{c}$ =0.9, so that the rate of correctly counting a die as T_A is 1-0.05=0.95, and the rate of correctly identifying a defect is 0.9- in effect allowing the probability of misclassifying a die to be lower than missing a defect- we can see how FP_{LY} might over-estimate the true FP. If *1-a* is less than *c*, then the FP_{LY} would under estimate the true FP. Table 2.7 shows the results of using these values for the inspection errors in Equation (2-27) compared to that assuming no inspection errors.

The results from Table 2.7 show that if we assume no errors in our inspection, we could possibly over estimate the FP, as long as 1-*a* is greater than *c*. A greater probability of missing a defect than misclassifying a die is more likely to occur with clustering. With clustering, there are some dice with many defects on them, and missing a few of these defects will not affect the classification of those dice as T_A, as much as it will lower the defect density estimate. Since clustering is significant at each of the inspection steps, it seems probable that *1-a* is greater than *c*.

Table 2.7 Comparison of FPLY estimates based on assumption of no inspection error with those based on inspection errors

Inspection Step	FP without error	FP with error
ISEF	0.0162	0.0157
M1EF	0.0270	0.0258
M2EF	0.0428	0.0409
M3EF	0.0052	0.0049
POLF	0.0000	0.0000
TN1T	0.1424	0.1352
TN2T	0.0297	0.0283
TN3T	0.0601	0.0571

In summary, we see that FP_{Ross} over estimates the fault probability because of undetected defects. FP_{LY} slightly overestimates the fault probability if we do not account for missing defects, but may be corrected by incorporating the inspection error rates in its estimation. FP_{sim} with x_o less than 0.5 um gives us estimates of FP that are consistent with yield as well.

2.4.4 The Components of the *FP* Estimated for an Inspection Step

We now verify that the FP_{LY} estimated by Equation (2-8) is equivalent to the FP_l defined by Equation (2-23). We can show that *FP* must be defined by Equation

(2-23) in order for the limited yields based on the *KR* , Equation (2-6) and the limited yield based on based on the Poisson equation, Equation (1-4), to agree; i.e., for Equation (2-8) to be valid. We begin by deriving an expression for KR_A in terms of the density probability distribution and component *FP* of each of the defects seen at inspection step A. To estimate *KR* for inspection step A, we need all the estimates for the parameters on the RHS of Equation (2-5). T_A can be estimated once we know the spatial probability distribution function of defects seen at inspection step A, p_A. Since T_A is the number of dice with the number of defects greater than or equal to 1 detected at inspection step A, we can use Equation (2-11) to estimate T_A,

$$T_A = T(1 - p_A(0)) \qquad (2\text{-}28)$$

To estimate T_{GA}, we use the formula for conditional probabilities [17]:

$$T_{GA} = T * P(GA) = T * P(A) * P(G/A) = T * (1 - p_A(0)) * [\frac{p_A(1)}{1 - p_A(0)} P(G/A_1) + \frac{p_A(2)}{1 - p_A(0)} P(G/A_2) + \frac{p_A(3)}{1 - p_A(0)} P(G/A_3) + \cdots] \qquad (2\text{-}29)$$

where $P(G/A_1)$, $P(G/A_2)$, $P(G/A_3)$, ..., are the conditional probabilities of a die not failing given it has exactly one defect found at inspection step A, exactly two defects found at inspection step A, exactly three defects found at inspection step A, ..., respectively. These probabilities are weighted by the probability that a die will have a certain number *n* of defects occurring on it, $p_A(n)$; i.e., the probability that exactly *n* number of defects will occur on the die. For $P(G/A_1)$, we have,

$$P(G/A_1) = [(1-\overline{FP}_x)t_x + (1-\overline{FP}_y)t_y + (1-\overline{FP}_z)t_z + ...]\cdot(LY)_B(LY)_C \cdots Y_S \tag{2-30}$$

where t_x, t_y, t_z,... are the fraction of defects with fault mechanism x, y, z...(or defect types x, y, z,...) found at inspection step A. LY_B, LY_C ...are the defect limited yields for all the other inspection steps B, C,.... , which are independent of the yield at other inspection steps, and Y_S is the systematic yield. Defining $\overline{FP}_A$ as,

$$\overline{FP}_A = \overline{FP}_x \cdot t_x + \overline{FP}_y \cdot t_y + \overline{FP}_z \cdot t_z + ... \tag{2-31}$$

where $t_x + t_y + t_z + ...=1$, Equation (2-30) becomes,

$$P(G/A_1) = [1-(\overline{FP}_x \cdot t_x + \overline{FP}_y \cdot t_y + \overline{FP}_z \cdot t_z)](LY)_B(LY)_C \cdots Y_S = (1-\overline{FP}_A)(LY)_B(LY)_C \cdots Y_S \tag{2-32}$$

We can show by a similar process that,

$$P(G/A_2) = (1-\overline{FP}_A)^2 LY_B LY_C \cdots Y_S \tag{2-33}$$

and

$$P(G/A_3) = (1-\overline{FP}_A)^3 LY_B LY_C \cdots Y_S \tag{2-34}$$

etc. Substituting Equations (2-32) to (2-34) back into Equation (2-29), we obtain,

$$T_{GA} = T(Y_B Y_C \cdots Y_s)[p_A(1)(1-\overline{FP_A}) + p_A(2)(1-\overline{FP_A})^2 + p_A(3)(1-\overline{FP_A})^3 + \cdots] \tag{2-35}$$

T_G is simply the number of die that are yielding,

$$T_G = T(LY)_A (LY)_B (LY)_C \cdots Y_S \qquad (2\text{-}36)$$

Substituting Equations (2-28), (2-35) and (2-36) into the expression for the *KR*, Equation (2-5), and simplifying, we have,

$$KR_A = 1 - \frac{[p_A(1)(1-\overline{FP}_A) + p_A(2)(1-\overline{FP}_A)^2 + p_A(3)(1-\overline{FP}_A)^3 + \ldots]}{(1-p_A(0))} \cdot \frac{p_A(0)}{Y_A - [p_A(1)(1-\overline{FP}_A) + p_A(2)(1-\overline{FP}_A)^2 + p_A(3)(1-\overline{FP}_A)^3 + \ldots]} \qquad (2\text{-}37)$$

Using the spatial probability distribution function, p_A, as given by Equation (2-9), and estimating LY_A by Equation (2-13), we can solve Equation (2-37) for $\overline{FP}_A$. The value of $\overline{FP}_A$ for each inspection step was calculated according to Equation (2-37). These values were equivalent to those calculated by Equation (2-8). Thus we have shown that the FP_{LY} for inspection step A estimated based on Equation (2-8) is consistent with the average *FP* for inspection step A as defined by Equation (2-31). This definition of $\overline{FP}_A$ is the only definition that would be appropriate in the yield equation given by Equation (2-13) because the term $DD_A \cdot FP_A$ in Equation (2-13) must refer to the average number of faults per die, λ, and only if FP_A is defined as in Equation (2-31) can this term be equivalent to λ. [18]

From Equation (2-37), we can also see that only when the defect density is extremely low are $\overline{FP}_A$ and KR_A the same. That is, as the defect density approaches zero,

$$p_A(0) \to 1 \qquad (2\text{-}38)$$

while $p_A(1)$, $p_A(2)$, ... approach zero, and Y_A approaches 1, so that Equation (2-37) becomes

$$\begin{aligned} KR_A &= 1 - \frac{p_A(1)(1-\overline{FP}_A)}{1-p_A(0)} \cdot \frac{p_A(0)}{1-(p_A(1))(1-\overline{FP}_A)} = \\ &1 - \frac{p_A(1)(1-\overline{FP}_A)}{p_A(1)} \cdot \frac{1}{1} = 1-(1-\overline{FP}_A) = \overline{FP}_A \end{aligned} \qquad (2\text{-}39)$$

Here we also use the approximation that $p_A(0) + p_A(1) = 1$, so that $1 - p_A(0) = p_A(1)$. The approach of $p_A(0)$ to 1, and hence the approach of KR_A to $\overline{FP}_A$, is faster for spatial distributions that follow a more random pattern; i.e., as α in Equation (2-37), contained in the expressions for LY_A and p_A, becomes larger. Figure 2.5 plots KR_A for different values of α using Equation (2-37) and compares them to the given $\overline{FP}_A$, the bottom most horizontal line on the plot. As α gets larger, and the *DD* becomes lower, the curves of KR_A approach $\overline{FP}_A$.

2.4.5 Estimation of *FP* when the Defects are Clustered

Up to this point, we have assumed in our estimation of FP_{LY} that the probability that a defect will occur on a die is the same for all die. However, we see from the estimated values of α shown in Table 2.4 that the defects do not completely follow this type of distribution. Nevertheless, we showed that because the defect density and values of *FP* were low, the limited yields based on the assumption of no

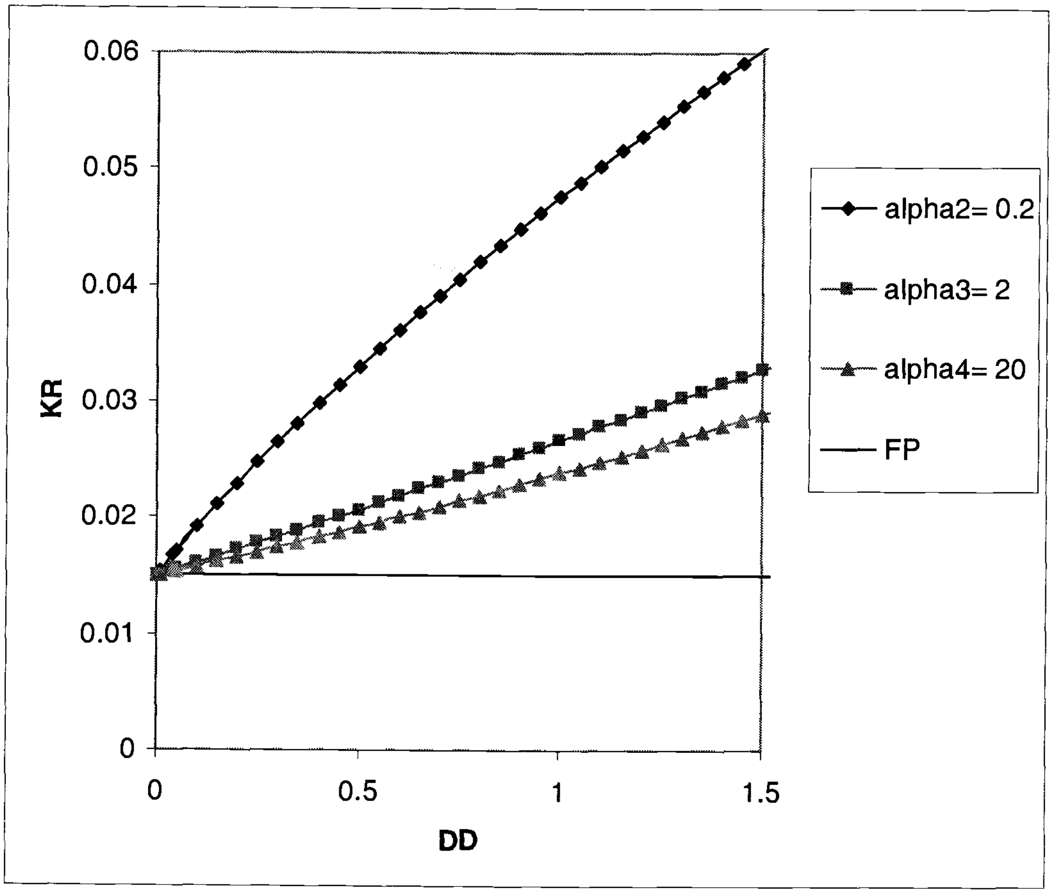


Figure 2.5 *KR* versus defect density at various cluster factors compared to *FP*

clustering, Equation (1-4), and based on clustering, Equation (2-13), are approximately the same, and the *FP* estimated from Equation (2-8) is thus approximately accurate.

If we need to be more accurate in our estimation of *FP*, or we have a situation in which the defect density or *FP* is high, there is another way to estimate *FP* based on defect data that is accurate for any defect distribution. This method is based on the following result derived from basic probability:

$$E(T_{GA}) = \frac{T_{G0}}{T_0} \sum_{i=1}^{T_A} \left(\prod_j (1 - FP_j)^{N_{ij}} \right) \qquad (2\text{-}40)$$

where T_{G0} is the number of die with zero visible defects that are good, T_0 is the number of die with zero visible defects, $E(T_{GA})$ is the expected value of T_{GA}, FP_j is the fault probability of defect type *j*, and N_{ij} is the number of type *j* defects in die *i*. [20] For each defect type *A*, then, in addition to determining T_{GA} and T_A, which is used to estimate $E(T_{GA})$, we must count the number of each defect type in each die with at least one defect type A , up to T_A dice. If we have *n* defect types that we have identified with our *n* inspection steps, then we will have *n* equations with *n* unknowns, where each equation is based on Equation (2-40).

Table 2.8 shows the results of estimating *FP* based on Equation (2-40) using the same defect data previously used, along with the values of α, the defect density, and the values of *FP* based on Equation (2-8) that we calculated previously. The resulting set of eight nonlinear equations was solved using a modified *Newton-Raphson* iteration method, where the initial values used were those calculated based on Equation (2-8). As can be seen from Table 2.8, the estimates of *FP* based on Equations (2-40) and (2-8) are practically the same, confirming our previous conclusion that clustering can be neglected when the defect density and *FP* are low.

Table 2.8 Comparison of the estimates of *FP* based on the assumption of no defect clustering (Eq. (2-8)) and based on no assumption regarding defect clustering (Eq. (2-40))

Inspection Step	DD (defects/ die)	α	FP_{LY}	FP_{NC}
ISEF	0.537	0.406	0.016	0.017
M1EF	0.165	0.266	0.027	0.029
M2EF	0.169	0.375	0.043	0.046
M3EF	0.067	0.642	0.005	0.005
POLF	0.453	0.297	0.000	0.000
TN1T	0.070	0.113	0.149	0.151
TN2T	0.090	0.181	0.030	0.031
TN3T	0.065	0.313	0.060	0.058

A computer simulation was developed in order to show the effect of ignoring clustering when the *FP* or *DD* is high. In the simulation, we can input the values for *DD*, α, and the true value of *FP* for up to 10 inspection steps. The estimates given in the simulation are based on Equations (2-8) and (2-40). Figure 2.6 shows the average of the two different estimates of *FP*, FP_{LY} and FP_{NC} (the FP based on Equation (2-40)) for a particular defect type at different defect densities, whose *FP* is set at 0.4 and α set at 0.1. Each point represents the average of 100 estimates. We see that as the defect density increases, the *FP* estimated by Equation (2-8) increasingly under

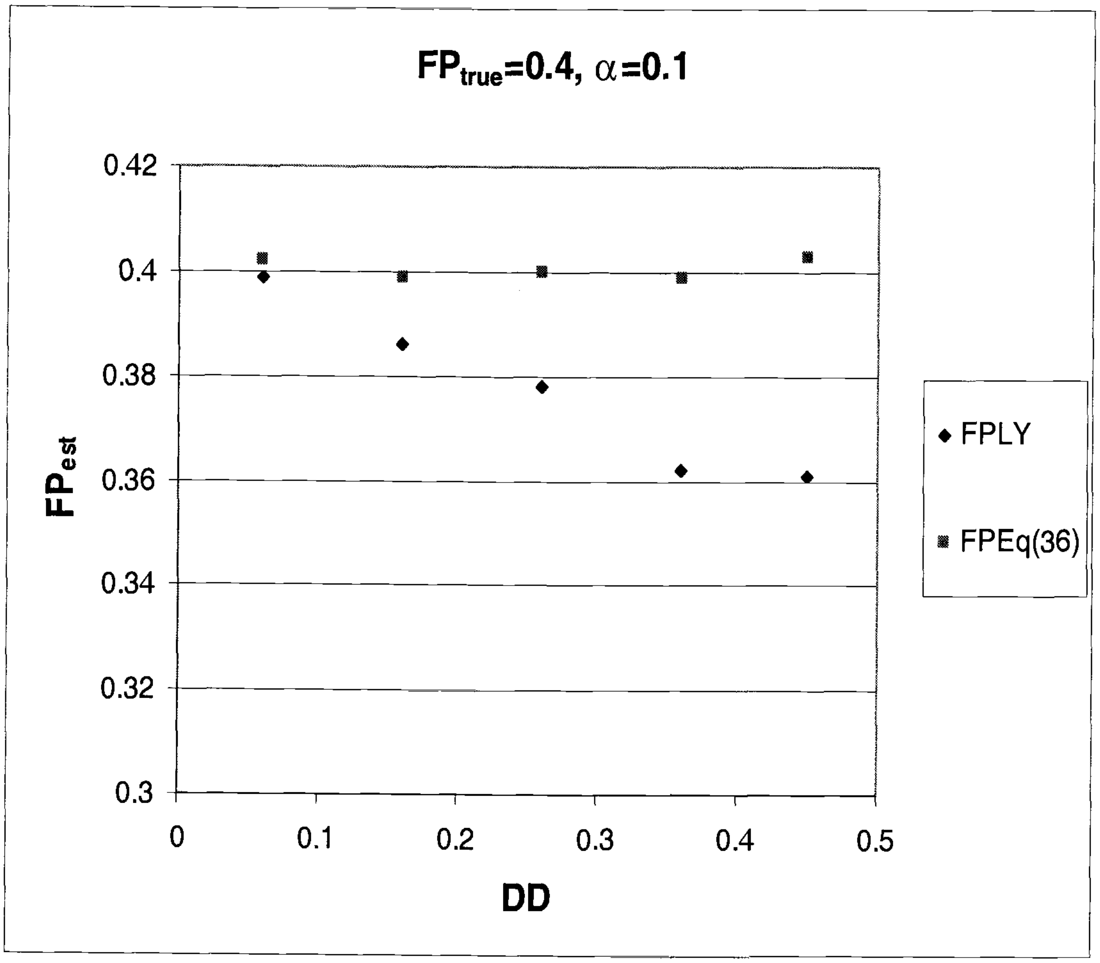


Figure 2.6 Average FP estimates based on assumption of no clustering, FP_{LY}, and based on no assumption regarding clustering, FP_{CI}, versus defect density.

estimates the true *FP*, while the *FP* estimated by Equation (2-40) remains approximately 0.4.

The simulation program can be used to assess the adequacy of using Equation (2-8) to estimate *FP*. Once we have estimated the *DD*, α, and the *FP* (estimated by Equation (2-8)) from the defect data, we can input these values into the simulation program. If the *simulation*-estimated *FP* based Equation (2-40) is close to the value of *FP* that was inputted, then Equation (2-8) should be adequate for estimating *FP* from the defect data. If not, we should use Equation (2-40). Estimating *FP* using Equation (2-8) is still useful in this case, since it can be used as an initial value to solve the nonlinear equation of Equation (2-40).

2.5 Conclusion

The most reliable means of estimating the fault probability for an inspection step are by CAA simulation of probable defect mechanisms on the layout for the layer closest to inspection step A, and by the use of the fault probability estimate based on equating the defect limited yield equation given by the Poisson equation with that given by the kill ratio. FP_{LY} can be a more accurate estimate of FP if we incorporate inspection error rates into its estimation. A defect distribution parameter of x_o=0.5 μm or slightly less causes the two methods to give approximately equal yields. This value of x_o is greater than the critical dimension of the device. Estimates using FP_{Ross} uniformly over estimate the fault probability due to the presence of undetected defects.

It was shown that the estimated FP_{LY} for inspection step A is equivalent to the weighted average of the various defect mechanisms detected at inspection step A. Our analysis also shows that KR_A approximates the FP of inspection step A only under the conditions of low defect density and low clustering (high cluster factor α). It also shows KR provides an estimate of an upper limit for FP and is an integral part of the derivation of the expression for FP_{LY}. Finally, a simulation program was developed for testing whether defect clustering can be ignored when the cluster factor α is low and the FP and defect density are high.

3. CONFIDENCE INTERVAL ESTIMATION BASED ON BOOTSTRAPPING FOR THE FAULT PROBABILITIES OF RANDOM DEFECTS SEEN IN INTEGRATED CIRCUIT PROCESSING

3.1. Introduction

3.1.1 Methods of Confidence Interval Estimation

To construct a confidence interval for an estimate, we must have some knowledge of the sampling distribution of the estimator. In bootstrapping, knowledge of this distribution comes from the bootstrap sampling distribution [21]. Referring to Figure 3.1, which is a general schematic of a sampling distribution, the distances a and b are approximated by the corresponding distances in the bootstrap distribution. If we knew the actual sampling distribution, a central $(1\text{-}2\alpha)\%$ CI could be estimated by the following estimates as the upper confidence limit (UCL), and lower confidence limit (LCL):

$$UCL = t + a \tag{3-1}$$

and

$$LCL = t - b \tag{3-2}$$

where t is the sample estimate [21].

Four methods of CI estimation by bootstrapping will be examined in this chapter. These methods are: the standard method, the first percentile method, the second percentile method, and the BCA method. The fundamental assumption used

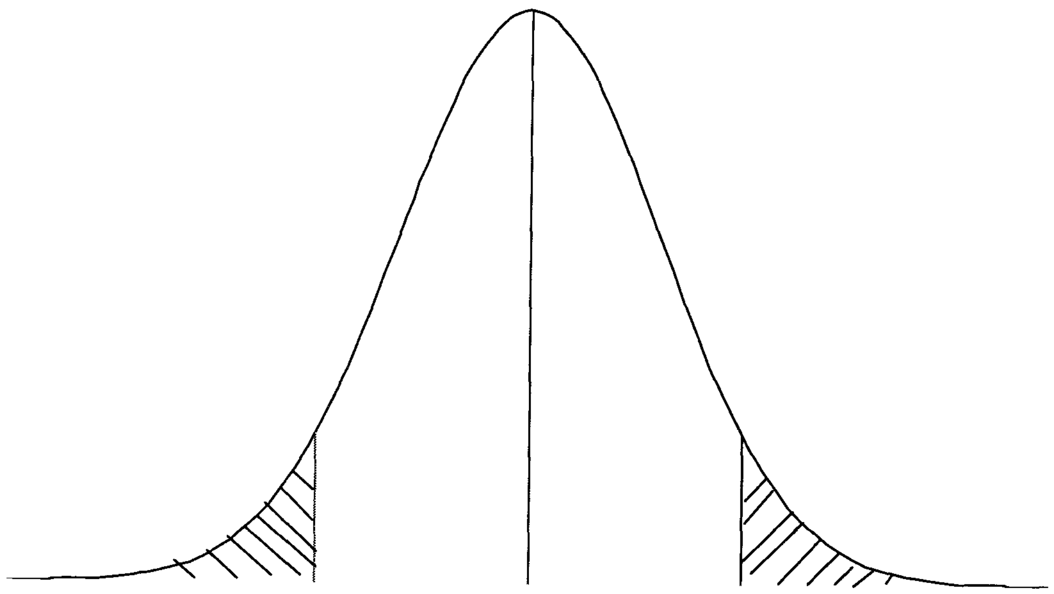


Figure 3.1 A general representation of a sampling distribution.

by the standard, first percentile, and second percentile methods, referred to hereafter as the basic pivotal methods, is that the variance of the sampling distribution of the estimator is independent of the value of the parameter θ being estimated. The basic pivotal methods rely on the assumption that the variance, or equivalently, $\hat{\theta}_\alpha - \theta$, can be approximated by the corresponding value in the bootstrap distribution. This statement is equivalent to assuming that $\hat{\theta}_\alpha - \theta$ is an approximately constant quantity no matter what the value of θ is. A measure such as this is called a pivotal quantity, and results from the variance of the sampling distribution of the estimator being approximately constant regardless of the value of the parameter being estimated.

The simplest method of estimating a confidence interval is variously called the standard method or the normal approximation. This method assumes that the estimator, $\hat{\theta}$, follows an approximately normal distribution with mean equal to the true value of the parameter, θ, plus the bias of the estimate, *B,* defined as $E(\hat{\theta})-\theta$. The distances *a* and *b* in Figure 3.1 are thus estimated by $z_{1-\alpha} V_R^{1/2}$, where $z_{1-\alpha}$ is the standard normal variable having an area of $1-\alpha$ to the left, and V_R is the sample variance of the resulting estimates from bootstrapping, t_r^*, $r=1,2,\ldots R$, where R is the total number of bootstrap samples:

$$V_R = \frac{1}{R-1}\sum_{r=1}^{R}(t_r^* - \overline{t^*}_R)^2 \qquad (3\text{-}3)$$

The bias is estimated by using the sample estimate, *t*, as the population parameter, and the average of all the bootstrap estimates,

$$\overline{t^*_R} = \frac{1}{R}\sum_{r=1}^{R} t^*_r \qquad (3\text{-}4),$$

as the estimate of the expectation of the estimator. The bootstrap estimate of bias, denoted as B_R, is then given by:

$$B_R = \overline{t^*_R} - t \qquad (3\text{-}5)$$

The second percentile method uses $b \approx t^*_{1-\alpha} - t$ and $a \approx t - t^*_{\alpha}$. In the case of the first percentile method, the distance *b* is estimated by $t - t^*_{\alpha}$ while *a* is estimated by $t^*_{1-\alpha} - t$. Therefore, the first percentile method also relies on the assumption that $\hat{\theta}_{\alpha} - \theta$ is an approximately pivotal quantity, but the estimates for the pivotal quantities are swapped with those of the second percentile method. Thus, we see that for the basic pivotal methods to work, the estimated distances *a* and *b* from bootstrapping must approximate that of the actual sampling distribution, for all values of $\hat{\theta}$. The estimates of the distances *a* and *b* for these three methods are summarized in Table 3.1. The upper and lower bounds of these methods can then be estimated by using Equations (3-1) and (3-2).

Table 3.1 Estimates of the distances *a* and *b* in the sampling distribution for various CI estimation methods

Method	***a***	***b***
Normal	$V_R^{1/2}\, z_{1-a}\text{-}B_R$	$V_R^{1/2}\, z_{1-a}+B_R$
First	$t^*_{1-\alpha}-t$	$t-t^*_{\alpha}$
Second	$t-t^*_{\alpha}$	$t^*_{1-\alpha}-t$

The BCA method is based on the assumption that a monotonic transformation m exists such that $\hat{\phi}=m(\hat{\theta})$ follows a normal distribution. The mean and variance of this normal distribution, however, incorporate two additional parameters, the acceleration and bias constants, a_c and β_0, respectively:

$$\hat{\phi} \sim nl(\phi-\beta_0\sigma_\phi,\sigma_\phi{}^2) \qquad \text{(3-6a)},$$

where $\phi=m(\theta)$, *nl( )* represents the normal distribution, and the standard deviation is given by

$$\sigma_\phi = 1+a_c\phi \qquad \text{(3-6b)}$$

[22].

The parameter a_c improves the approximation of the first percentile method by accounting for distributions where the variance might change with the population parameter being estimated, on the normalized scale. From Equation (3-6b) we see can see that $a_c = \frac{d\sigma_\phi}{d\phi}$. Thus a_c is a measure of the rate of change of the standard deviation of the transformed estimator $\hat{\phi}$ with the transformed parameter ϕ.

As seen from Equation (3-6a), the parameter β_0 is a measure of the bias of the normalized estimator. If the untransformed estimator is biased, as estimated by the bootstrap estimate of bias, Equation (3-5), the bias constant will also reflect this bias, but on the normalized scale. Even if the untransformed estimator is unbiased, however, there may be a bias once the estimator is normalized. This situation arises when the untransformed estimator has a skewed distribution. In this case, the mean and median of the untransformed distribution differ, that is there is a median bias, and the normally transformed skewed distribution will have a bias relative to ϕ. Thus the parameter β_0 allows for the normal transformations of skewed distributions, in addition to accounting for biases in the estimator.

From relationship (3-6a) it can be shown that a *1-2*αconfidence interval for θ based on the BCA method is:

$$t^*_{\alpha_1} < \theta < t^*_{\alpha_2}, \qquad (3\text{-}7)$$

where α_1 and α_2 are given by the following probabilities

$$\alpha_1 = \Pr(\, z < \hat{\beta}_0 + \frac{\hat{\beta}_0 + z_\alpha}{1 - \hat{a}_c(\hat{\beta}_0 + z_\alpha)} \,) \qquad (3\text{-}8a)$$

and

$$\alpha_2 = \Pr(\, z < \hat{\beta}_0 + \frac{\hat{\beta}_0 + z_{1-\alpha}}{1 - \hat{a}_c(\hat{\beta}_0 + z_{1-\alpha})} \,) \qquad (3\text{-}8b)$$

$\hat{\beta}_0$ and $\hat{a}_c$ are the bootstrap estimates of the bias and acceleration constants. The bias-correction constant $\hat{\beta}_0$ can be estimated by the standard normal variable whose area to the left is equal to the proportion of bootstrap estimates, $\hat{\theta}^*(b)$, that are below the sample estimate, $\hat{\theta}$. It can be computed by,

$$\hat{\beta}_0 = \Phi^{-1}\left(\frac{\#\{t^*(b) < t\}}{B}\right) \tag{3-9}$$

where B is the total number of bootstrap estimates, and $\#\{\hat{\theta}^*(b) < \hat{\theta}\}$ is the number of bootstrap estimates below the sample estimate. The acceleration constant $\hat{a}_c$ can be estimated by,

$$\hat{a}_c = \frac{\sum_{i=1}^{n}\left(\overline{t_{(\cdot)}} - t_{(i)}\right)^3}{6\left\{\sum_{i=1}^{n}\left(\overline{t_{(\cdot)}} - t_{(i)}\right)^2\right\}^{2/3}} \tag{3-10}$$

Here, $t_{(i)}$ is what is called a *jackknife* value of the statistic $\hat{\theta} = s(\mathbf{x})$ [22]. It is computed as $t_{(i)} = s(\mathbf{x}_{(i)})$, where $\mathbf{x}_{(i)}$ is the original sample with the ith data point removed. $\overline{t_{(\cdot)}}$ is the average of all the jackknife estimates from the sample $\mathbf{x}$: $\overline{t_{(\cdot)}} = \sum_{i=1}^{n} t_{(i)} / n$. Like the basic pivotal methods, the BCA percentiles are also percentiles of the bootstrap distribution based on a sample with t as its estimate. If β_0 and a_c are equal to zero, for example, then relationship (3-7) is equivalent to the confidence limits based on the first percentile method.

3.1.2 Bootstrap Simulation

Since the KR method is more economical than the CAA method, it is desirable for FP estimation. However, the KR method relies on a sample of limited size. Thus this chapter explores the uncertainty associated with this FP estimate through Monte Carlo simulation. In the simulation, the values of the FP estimates from the previous chapter will be used, as they reflect realistic estimates from high volume manufacturing data.

In the simulation, we use three defect types, A, B and C, with assigned fault probabilities FP_A, FP_B, and FP_C, respectively, for each run. Thus the defect types are classified by the value of the FP assigned to them. These defects are distributed on wafers of 100 dice each, arranged in a 10-row by 10-column configuration. The upper and lower number of defects of each type that can occur on a wafer range from 30 to 40 defects per wafer. There is no clustering of defects, i.e., they can occur with equal probability anywhere on the wafer. A typical defect distribution output from the simulation is shown in Figure 3.2.

The re-sampling procedure is as follows: Each original sample contains 20 wafers, with defects and faults distributed on them according to values of FP, and density. Each die is then assigned a bin number, or pass/ fail status, depending on whether a fault lies in it. The bootstrap sample is created by randomly picking dice from this original sample, with replacement, until 20 x 100 dice are picked. These 2,000 dice, the same number of dice in the original sample, constitute a bootstrap sample. From this bootstrap sample the fault probability of any defect type *i*, FP_i, is

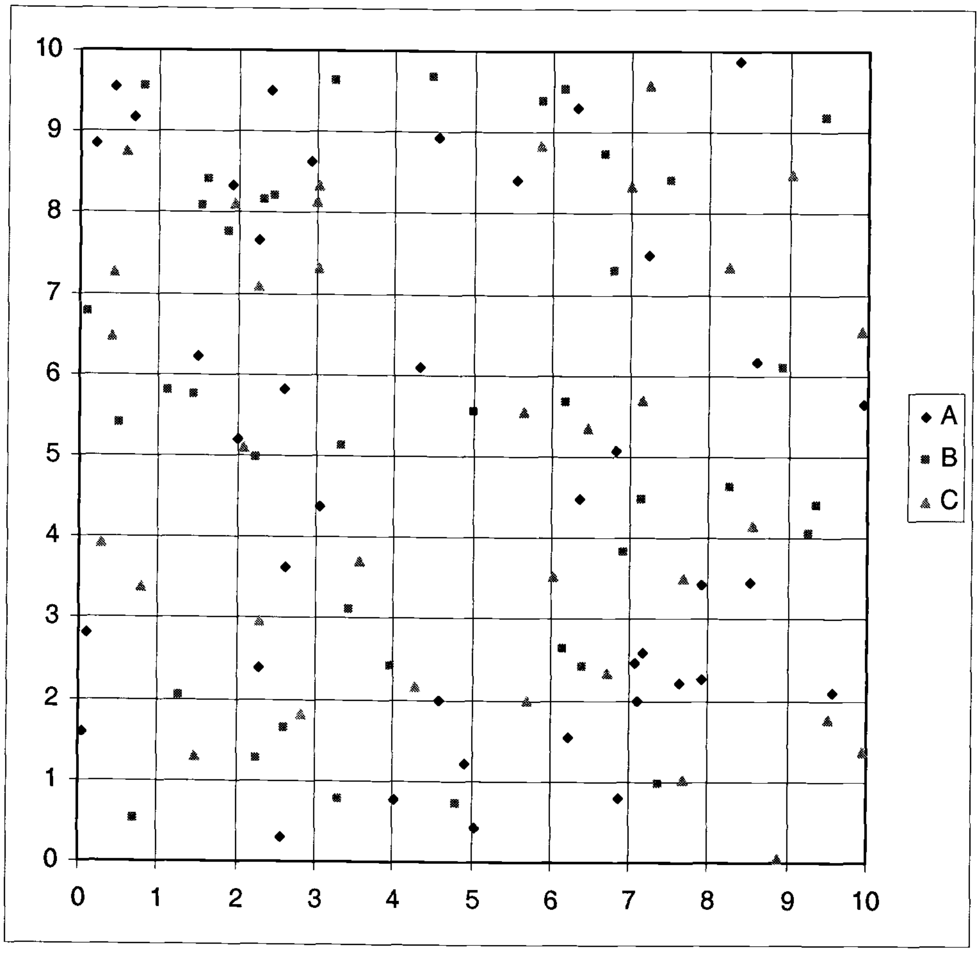


Figure 3.2 Typical random distribution obtained of the three defect types used in the simulation on a 10x10 wafer

estimated the same way as it was in the original sample, i.e., by equating the limited yield for defect type *i*, as estimated by the Poisson yield equation, with that estimated from the kill ratio:

$$LY_i = \exp(-FP_i * DD_i) = \frac{T_G(T - T_i)}{T \cdot (T_G - T_{Gi})} \tag{3-11}$$

where DD_i is the defect density of defect type *i* per die, *T* is the total number of dice, T_G is the total number of good dice, T_i is the total number of dice with at least one defect type *i* on them, and T_{Gi} is the total number of good dice with at least one defect type *i* on them. Rearranging Equation (3-11), we have:

$$FP_i = \frac{-\ln[\frac{T_G(T - T_i)}{T(T_G - T_{Gi})}]}{DD_i} \tag{3-12}$$

A total of four runs were done, with FP values ranging from 0.15 to 0.001. This range reflects the FP values based on actual fab defect data obtained in the previous chapter. The FP values assigned for each run are shown in Table 3.2. Run 1 has the largest values of FP, with the values for FP assigned in each run decreasing to the smallest values in run 4.

Table 3.2 FP values assigned for each run

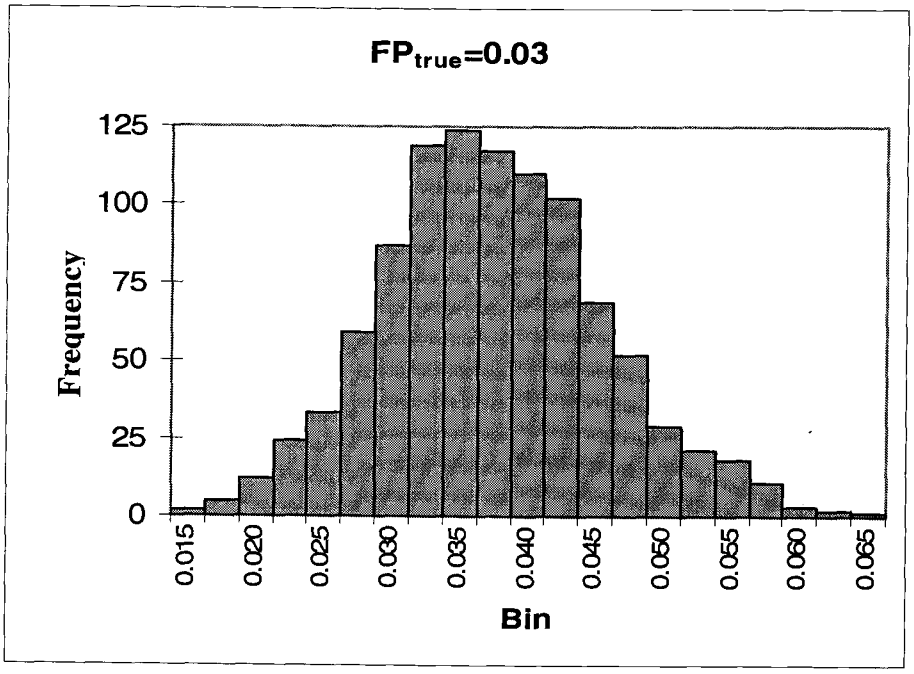
Run	FP_A	FP_B	FP_C
1	0.15	0.10	0.07
2	0.03	0.02	0.01
3	0.008	0.006	0.004
4	0.001	0.002	0.001

3.2 Results And Discussion

3.2.1 Bootstrap Sampling Distribution Results

Figure 3.3 shows two bootstrap sampling histograms from two separate samples taken from populations with FP_{true} values of 0.03 and 0.006. The respective sample FP estimates, FP_{samp}, are 0.0362 and 0.00757. For these two samples, the values of FP_{samp} are close to their respective FP_{true} values, and the estimated CI values based on all four bootstrap methods cover FP_{true}. In both cases, the sampling distributions are right-skewed, with the one for FP_{true} =0.006 more so. At smaller values of FP_{true}, some values of FP_{samp} are negative, and some values in the bootstrap sampling distribution are negative even when FP_{samp} is positive, as can be seen in Figure 3b. The negative estimate is a natural result of using the FP estimator of Equation (3-12). Although the negative values in the bootstrap sampling distributions are not realistic, they are kept, because this facilitates computation of the various bootstrap estimates. Only when the resulting confidence limit is negative, do we set its value to zero.

Figure 3.4 shows the bootstrap-estimated bias, variance, and 5th and 95th percentiles for various values of R, the number of bootstrap replications, for a sample drawn from a population with FP_{true}=0.008, whose sample FP estimate, FP_{samp}, has a value of 0.00762. The general trends seen in Figure 3.4 are seen for the other eleven defect types used in the simulation. As can be seen from Figure 3.4a, the magnitude of the bias tends to decrease and stabilize with increasing R, and is quite small



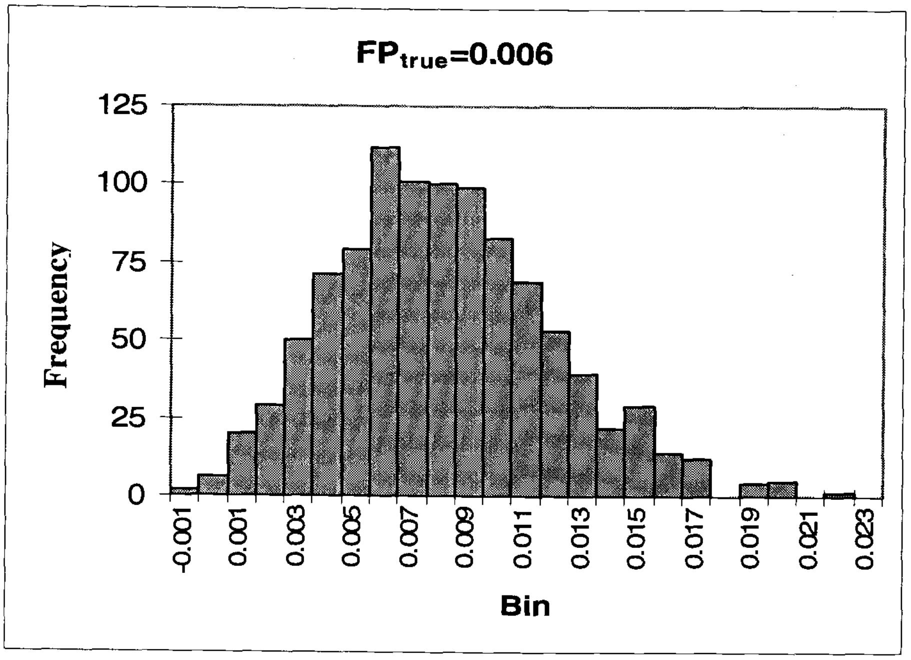
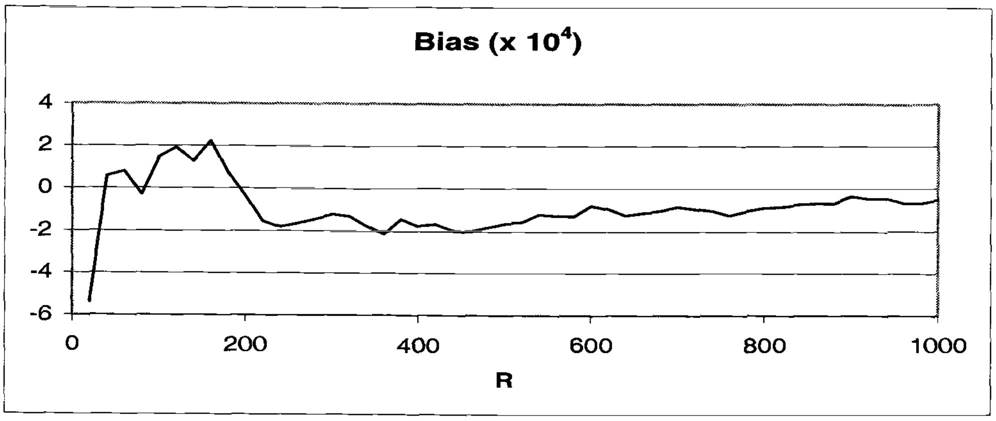
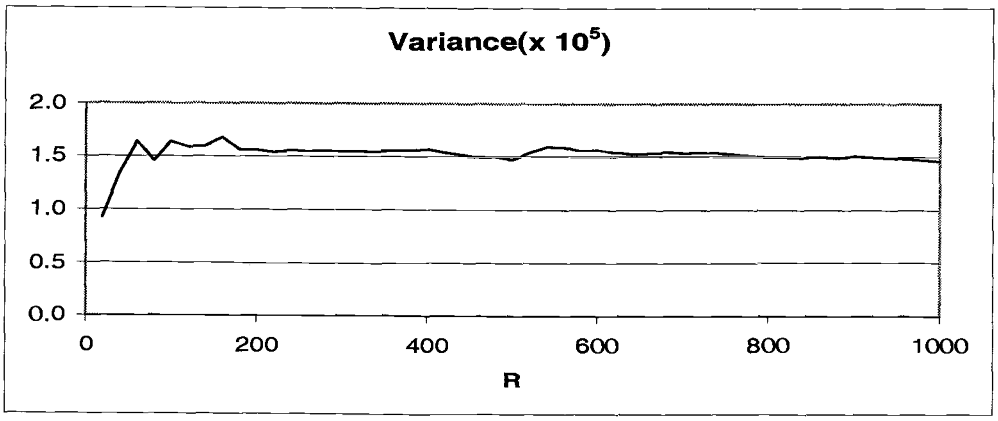


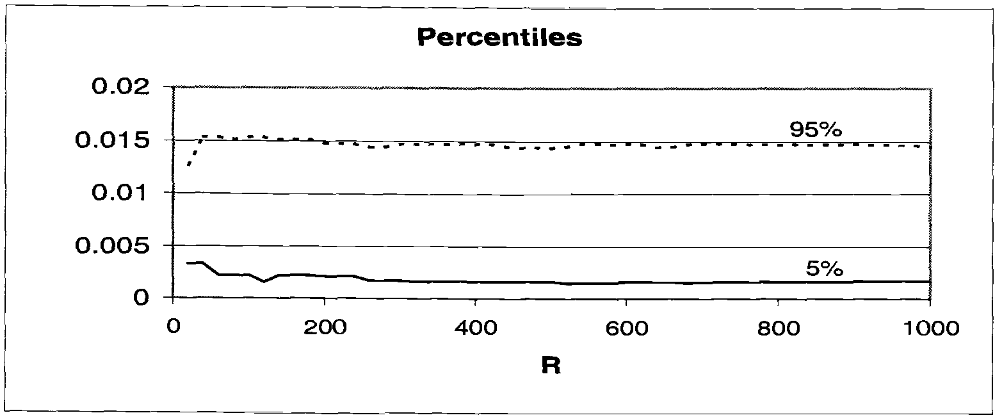
Figure 3.3 Two bootstrap sampling histograms for representative values of FP_{true}: a) FP_{true}=0.03, b) FP_{true}=0.006



a



b



c

Figure 3.4 Bootstrap estimated bias, variance, and 5th and 95th percentiles vs. the number of bootstrap replications for FP_{true}=0.008, at

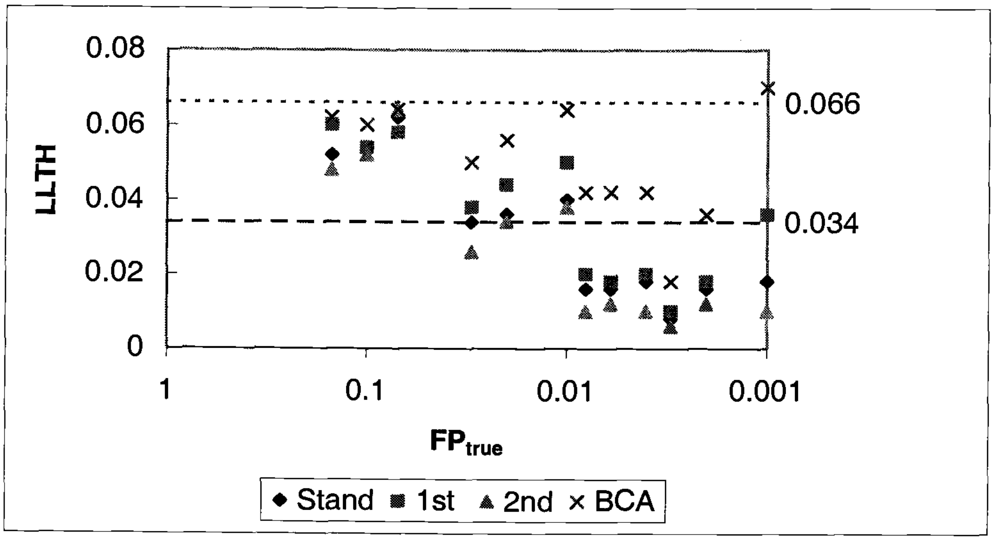
compared with the value of FP_{samp}. The biases are less than one percent of their respective estimates for all the defect types, at high values of R. The stabilization in the bias value as R increases is one reason that the number of replicates needs to be large. Therefore, we have good evidence that the bias of the estimate of FP using Equation (3-5) is negligible.

Figure 3.4b shows that the variance stabilizes more quickly than the bias. For example, at approximately R >200, the variance stabilizes, while at approximately R >600, the bias stabilizes. Figure 3.4c shows the 5th and 95th percentiles of the bootstrap estimates of FP. Like the variance, the percentiles also stabilize more quickly than the biases. Again, these general trends are seen for all the other eleven defect types. It is concluded that 1000 bootstrap replications should provide enough bootstrap estimates for accurate estimates of the bias, variance, and percentiles needed for the bootstrap CI estimation methods.

To assess whether a CI estimation method works for a given run, we estimated the proportion of the estimated confidence intervals that failed to capture the true value of the parameter FP (FP_{true}), for a large number of samples. Table 3.3 shows a summary of the performance of the four different methods of CI estimation for a 90% CI, for various values of FP_{true}. The performance of each method was measured by the proportion of failed CIs that were below FP_{true} (Upper Limit too Low, ULTL) and the proportion of failed CIs above FP_{true} (Lower Limit too High, LLTH). Each estimate of performance was determined from a total of 500 samples drawn from a particular population with the assigned FP_{true}. Thus, for each FP_{true} in Table 3.3, a total of, (500samples) x (1000 bootstrap estimates/ sample)=500,000 bootstrap estimates were used to determine the 500 CI's for each CI estimation method.

Ideally, when estimating a central 90% CI, each of the two categories, ULTL and LLTH, should be approximately 5% for a CI estimation method to be qualified as a success. However, we must also account for the variation in Table 3.3 due to generating only 500 estimates of CI. To determine acceptance criteria, we use a hypothesis test. If we let our null hypothesis be that the true proportion of ULTL or LLTH is 5%, that is, we have a procedure that estimates an exact central 90% confidence interval, an approximately 95% acceptance region for this null hypothesis can be estimated from a binomial distribution with p=0.05 and n=500. In this case, the proportion of ULTL or LLTH from 0.034 to 0.066 indicates we cannot reject the null hypothesis. Thus, if both ULTL and LLTH for a particular CI estimation method are between 0.034 and 0.066, we will count the method as a success. If either is outside this range, we will count the method as a failure.

Figure 3.5 shows the proportions of LLTH and ULTL for each of the four methods of CI estimation, along with the lines showing the upper and lower success criteria. Using these criteria, we see that the standard method, the first percentile, and the second percentile method all consistently fail below FP_{true}=0.01. The BCA method, however, does not consistently fail until FP_{true}=0.003 or below. The standard, the first percentile, and the second percentile methods work fairly well for runs 1 and 2. In run 3, however, these three methods all fail, while the BCA method is still successful. In run 4, where the values of FP_{true} are set to 0.003, 0.002, and 0.001,



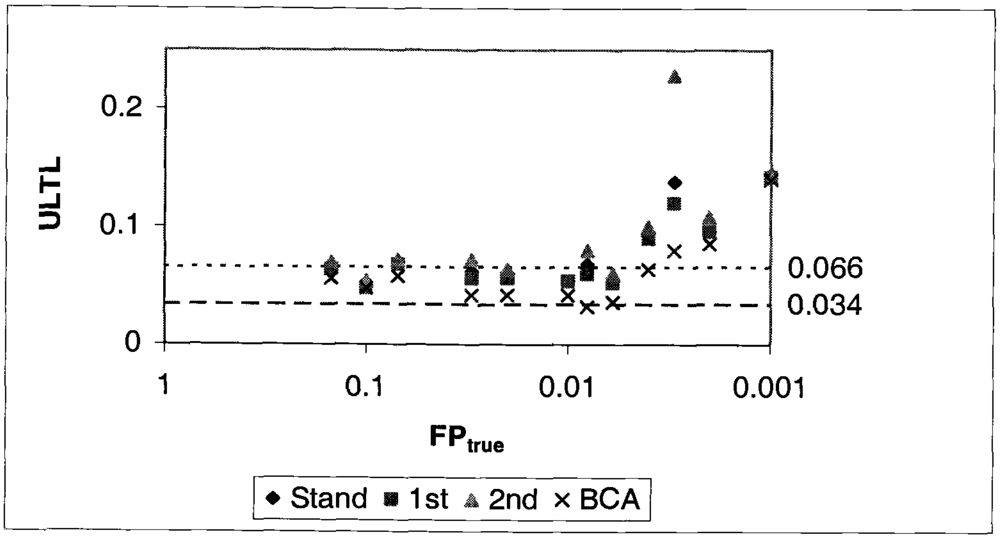


Figure 3.5 The proportions of LLTH and ULTL vs. FP_{true} for the four methods of CI estimation

all four methods fail. Moreover, when they fail, the LLTH tends to predict too few ranges of CI out of range while the ULTL tends to predict too many.

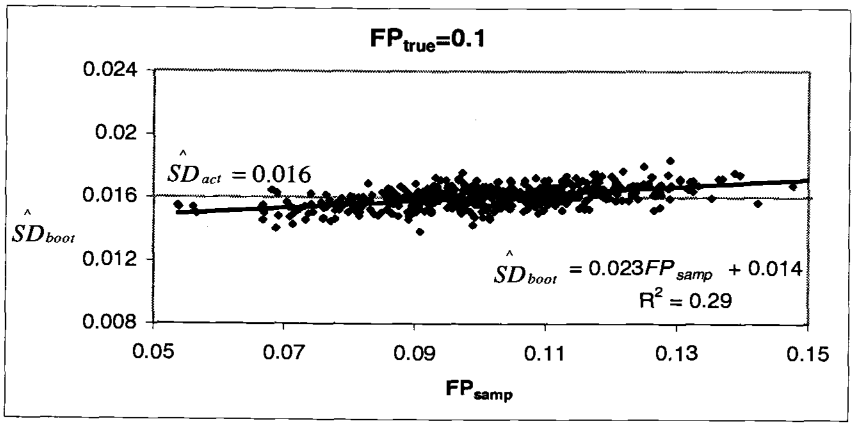
Table 3.3 Performance of four different methods of CI estimation for a 90% CI in terms of the proportion of failed CIs

Run	FP_{true}		**Standard**	**1st Percentile**	**2nd Percentile**	**BCA**
1	**0.15**	LLTH	0.052	0.06	0.048	0.062
		ULTL	0.066	0.064	0.070	0.056
	0.10	LLTH	0.054	0.054	0.052	0.060
		ULTL	0.052	0.048	0.054	0.048
	0.07	LLTH	0.062	0.058	0.064	0.064
		ULTL	0.070	0.068	0.072	0.058
2	**0.03**	LLTH	0.034	0.038	0.026	0.050
		ULTL	0.062	0.056	0.072	0.042
	0.02	LLTH	0.036	0.044	0.034	0.056
		ULTL	0.060	0.056	0.064	0.042
	0.01	LLTH	0.040	0.050	0.038	0.064
		ULTL	0.054	0.054	0.054	0.042
3	**0.008**	LLTH	0.016	0.020	0.010	0.042
		ULTL	0.068	0.060	0.080	0.032
	0.006	LLTH	0.016	0.018	0.012	0.042
		ULTL	0.054	0.052	0.060	0.036
	0.004	LLTH	0.018	0.020	0.010	0.042
		ULTL	0.094	0.090	0.100	0.064
4	**0.003**	LLTH	0.008	0.010	0.006	0.018
		ULTL	0.138	0.120	0.228	0.080
	0.002	LLTH	0.016	0.018	0.012	0.036
		ULTL	0.096	0.096	0.108	0.086
	0.001	LLTH	0.018	0.036	0.010	0.070
		ULTL	0.144	0.142	0.144	0.140

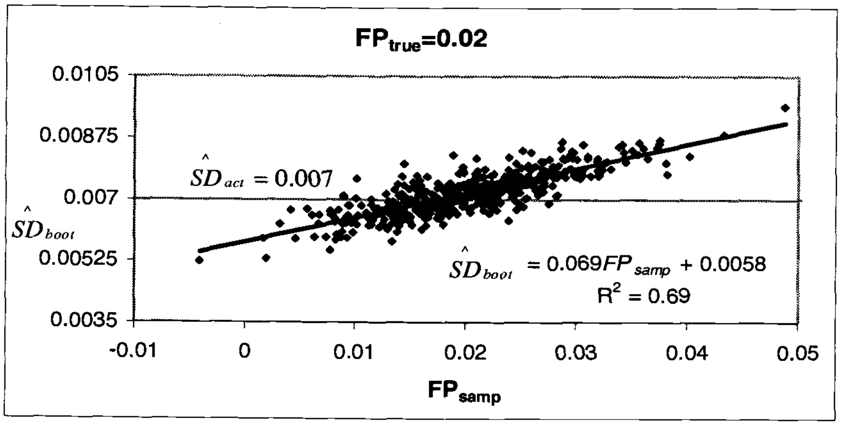
3.2.2 Performance of the Basic Pivotal Methods

To understand the limitations of the standard, the first percentile, and the second percentile methods, the basic assumption employed by these methods must be recalled, which is that the standard deviation of the bootstrap distribution is approximately independent of the value of the sample estimate. Figure 3.6 shows the standard deviation of the bootstrap FP estimates ($\hat{SD}_{boot}$) versus the sample FP values (FP_{samp}) for three representative values of FP_{true} from Table 3.2. Each plot in Figure 3.6 shows 500 values of $\hat{SD}_{boot}$ estimated from 500 samples, each sample being drawn from the population with the specified FP_{true}. The sample standard deviation, $\hat{SD}_{act}$, is also shown. It represents the best estimate of the standard deviation of the actual sampling distribution of FP_{samp} for each value of FP_{true}. To facilitate comparison, the range of the y-axis for each plot is scaled to match $\hat{SD}_{act}$. Figure 3.6 indicates that the $\hat{SD}_{boot}$ increases with FP_{samp}. We also see that the slope of the best-fit line increases with decreasing FP_{true}.

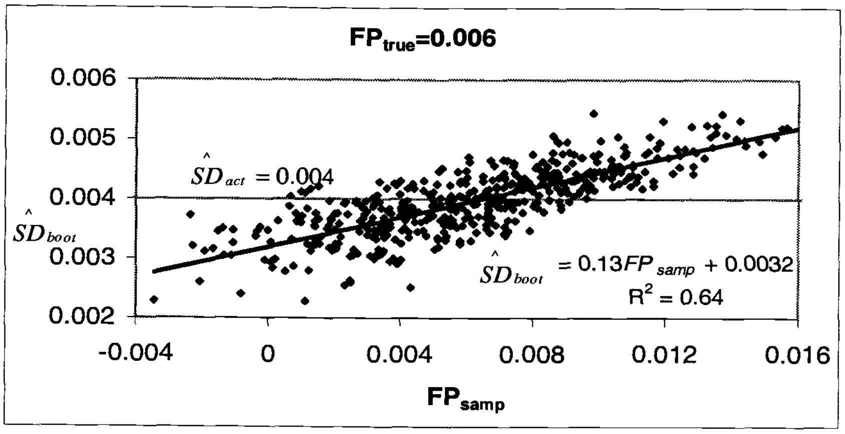
Table 3.4 presents the equations of the best-fit lines of $\hat{SD}_{boot}$ to $FP_{samp,}$ for each of the twelve defect types from Table 3.2. It also gives the values of the sample standard deviation of the 500 values of FP_{samp}, for each value of $FP_{true.}$ The slope of the best-fit line gives us an indication of the change of the standard deviation of FP_{samp} with FP_{true} for each of the four runs. Table 3.4 shows that the slope of the $\hat{SD}_{boot}$ increases by approximately 6 fold from run 1(high FP_{true}) to run 3(low FP_{true}),



a



b



c

Figure 3.6 The 500 standard deviations of bootstrap distributions ($\hat{SD}_{boot}$) versus FP_{samp} for three representative values of FP_{true}.

and by approximately 12 fold from run 1 to run 4. However, within the same run, the slopes are relatively constant. Thus, the assumption of constant variance for all values of FP_{samp} becomes worse from run 1 to run 4, and we should expect the CI estimation based on the basic pivotal methods to become worse. This expectation is confirmed by the results in Table 3.3, where the standard, first percentile, and second percentile methods all fail beginning at run 3.

Examination of Figure 3.5 reveals that for the basic pivotal methods the proportion of LLTH decreases as FP_{true} decreases. Conversely, the proportion of ULTL increases as FP_{true} decreases. For example, at FP_{true}=0.15, and FP_{true}=0.10, these two proportions are roughly the same, at approximately 5%, and these methods are successful; on the other hand, at FP_{true}=0.004, the proportion of LLTH is only about 1% to 2%, while that of ULTL is 9 to 10%. As Figure 3.6 shows, the further away the value of FP_{samp} from FP_{true}, the more $\hat{SD}_{boot}$ deviates from the $\hat{SD}_{act}$. When FP_{samp} is larger than FP_{true}, $\hat{SD}_{boot}$ tends to be larger than $\hat{SD}_{act}$, and when FP_{samp} is smaller than FP_{true}, $\hat{SD}_{boot}$ tends to be less than $\hat{SD}_{act}$.

Figure 3.7 schematically illustrates the case when the pivotal approximation is valid, i.e., when $\hat{SD}_{boot}$ approximates $\hat{SD}_{act}$. Figure 3.7a represents a general sampling distribution, with the areas of the lower and upper α percentiles shaded. Figures 3.7b and 3.7c represent bootstrap distributions resulting from samples whose FP_{samp} have small and large values, t_{sm} and t_{lg}, respectively. Consider a CI estimated,

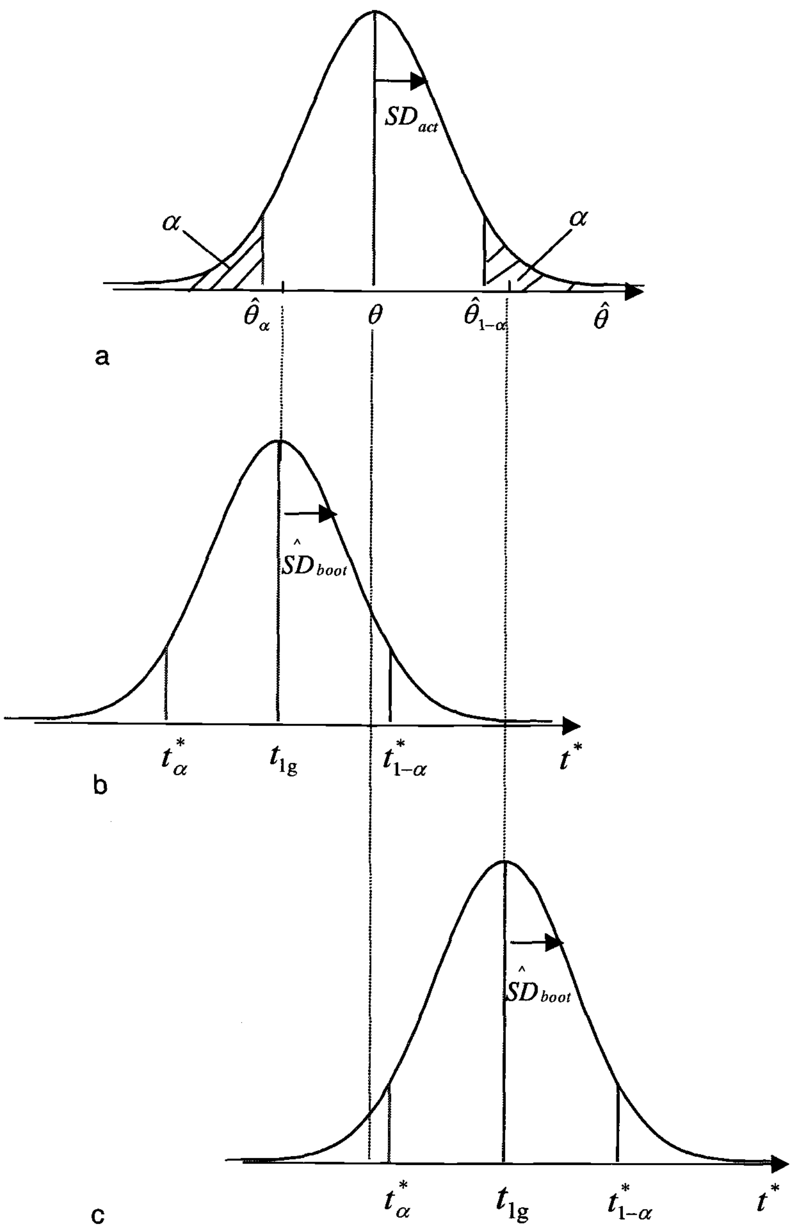


Figure 3.7 Bootstrapping estimates when the pivotal approximation is valid

Table 3.4 The equation of the best-fit line of $\hat{SD}_{boot}$ to FP_{samp}, and the estimated sample SD for various values of FP_{true}

Run	FP_{true}	**Fit of** $\hat{SD}_{boot}$	$\hat{SD}_{act}$
1	0.15	0.024 FP_{samp} +0.013	0.018
	0.10	0.023 FP_{samp} +0.014	0.016
	0.07	0.024 FP_{samp} +0.014	0.016
2	0.03	0.070 FP_{samp} +0.0055	0.0074
	0.02	0.069 FP_{samp} +0.0058	0.0071
	0.01	0.066 FP_{samp} +0.0060	0.0066
3	0.008	0.14 FP_{samp} +0.0030	0.0038
	0.006	0.13 FP_{samp} +0.0032	0.0036
	0.004	0.13 FP_{samp} +0.0032	0.0036
4	0.003	0.25 FP_{samp} +0.0016	0.0023
	0.002	0.26 FP_{samp} +0.0016	0.0022
	0.001	0.22 FP_{samp} +0.0018	0.0021

for example, by the first percentile method. When the pivotal approximation is true, any CI, estimated from a sample whose sample estimate, t, is taken from the area between the shaded areas, would cover the true value of the parameter, θ. On the other hand, any CI estimated from a sample with sample estimate t taken within the shaded regions would fail. Thus, an approximately $1\text{-}2\alpha$ proportion of the CIs would succeed.

For example, Figure 3.7b shows a bootstrap distribution with mean centered at t_{sm}, connected by the dotted line to the same value in Figure 3.7a. In this case, we see that the resulting *UCL* estimated by the first percentile method, $t^*_{1-\alpha}$, will just be large enough so that the resulting CI will contain θ. Thus, we see that the first percentile works only if the bootstrap distribution has the same spread, or variance, as the actual sampling distribution; i.e., the quantity $\theta - \hat{\theta}_\alpha$ is equal to $t^*_{1-\alpha} - t_{sm}$. An example of a case when the CI fails is shown in Figure 3.7c. The estimate is taken from the shaded region, where t_{lg} is an estimate that is slightly larger than $\hat{\theta}_{1-\alpha}$. In this case, we see that the resulting LCL estimated by the first percentile method, t^*_α, will not bracket θ, as seen by the dotted line connecting θ to the bootstrap sampling distribution of Figure 3.7c. This result again is due to the spread of the bootstrap distribution being the same as that of the actual sampling distribution; i.e., the quantity $\hat{\theta}_{1-\alpha} - \theta$ is equal to $t_{\mathrm{lg}} - t^*_\alpha$. Thus, when the pivotal approximation is valid, the proportion of ULTL and LLTH is approximately α. Similar arguments apply in the case of the standard and second percentile methods.

Figure 3.8 is analogous to Figure 3.7 except that it shows what happens when the variance of the bootstrap distribution does not approximate that of the actual sampling distribution. Figure 3.8b shows a bootstrap distribution whose variance is lower than that of the actual sampling distribution, represented in Figure 3.8a, while Figure 3.8c shows a bootstrap distribution whose variance is larger. Figure 3.8b shows that when the $\hat{SD}_{boot}$ under-estimates the $\hat{SD}_{act}$, the resulting UCL estimated by the first percentile method, $t^*_{1-\alpha}$, will not bracket θ. Thus the dotted line from $t^*_{1-\alpha}$ in Figure 3.8b is to the left of θ in Figure 3.8a. In summary, some CI ranges estimated from samples with sample estimate *t* taken from the region $\hat{\theta}_\alpha < \hat{\theta} < \theta$ would fail, where they should succeed. Thus Figure 3.8b graphically represents the consequence

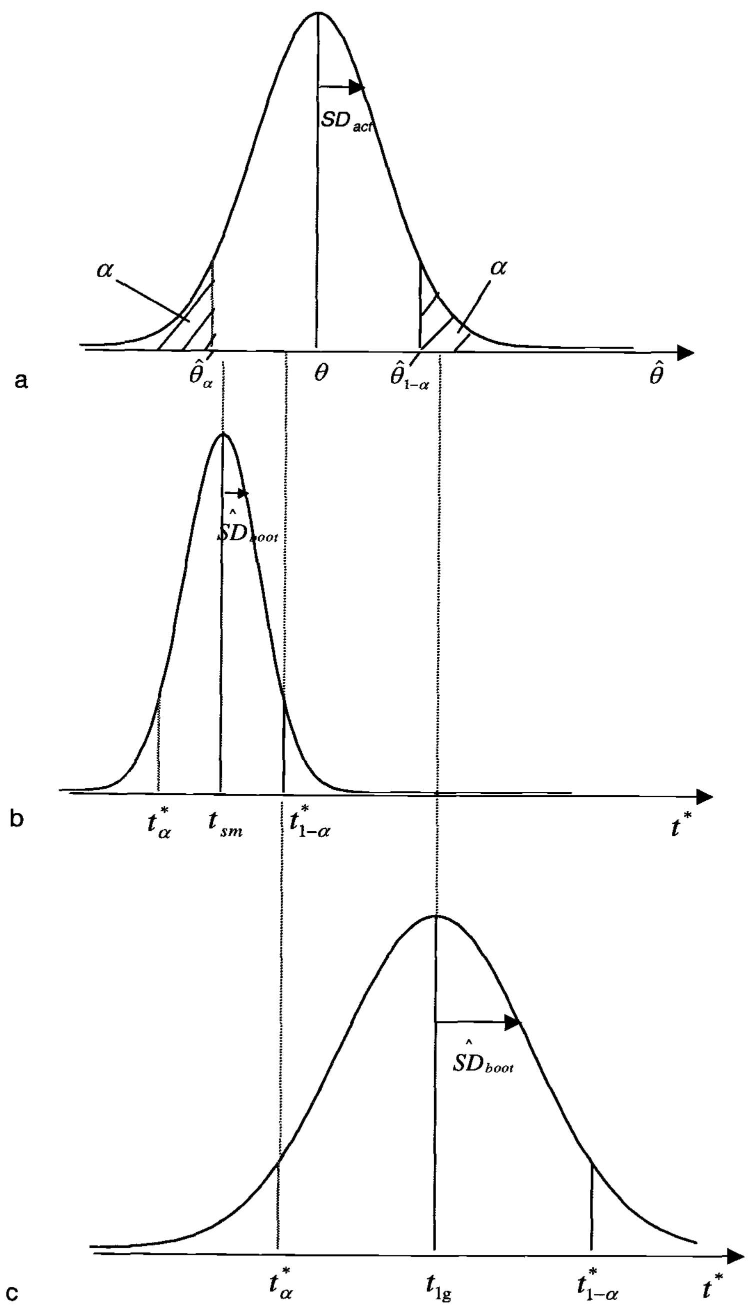


Figure 3.8 Bootstrapping estimates when the pivotal approximation is not valid. 8b shows the case when $\hat{SD}_{boot} < SD_{act}$. 8c shows the case when $\hat{SD}_{boot} > SD_{act}$.

of the quantity $t^*_{1-\alpha} - t_{sm}$ being less than $\theta - \hat{\theta}_\alpha$, which is that the probability of ULTL will be higher than α. This result is seen at small values of FP_{true}.

Conversely, Figure 3.8c shows a case where $\hat{SD}_{boot}$ over-estimates $\hat{SD}_{act}$ when estimating from a sample whose estimate t is larger than θ. Some CI ranges estimated from samples with sample estimates from the region $\hat{\theta}_{1-\alpha} < \hat{\theta}$ would succeed, where they should fail. This case is represented by the dotted line from t^*_α, the LCL of the first percentile method, being to the left of θ in Figure 3.8a. Thus Figure 3.8c graphically represents the consequences of the quantity $t_{1g} - t^*_\alpha$ being greater than $\hat{\theta}_{1-\alpha} - \theta$. This case results in the probability of LLTH being lower than α, as seen at low values of FP_{true}. Similar arguments can be used in the case of the standard and second percentile methods to show that the probability of ULTL will be higher than α and the probability of LLTH will be lower than α when the SD_{boot} increases with FP_{samp}.

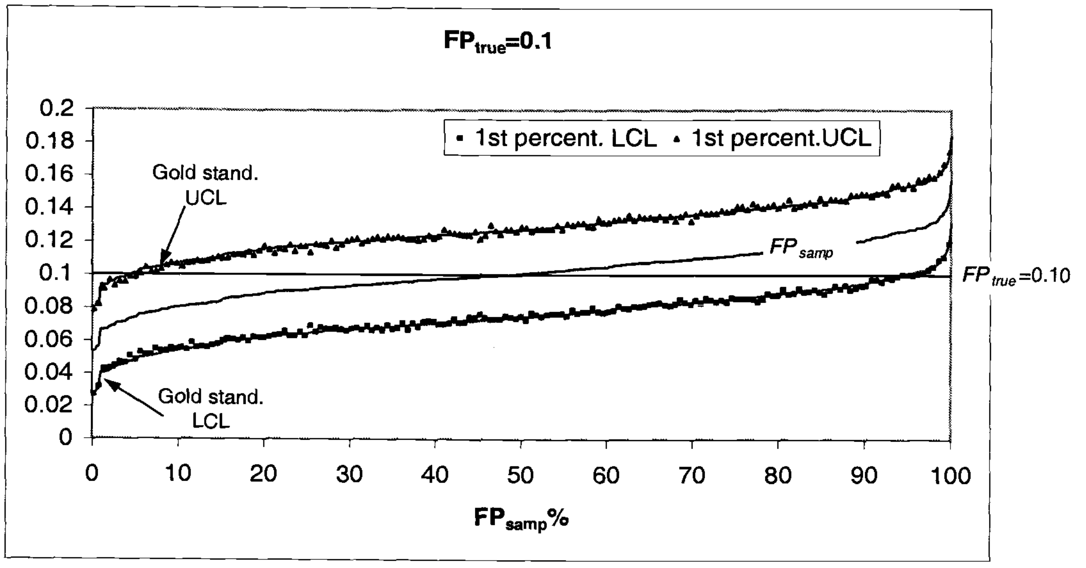
We can estimate the quantities $a = \theta - \hat{\theta}_\alpha$ and $b = \hat{\theta}_{1-\alpha} - \theta$ of Figure 3.1 directly from the 500 values of FP_{samp} for a given FP_{true} and compare them with those estimated by each of the basic pivotal methods. Let $\hat{\theta}_{1-\alpha} - \theta$ be estimated by $FP_{samp,1-\alpha} - \overline{FP}_{samp}$, where $FP_{samp,1-\alpha}$ is the (1-α)th percentile of the 500 values of FP_{samp}, and $\overline{FP}_{samp}$ is the average. In a similar way, we can estimate $\theta - \hat{\theta}_\alpha$ by $\overline{FP}_{samp} - FP_{samp,\alpha}$. Since we have 500 original samples, these two estimates should be fairly accurate, as should the confidence limits based on them. For any value of FP_{samp}, we calculate the lower and upper confidence limits as follows:

$$LCL = FP_{samp} - (FP_{samp,1-\alpha} - \bar{F}P_{samp}) \tag{13a}$$

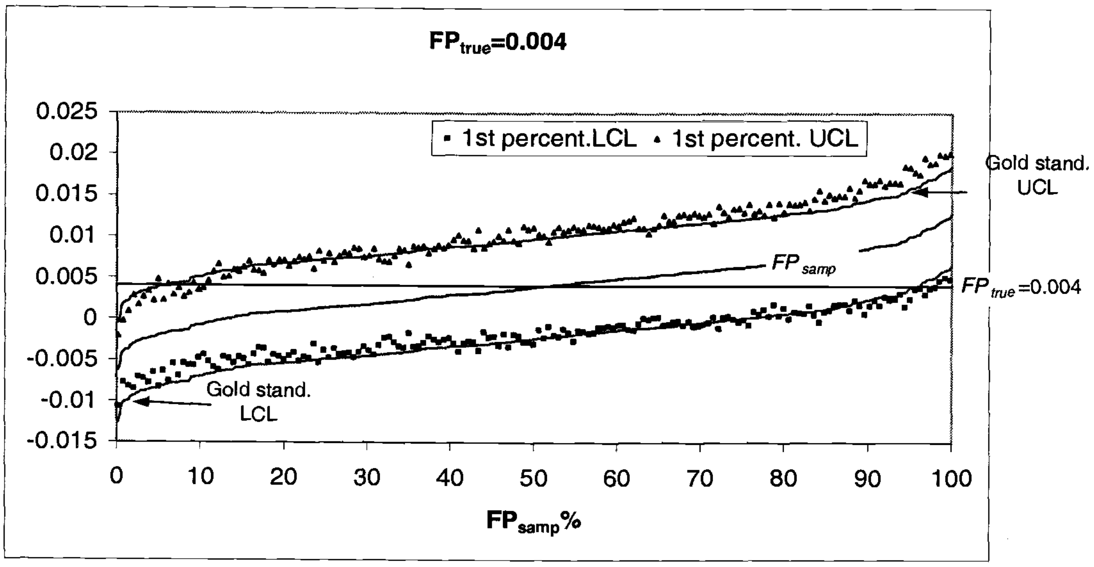
$$UCL = FP_{samp} + (\bar{F}P_{samp} - FP_{samp,\alpha}) \tag{13b}$$

We term the estimates represented by Equations (3-13a) and (3-13b) our "gold standard", since in practice we do not have 500 samples, but only one.

Figure 3.9 compares confidence limits estimated by the first percentile method with those by the gold standard, Equations (3-13). Only every third of the 500 confidence limits estimated by the first percentile method are shown to preserve clarity. Figure 3.9a represents samples drawn from the population with FP_{true}=0.1. Figure 3.9a shows that the values for LCL and UCL estimated from the first percentile method fall over the LCL and UCL curves estimated from Equations (3-13). Analogous plots for the standard and second percentile method show similar results. These results indicate that the quantities $\hat{\theta}_{1-\alpha} - \theta$ and $\theta - \hat{\theta}_{\alpha}$ of Figure 3.1 are approximately pivotal in the case of FP_{true}=0.1. Figure 3.10 shows the 500 values of $\hat{SD}_{boot}$, each estimated from one original sample, compared with $\hat{SD}_{act}$, vs. percentiles of FP_{samp}. Figure 3.10a represents values for FP_{true}=0.1. While there exists a slight positive slope, all values of $\hat{SD}_{boot}$ are still approximately centered about the horizontal line representing $\hat{SD}_{act}$. Thus the quantities $\hat{\theta}_{1-\alpha} - \theta$ and $\theta - \hat{\theta}_{\alpha}$ are

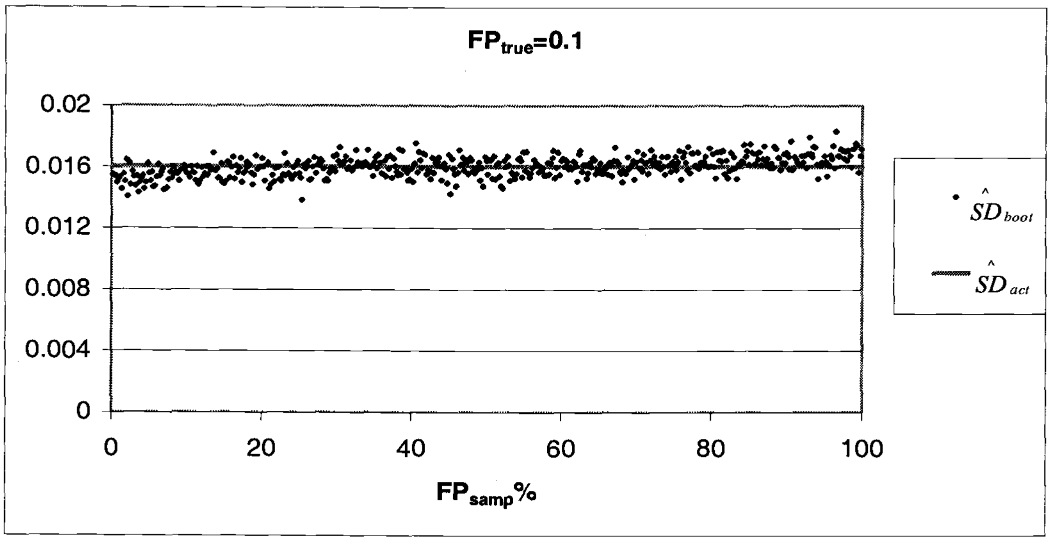


a

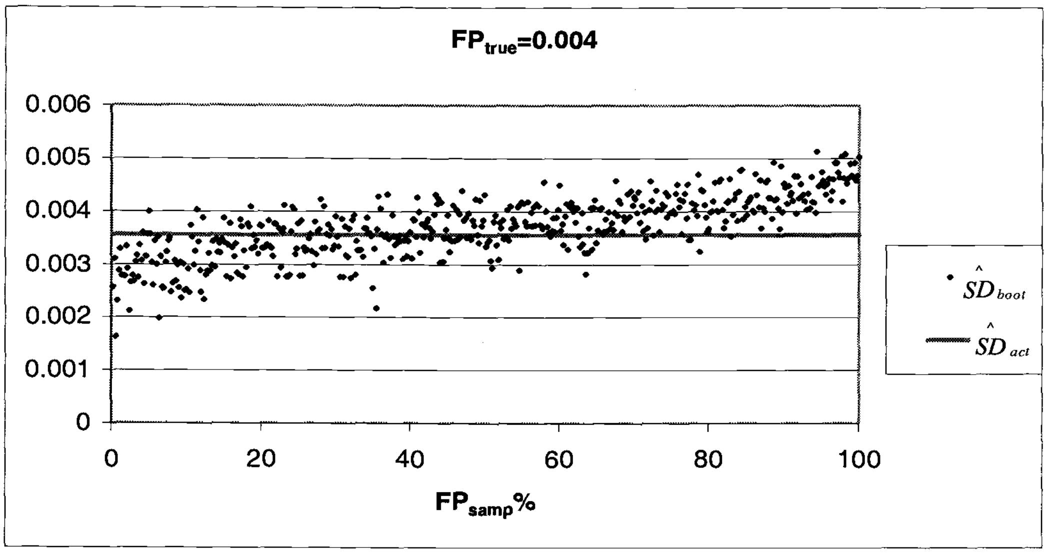


b

Figure 3.9 Comparison of UCL and LCL estimated by the first percentile method to that of the "gold standard" estimated from the actual sampling distribution for a) FP_{true}=0.1, b) FP_{true}=0.004.



a



b

Figure 3.10 $\hat{SD}_{boot}$ and $\hat{SD}_{act}$ vs. percentile of FP_{samp} for a) FP_{true}=0.1, b) FP_{true}=0.004.

indeed approximately pivotal in the case of FP_{true}=0.1, and consequently the proportions of ULTL and LLTH reported in Table 3.3 are close to 5% for FP_{true}=0.1.

Figures 3.9b and 3.10b are analogous plots to Figure 9a and 10a, respectively, for the case of FP_{true}=0.004. In Figure 3.9b, the points representing the LCL and UCL estimated by the first percentile method deviate from the LCL and UCL curves estimated by Equations (3-13) at low and high FP_{samp} percentiles. Analogous plots for the standard and second percentile method show similar behavior. These results can be attributed to the increased rate of change of $\hat{SD}_{boot}$ with FP_{samp}, which is nearly six times greater for FP_{true}=0.004 than for FP_{true}=0.1. Figure 3.10b shows that this increased rate of change of $\hat{SD}_{boot}$ with FP_{samp} increases the deviations of the values of $\hat{SD}_{boot}$ from $\hat{SD}_{act}$, at the more extreme values of FP_{samp}. This increased deviation from $\hat{SD}_{act}$ manifest in Figure 3.9b. At high percentiles of FP_{samp}, the estimated CIs are wider than those estimated by Equations (3-13), since the quantity $\hat{\theta}_{1-\alpha} - \theta$ is over-estimated. At low percentiles, the converse is true, since this same quantity is under-estimated.

The first percentile estimated UCL and LCL points shown in Figure 3.9b are consistent with the trends seen in Figure 3.5. In Figure 3.9b, the underestimation of $\theta - \hat{\theta}_{\alpha}$ tends to make the points representing the UCL estimated by the first percentile method fall below the FP_{true}=0.004 line at a point greater than the 5th percentile of FP_{samp}, reflecting the increase in ULTL above 5% at lower values of FP_{true}, as seen in Figure 3.5. For the points representing the LCL estimated by the first percentile method, they tend to cross the FP_{true}=0.004 line at a point greater than the 95th percentile of FP_{samp}, reflecting the decrease in LLTH below 5% at higher values of FP_{true}, as seen in Figure 3.5. Similar figures for the second and standard percentile method show similar behavior. Thus the effects of over and under –estimating the pivotal quantity on the proportions of LLTH and ULTL are seen directly from the plot.

The success of the first percentile method in the case of FP_{true}=0.1 and failure in the case of FP_{true}=0.004 have been accounted for. The same principle applies to the other values of FP_{true} and to the other basic pivotal methods. Next, the three pivotal methods are compared to one another. By the criteria that have been set, all three methods perform well enough to be generally classified as successful in runs 1 and 2; nevertheless the first percentile method displays better performance. The proportions of LLTH and ULTL of the first percentile method are closer to 5% than the other two methods, especially in run 2. The improved performance results from the slightly right-skewed shape of the bootstrap distributions.

Table 3.5 shows the average "shape factor" of the bootstrap distributions for the lower 50 values (lower 10%) and upper 50 values (upper10%) of FP_{samp} for each FP_{true}. The "shape factor", *Sh*, is measured by,

$$Sh = (FP^*_{1-\alpha} - \overline{FP}^*)/(\overline{FP}^* - FP^*_{\alpha}) \qquad (3\text{-}14)$$

A "shape factor" greater than one will tend to come from a right- skewed distribution. In Table 3.5, the average values of *Sh* are all greater than one. If the distributions were symmetric, we would expect an approximately equal number of occurrences below one as above. Thus it is likely that the bootstrap distributions are right- skewed for each FP_{true}. Table 3.5 also shows that the degree to which the bootstrap distributions are right-skewed increases with decreasing FP_{true}, from approximately 1.0 in run 1 to 1.1 in run 3.

Table 3.5 Average "shape factor" of the bootstrap distributions for the lower and upper 10% values of FP_{samp}

FP_{true}	$\bar{Sh}$	
	lower 10%	**upper 10%**
0.15	1.022	1.012
0.10	1.012	1.002
0.07	1.022	1.013
0.03	1.048	1.047
0.02	1.042	1.051
0.01	1.039	1.051
0.008	1.145	1.092
0.006	1.084	1.102
0.004	1.057	1.106

Figure 3.11 schematically shows the effect of such a right skewed distribution on CI estimation. In Figure 3.11a, we have a right-skewed sampling distribution. In Figure 3.11b, we have a bootstrap distribution resulting from a sample with estimate t_{sm}, taken from $\hat{\theta}_\alpha < \hat{\theta} < \theta$. We see that the UCL for the first percentile method, $t^*_{1-\alpha}$, still covers θ, but that the UCL for the second percentile method, $t_{sm} + (t_{sm} - t^*_\alpha)$, will just fall short of θ. This result will only occur if the variance of the bootstrap distribution is smaller than that of the actual sampling distribution when sampling is

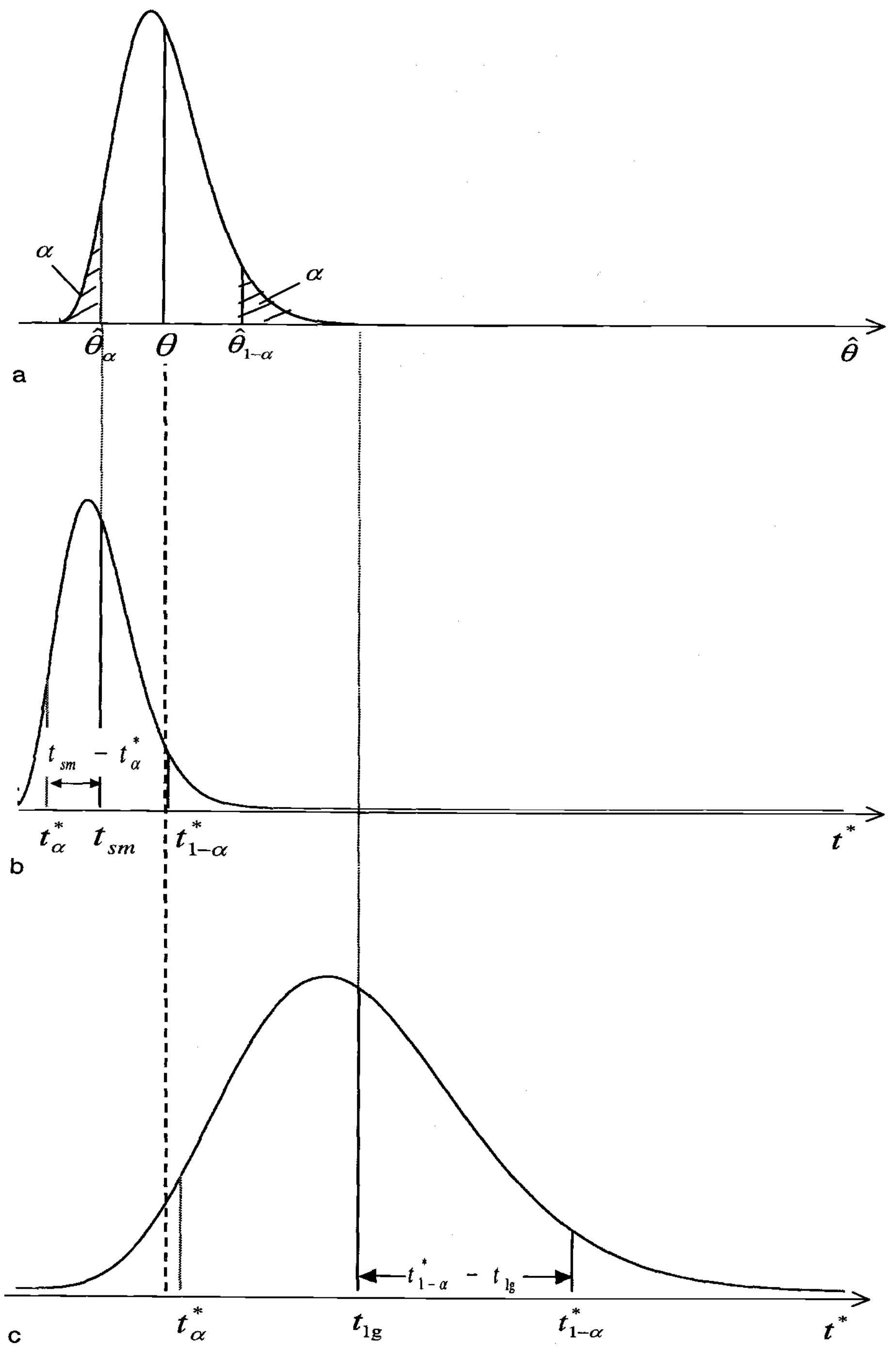


Figure 3.11 Effect of the right-skewed shape of the sampling distribution on the coverage of the first and second percentile methods.

such that $\hat{\theta}_{\alpha} < \hat{\theta} < \theta$. Thus the proportion of ULTL will be higher for the second percentile method than for the first percentile method, as shown in Figure 3.5.

On the other hand, Figure 3.11c shows a bootstrap distribution resulting from a sample with the estimate t_{1g}, taken from $\hat{\theta}_{1-\alpha} < \hat{\theta}$. The LCL for the first percentile method, t_{α}^{*}, will, correctly, not cover θ. However, the LCL for the second percentile method, $t_{1g} - (t_{1-\alpha}^{*} - t_{1g})$, will just cover θ. Again, this will only occur if the variance of the bootstrap distribution is larger than that of the actual sampling distribution when sampling from $\hat{\theta}_{1-\alpha} < \hat{\theta}$. Thus we see that the proportion of LLTH will be lower for the second percentile method than for the first percentile method. This result is also reflected in Figure 3.5. The standard method would have proportions of LLTH and ULTL between that of the first and second percentiles, since it uses the variance of the entire bootstrap distribution- not just the distance of one side or the other about $\hat{\theta}$- to estimate the pivotal distance $\hat{\theta}_{\alpha} - \theta$.

3.2.3 Performance of the BCA Method

The basic pivotal methods fail at low values of FP_{true} because the bootstrap estimated distances a and b vary with the actual distances a and b in Figure 3.1. These discrepancies are a result of the change of $\hat{SD}_{boot}$ with FP_{samp}. Out of the four methods of CI estimation applied in this study, only the BCA method recognizes the possibility of this change, by specifying an acceleration constant, a_c, With a_c specified as an additional parameter, the transformation function assumed in the BCA method, given by Equation (3-6), need only be normalizing, but not necessarily variance stabilizing, where the mean and variance are independent. The parameter β_0 allows the normal transformation of distributions that are skewed. Thus the effect of the additional parameters a_c and β_0 is to make the BCA assumption more general than that of the basic pivotal methods, allowing the BCA method to work for a wider class of problems. [23]

Table 3.6 shows the averages of the estimated acceleration and bias constants, $\hat{a}_c$ and $\hat{\beta}_0$, respectively, based on the 500 samples drawn, for each FP_{true}. It also shows the averages of the estimated percentiles, $\hat{\alpha}_1$ and $\hat{\alpha}_2$, in Equations (3-8). Table 3.6 shows that $\hat{a}_c$ and $\hat{\beta}_0$ are both greater than 0. This result is expected for the acceleration constant since we have seen that the slope of SD_{boot} vs. FP_{samp} is positive. It is also expected for the bias constant, the measure of the bias of the normalized estimator. The more right-skewed the untransformed distribution, the greater the bias of the normalized estimator, and the more positive the value of the bias constant. This will be the case even if the untransformed estimator does not have bias, as defined in Equation (3-5). In the present case, it seems reasonable to conclude that the positive values of the bias constants are due to the right-skewed bootstrap sampling distributions, as the biases as measured by Equation (3-5) are negligible. This conclusion is bolstered by Table 3.6, which shows that the average values of $\hat{\beta}_0$ are greatest in Run 3, where the shape factors of the bootstrap distributions are also greatest, as shown in Table 3.6. The values of the bias constants in Run 4 may not be meaningful as the BCA method fails in Run 4.

Table 3.6 Average values of the BCA parameters estimated from 500 samples for each value of FP_{true}.

Run	FP_{true}	β_0	a_c	α_1	α_2
1	0.15	0.010	0.008	0.055	0.953
	0.10	0.011	0.008	0.055	0.954
	0.07	0.007	0.008	0.054	0.953
2	0.03	0.022	0.025	0.062	0.961
	0.02	0.023	0.025	0.063	0.961
	0.01	0.023	0.024	0.062	0.960
3	0.008	0.050	0.047	0.076	0.970
	0.006	0.050	0.047	0.076	0.970
	0.004	0.046	0.044	0.075	0.969
4	0.003	0.021	0.072	0.092	0.965
	0.002	0.020	0.067	0.091	0.963
	0.001	0.010	0.055	0.086	0.958

The average acceleration constants tend to increase with decreasing FP_{true}. For example, at FP_{true}=0.15, it is approximately 0.008, but at FP_{true}=0.004, it is about 0.047, almost a 6 fold increase. This trend is a reflection of the increasing slope of the best-fit line of the SD_{boot} versus FP_{samp} data shown in Table 3.4, which also displays a 6 fold increase. Whether this is a coincidence, or there is a direct proportionality between $a_c = \frac{d\sigma_\phi}{d\phi}$ and $\frac{d\sigma_\theta}{d\theta}$ needs to be further investigated.

The positive values of the acceleration and bias constants each shift the bootstrap percentiles in the same rightward direction, as seen in Equations (3-8), leading to $\hat{\alpha}_1$ and $\hat{\alpha}_2$ being greater than 5% and 95%, respectively. Table 3.6 shows this to be the case. The increase in $\hat{a}_c$ and $\hat{\beta}_0$ in turn is reflected in the increase in $\hat{\alpha}_1$ and $\hat{\alpha}_2$ away from 0.05 and 0.95, as FP_{true} decreases.

To see how the BCA method outperforms the basic pivotal methods in the present case, let us refer back to Figure 3.8. In Figure 3.8b, we see that although the estimate t_{sm} is taken from the region $\hat{\theta}_\alpha < \hat{\theta} < \theta$, the UCL estimated by the first percentile method is not greater than θ, since $\hat{SD}_{boot} < SD_{act}$. Those estimated by the standard and second percentile methods will fall even further short of θ due to the right-skewed bootstrap distribution. For the BCA method $\hat{a}_c > 0$ and $\hat{\beta}_0 > 0$, and we see from Equation (3-8) that $\alpha_2 > 1 - \alpha$. Therefore, the BCA estimate of the UCL, $t^*_{\alpha_2}$, will be greater than $t^*_{1-\alpha}$, and the confidence interval can cover θ. The BCA method can succeed even when the SD of the bootstrap sampling distribution is smaller than that of the actual sampling distribution, as long as the assumption of Equation (3-6) is met. In the present case, the probability of ULTL when the BCA method is applied will thus remain approximately 5% even while those of the basic pivotal methods exceed this percentage. For example, at FP_{true}=0.004 in run 3, the proportion of ULTL is approximately 0.1 in the case of the basic pivotal methods, but is 0.064 in the case of the BCA method. Similar arguments can be used to show that the percentage of LLTH will remain at approximately 5% even when $\hat{SD}_{boot} > SD_{act}$.

We see that the BCA method is superior to the basic pivotal methods because it allows for the SD of the sampling distribution to change with the value of the parameter being estimated, and accounts for skewed sampling distributions, where the mean and the median differ. When the sample size is large, or equivalently, when the number of fatal defects is large, both the basic pivotal methods and the BCA method work, because at large sample sizes, the SD of the bootstrap sampling distribution will approximate the SD of the actual sampling distribution, and the sampling distribution will be closer to being symmetric. But the BCA method will work at smaller sample sizes where the basic pivotal methods fail, that is, when the SD estimated from bootstrapping no longer approximates the SD of the actual sampling distribution, or the sampling distribution becomes skewed.

In run 4, however, we see that even the BCA method of CI estimation fails. In this run, the sample size is too small, as measured by the smaller values of FP_{true}. Since we have approximately 30-40 defects per wafer, at FP_{true}=0.1, we have: *30-40 x* 0.1= 3-4 fatal defects per wafer, on average. Thus we have approximately 60-80 fatal defects in each sample of 20 wafers. However, at FP_{true}=0.001, we have, on average, less than one fatal defect in each sample of 20 wafers. At these sample sizes, the acceleration and bias constants can no longer be accurately estimated, and, as a result these estimated BCA parameters can no longer be relied upon.

3.3 Conclusion

Bootstrapping has been applied to estimate the confidence interval for the fault probability of random defects at values representative of those measured in an integrated circuit fab. The standard, the first percentile, the second percentile, and the

BCA methods succeed at values of FP between 0.15 and 0.01 where the sample size is reasonably large. If the sample size is measured by the number of fatal defects, a reasonably large size might be between 60-80 fatal defects (FP=0.1) and 6-8 fatal defects (FP=0.01) per sample of 2000 dies. Additionally, the BCA method succeeds while the other three methods fail when the values of FP range from 0.008 to 0.004, and the number of fatal defects per sample is between 3 and 6. All four methods fail when the FP is 0.003 or less, where the number of fatal defects per sample is less than 3.

It was also observed that the standard deviation of the bootstrap sampling distribution increases with the sample estimate of FP, and the rate of this increase increases with decreasing values of the population FP. The bootstrap sampling distributions were also observed to be right-skewed, and become more right-skewed with decreasing values of the population FP. The success of the BCA method at lower values of FP_{true}, i.e., lower sample sizes, is explained based on its ability to account for the change in $\hat{SD}_{boot}$ with FP_{samp}, via the acceleration constant, and its ability to account for skewed sampling distributions that have a median bias, via the bias constant. The right-skewed sampling distributions also lead to the better performance of the first percentile method over the other basic pivotal methods. When the values of FP_{true} become too low, sample size limits the effectiveness of any method to make accurate CI estimates.

4. CONCLUSIONS AND FUTURE WORK

4.1 Conclusions

From the fault probability analysis based on the fab data, we see that the most reliable means of estimating FP are by means of critical area analysis, and by use of the defect limited yield equation with the kill ratio. The latter method of FP estimation is based on defects detected at a specific inspection step and represents the weighted average of the FP values of the defect mechanisms operating at the process steps immediately preceding the inspection step. The estimated FP values are based on the assumption of a Poisson distribution of the defects, but our analysis shows that this approximation produces negligible error even when clustering is present due to the relatively low defect density and FP values.

From our bootstrap analysis, we see that the standard, the first percentile, the second percentile, and the BCA methods succeed at values of FP between 0.15 and 0.01, if the number of die sampled is 2000, and the defect density is between 30 to 40 per wafer. These values of FP correspond to 60-80 fatal defects (FP=0.1) and 6-8 fatal defects (FP=0.01) per sample. Additionally, the BCA method succeeds while the other three methods fail when the values of FP range from 0.008 to 0.004, and the number of fatal defects per sample is between 3 and 6. All four methods fail when the FP is 0.003 or less, where the number of fatal defects per sample is less than 3. The success of the BCA method at lower FP is due to its incorporation of the acceleration factor and the bias constant. The acceleration constant accounts for the change in the standard deviation of the sampling distribution of the estimate with the population parameter being estimated. The bias constant accounts for skewed sampling distributions that have a median bias. The right-skewed sampling distributions also lead to the better performance of the first percentile method over the other basic pivotal methods. When the values of FP_{true} become too low, sample size limits the effectiveness of any method to make accurate CI estimates.

4.2 Suggestions for Future Work

The current work defines a defect type by where in the inspection process it is detected by the inspection tools. There is no way of knowing the specific proportions of each fault mechanism this defect type represents. Also due to limited sample sizes for larger defects, it is also not practical to classify the defect type by size.

The only way to further classify the current defect types into their components is by failure analysis at each of the inspections steps. Once an established database of defect signatures is in place, the use of an automatic defect classification system (ADC) to supplement the inspection tools would help in automating the classification of the defects on-line [24-26]. The only way to classify the defect types by size would be to increase the sample size, i.e., the total number of wafers inspected, so that FP estimates for larger sized defects would be statistically meaningful. Thus, for example, instead of classifying a defect type as ISEF only, it could be further broken down into ISEF-short-Sz5 or ISEF-break-Sz8.

From our bootstrap analysis we observed that the bootstrap-estimated standard deviation of the FP estimate changes with the estimated FP. This change appears linear with respect to the estimated FP. Furthermore the rate of this linear change increases with the decreasing values of the FP being estimated. It appears that the relationship may be modeled by the following equation:

$$\sigma_{\hat{FP}} = \frac{C_1}{\sqrt{n}} \cdot FP + C_2 \qquad (5\text{-}1)$$

where $\sigma_{\hat{FP}}$ is the standard deviation of the FP estimate, $\hat{FP}$, C_1 and C_2 are constants, and n represents the sample size. This equation would explain not only why the bootstrap estimated $\sigma_{\hat{FP}}$ changes linearly with $\hat{FP}$, but also why the slope increases with decreasing FP, as the FP value represents the proportion of fatal defects and is thus proportional to sample size n. An interesting question is what precisely constitutes n in our case. Since as FP decreases, $T\text{-}T_G$ decreases, n could be proportional to $T\text{-}T_G$.

Having established that the BCA method is the best method of CI estimation, perhaps the most important practical issue to be addressed is how do we ascertain that the BCA method is working in practice. In our simulations, of course, we know the true value of FP and thus can easily evaluate the reliability of the BCA method. But what method or criteria can we rely upon when we are given a sample from some unknown population with many defect types each with unknown FP? More specifically, how do we know if the sample size is large enough for the BCA-estimated CI to be reliable? In the literature it is suggested that the sample size should be at least 30 [27]. But in the case of FP estimation based on in-line defect data we do not know exactly what constitutes a sample size. Examination of the behavior of the distribution of the bootstrap estimates may be a good starting point to determine the adequacy of the bootstrap CI estimate.

Another topic to explore is what is the relationship between the observed slope of $\sigma_{\hat{FP}}$ vs. $\hat{FP}$ and the acceleration constant used in the BCA method. From our data we can observe that the ratio of the slope of $\sigma_{\hat{FP}}$ vs. $\hat{FP}$ from one run to another corresponds to the ratio of the acceleration constants from the same runs. This result seems reasonable, as the acceleration constant is a measure of the rate of change of the standard deviation with the parameter being estimated on a normalized scale.

Another result from the bootstrap analysis that should be further addressed is the approximately constant value of the slopes of $\sigma_{\hat{FP}}$ vs. $\hat{FP}$ within runs, even though the FP changes within each run. This result suggests that the n in Equation (5-1) may be more correlated with T-T_G, than T_A-T_{GA}. A related issue is the degree of covariance or correlation that exists between $\hat{FP}_A$ and $\hat{FP}_B$, $\hat{FP}_A$ and $\hat{FP}_C$, etc. Furthermore, how do estimates of their standard deviation or the confidence intervals change when the relative values of the FP_A and FP_B, or FP_A and FP_C, are changed?

BIBLIOGRAPHY

1. Peters, L. "Introduction to Integrated Yield Analysis", *Semiconductor International*, Jan. 1999.

2. Ferris-Prabhu, A.V., "Computation of the critical area in Semiconductor Yield Theory", *Proc. Electronic Automation Design Conf.* (EDA84), Mar. 1984, p.171.

3. Pineda de Gyvez, J., *Integrated Circuit Defect Sensitivity: Theory and Computational Models*, 1993, Kluwer Academic Publishers, Norwell, MA.

4. Kasten A., Zalnoski, J., and Mullenix, P., "Calculating Defect Limited Yields From In-Line Inspections", *Semiconductor International*, July 1997, p.202.

5. Stapper, C.H., "Integrated Circuit Yield Statistics", *Proc. IEEE*, vol.71, April 1983.

6. Andrieu, G., Caraux, G., and Gascuel, O., "Confidence intervals of evolutionary distances between sequences and comparison with usual approaches including the bootstrap method," *Mol. Biol. Evol.*, vol. 14, pp. 875-882, 1997.

7. Dopazo, J., "Estimating errors and confidence intervals for branch lengths in phylogenetic trees by a bootstrap approach," *J. Mol. Evol.,* vol. 38, pp. 300-304, 1994.

8. Manly, B.F.J., *Randomization and Monte Carlo Methods in Biology*, 2nd Ed., London, Chapman and Hall, 1997.

9. Kostelich, E.J., "Bootstrap estimates of chaotic dynamics," *Phys. Rev. E.,* vol. 64, pp. 162130/1-16213/10, 2001.

10. Seki, T., and Yokoyama, S., "Robust parameter estimation using the bootstrap method for the two- parameter Weibull distribution," *IEEE Trans. On Reliab.,* vol. 45, pp. 34-41, 1996.

11. Martz, H.F., "A comparison of three methods for calculating lower confidence limits on system reliability using binomial component data," *IEEE Trans. On Reliab.*, vol. 34, pp. 113-120, 1985.

12. Lunneborg, C.E., "Bootstrap applications for the behavioral sciences," *Psychometrika,* vol. 52, pp. 477-478, 1987.

13. Mooney, C.Z., and Duval, R.D., *Bootstrapping: A Nonparametric Approach to Statistical Inference. Quantitative Applications in the Social Sciences 95.* Newbury Park, CA, Sage Publications, 1993.

14. Ross, R. and Atchison, N., "The Calculation of Wafer Probe Yield Limits from In-Line Defect Monitor Data", *TI Technical Journal*, Oct. 1998.

15. Kasten A., Zalnoski, J., and Mullenix, P., "Limited Yield Estimation for Visual Defect Sources", *IEEE Transactions On Semiconductor Manufacturing*, vol.10, Feb. 1997.

16. Simulations done by Julie Segal of HPL Inc.

17. Ross, S.M, *Introduction to Probability Models*, Academic Press, 1993.

18. Stapper, C.H., "Modeling of Integrated Circuit Defect Sensitivities", *IBM Journal of Research and Development*, vol.27, November 1983.

19. Ferris-Prabhu, A.V., Introduction *to Semiconductor Device Yield Modeling*, 1992, Artech House, Inc. Norwood, MA, pp.43-44.

20. Kikuchi, H., Nishio, N., Ikeyama, K., and Misumi, A. "Advanced Defect Kill-Rate Estimation and Yield Analysis Incorporating Defect Clustering", *IEEE Conference Proceedings*, 1999.

21. Lunneborg, C.E., *Data Analysis by Resampling: Concepts and Applications*, Ch.7, Pacific Grove, CA, Duxbury, 2000.

22. Efron, B., *An Introduction to the Bootstrap,* Ch.13. Boca Raton, FL, Chapman and Hall /CRC, 1993.

23. Efron, B., "Better Bootstrap Confidence Intervals", *Journal of the American Statistical Association*, vol.82, pp.171-185, 1987.

24. Poag, F., Paradis, D., Reddy, M. and Button, J., "Implementing on-line ADC and an automated yield information management system", *Micro*, vol. 67, pp. 67 –76, 2000.

25. Riley, S.L., "Limitations to Estimating Yield Based on In-Line Defect Measurements", *International Symposium on Defect and Fault Tolerance in VLSI Systems*, 1999.

26. Segal, J., "A Framework for Extracting Defect Density Information for Yield Modeling from In-line Defect Inspection for Real-time Prediction of Random Defect Limited Yields", *IEEE International Symposium on Semiconductor Manufacturing Conference, Proceedings*, 1999.

27. Chernick, M.R., *Bootstrap Methods: A Practitioner's Guide*. Hoboken, NJ, Wiley-Interscience, 1999

APPENDIX

Simulation Code for Ch.3 written in Excel Visual Basic for Applications

```
Option Explicit

Sub GetCI2()

Dim arAProb() As Long
Dim arBProb() As Long
Dim arCProb() As Long
Dim arWafer() As Integer
Dim arSampFP() As Single
Dim arSampLY() As Single
Dim arRFP() As Single
Dim arStandCIA() As Single
Dim arStandCIB() As Single
Dim arStandCIC() As Single
Dim ar1stPrcntCIA() As Single
Dim ar1stPrcntCIB() As Single
Dim ar1stPrcntCIC() As Single
Dim ar2ndPrcntCIA() As Single
Dim ar2ndPrcntCIB() As Single
Dim ar2ndPrcntCIC() As Single
Dim arBCACIA() As Single
Dim arBCACIB() As Single
Dim arBCACIC() As Single
Dim FPA As Single
Dim FPB As Single
Dim FPC As Single
Dim BValueA As Single
Dim BValueB As Single
Dim BValueC As Single
Dim Aupper As Integer
Dim Alower As Integer
Dim Bupper As Integer
Dim Blower As Integer
Dim Cupper As Integer
Dim Clower As Integer
Dim intNoWafers As Integer
Dim intDieIDCnt As Integer
Dim Row As Integer
Dim TAdefA As Integer
Dim TAdefB As Integer
Dim TAdefC As Integer
Dim TGAdefA As Integer
Dim TGAdefB As Integer
Dim TGAdefC As Integer
Dim TG As Integer
Dim T As Integer
Dim NoCI As Integer
Dim intNoCICounter As Integer
Dim sngConfLevel As Single
Dim i As Integer
Dim k As Integer
Dim n As Integer
Dim m As Integer
Dim RowNumb  As Integer
Dim LastRow As Integer
Dim intRow As Integer
Dim intCol As Integer
Dim B As Integer

  'Get input values from WorkSheet 2
  With Worksheets("Sheet2")
    intNoWafers = .Cells(3, 2).Value        'Number of wafers per sample
    BValueA = .Cells(4, 2).Value            'These control the clustering;
    BValueB = .Cells(5, 2).Value            '0 corresponds to no clustering
    BValueC = .Cells(6, 2).Value
    Aupper = .Cells(7, 2).Value             'Number of defects per wafer upper
    Alower = .Cells(8, 2).Value             'and lower limits
    Bupper = .Cells(9, 2).Value
    Blower = .Cells(10, 2).Value
    Cupper = .Cells(11, 2).Value
    Clower = .Cells(12, 2).Value
    FPA = .Cells(13, 2).Value               'Assigned FP values
    FPB = .Cells(14, 2).Value
    FPC = .Cells(15, 2).Value
    NoCI = .Cells(16, 2).Value              'Number of CIs to estimate-
    sngConfLevel = .Cells(17, 2).Value        'corresponds to number of original samples
    B = .Cells(18, 2).Value                'Number of bootstrap samples per original
  End With                                'sample

  ReDim arSampFP(1 To NoCI, 1 To 3)         'This array holds all the Sample FP estimates
  ReDim arSampLY(1 To NoCI, 1 To 3)          'This array holds all the Sample LY estimates

  ReDim arStandCIA(1 To NoCI, 1 To 7)       'Holds conf. limis etc. for Standard method
  ReDim arStandCIB(1 To NoCI, 1 To 7)
  ReDim arStandCIC(1 To NoCI, 1 To 7)

  ReDim ar1stPrcntCIA(1 To NoCI, 1 To 2)     'Holds conf. limts for 1st percentile method
  ReDim ar1stPrcntCIB(1 To NoCI, 1 To 2)
  ReDim ar1stPrcntCIC(1 To NoCI, 1 To 2)

  ReDim ar2ndPrcntCIA(1 To NoCI, 1 To 2)     'Holds conf. limits for 2nd percentile method
  ReDim ar2ndPrcntCIB(1 To NoCI, 1 To 2)
  ReDim ar2ndPrcntCIC(1 To NoCI, 1 To 2)

  ReDim arBCACIA(1 To NoCI, 1 To 6)         'Holds conf. limits etc. for BCA method
  ReDim arBCACIB(1 To NoCI, 1 To 6)
  ReDim arBCACIC(1 To NoCI, 1 To 6)

  'Clear previous results
  For intRow = 3 To 3002
```

```
    For intCol = 3 To 36
        Worksheets("Sheet2").Cells(intRow, intCol).Clear
    Next intCol
Next intRow
For intRow = 3 To 1002
    For intCol = 2 To 256
        Worksheets("RFP").Cells(intRow, intCol).Clear
    Next intCol
Next intRow

'Start Loop Here
'^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
For intNoCICounter = 1 To NoCI
'Create Wafer Array showing all dies, number of defects and faults on each
'one of them, and each of their bin numbers

ReDim arWafer(1 To intNoWafers * 100, 1 To 9)   'Holds wafers array

'This subprocedure will create the wafers based on the input paramters
GetWaferArray Aupper, Alower, Bupper, Blower, Cupper, Clower, _
    BValueA, BValueB, BValueC, intNoWafers, FPA, FPB, FPC, arWaferHolder:=arWafer

'Display last wafer created on spreadsheet "Wafer"
If intNoCICounter = NoCI Then
    LastRow = UBound(arWafer, 1)
    With Worksheets("Wafer")
        For RowNumb = 1 To LastRow
            .Cells(RowNumb + 1, 2).Value = arWafer(RowNumb, 1)
            .Cells(RowNumb + 1, 3).Value = arWafer(RowNumb, 2)
            .Cells(RowNumb + 1, 4).Value = arWafer(RowNumb, 3)
            .Cells(RowNumb + 1, 5).Value = arWafer(RowNumb, 4)
            .Cells(RowNumb + 1, 6).Value = arWafer(RowNumb, 5)
            .Cells(RowNumb + 1, 7).Value = arWafer(RowNumb, 6)
            .Cells(RowNumb + 1, 8).Value = arWafer(RowNumb, 7)
            .Cells(RowNumb + 1, 9).Value = arWafer(RowNumb, 8)
            .Cells(RowNumb + 1, 10).Value = arWafer(RowNumb, 9)
        Next RowNumb
    End With
End If

'Get TA for each defect type
TAdefA = FuncTA("A", arWaferHolder:=arWafer)
TAdefB = FuncTA("B", arWaferHolder:=arWafer)
TAdefC = FuncTA("C", arWaferHolder:=arWafer)

'Get TGA for each defect type
TGAdefA = FuncTGA("A", arWaferHolder:=arWafer)
TGAdefB = FuncTGA("B", arWaferHolder:=arWafer)
TGAdefC = FuncTGA("C", arWaferHolder:=arWafer)

'Get TG and T
TG = FuncTG(arWaferHolder:=arWafer)
T = UBound(arWafer, 1)
```

```
arSampFP(intNoCICounter, 1) = FuncFP(T, TG, TAdefA, TGAdefA, "A", _
                        arWaferHolder:=arWafer)
arSampFP(intNoCICounter, 2) = FuncFP(T, TG, TAdefB, TGAdefB, "B", _
                        arWaferHolder:=arWafer)
arSampFP(intNoCICounter, 3) = FuncFP(T, TG, TAdefC, TGAdefC, "C", _
                        arWaferHolder:=arWafer)
arSampLY(intNoCICounter, 1) = FuncLY(T, TG, TAdefA, TGAdefA)
arSampLY(intNoCICounter, 2) = FuncLY(T, TG, TAdefB, TGAdefB)
arSampLY(intNoCICounter, 3) = FuncLY(T, TG, TAdefC, TGAdefC)

'Create Array of Bootstrap FP Values from arWafer
CreateRFPArray B, arSampFP(intNoCICounter, 1), arSampFP(intNoCICounter, 2), _
   arSampFP(intNoCICounter, 3), False, arWaferHolder:=arWafer, arRFPHolder:=arRFP

'Display bootstrap FP estimates for last 80 original samples (CI's)
If intNoCICounter > NoCI - 80 Then
   DisplayRFP 80, NoCI, intNoCICounter, arRFPHolder:=arRFP
End If

'Display Biases, Variances, and Replicate FP Values for last C.I.
If intNoCICounter = NoCI Then
   DisplayBR B, arSampFP(intNoCICounter, 1), arSampFP(intNoCICounter, 2), _
   arSampFP(intNoCICounter, 3), arRFPHolder:=arRFP

   DisplayVariance B, arRFPHolder:=arRFP
End If

'Get Standard C.I.'s and store into arStandCIA, arStandCIB, and arStandCIC
GetStandCI "A", sngConfLevel, arSampFP(intNoCICounter, 1), intNoCICounter, _
   arRFPHolder:=arRFP, arStandCIHolder:=arStandCIA
GetStandCI "B", sngConfLevel, arSampFP(intNoCICounter, 2), intNoCICounter, _
   arRFPHolder:=arRFP, arStandCIHolder:=arStandCIB
GetStandCI "C", sngConfLevel, arSampFP(intNoCICounter, 3), intNoCICounter, _
   arRFPHolder:=arRFP, arStandCIHolder:=arStandCIC

'Get 1stPercentile C.I.'s and store int ar1stPercentCIA, ar1stPercentCIB,
'ar1stPercentCIC
Get1stPrcntCI "A", sngConfLevel, intNoCICounter, arRFPHolder:=arRFP, _
   arCIHolder:=ar1stPrcntCIA
Get1stPrcntCI "B", sngConfLevel, intNoCICounter, arRFPHolder:=arRFP, _
   arCIHolder:=ar1stPrcntCIB
Get1stPrcntCI "C", sngConfLevel, intNoCICounter, arRFPHolder:=arRFP, _
   arCIHolder:=ar1stPrcntCIC

Get2ndPrcntCI arSampFP(intNoCICounter, 1), ar1stPrcntCIA(intNoCICounter, 2), _
   ar1stPrcntCIA(intNoCICounter, 1), intNoCICounter, arCIHolder:=ar2ndPrcntCIA

Get2ndPrcntCI arSampFP(intNoCICounter, 2), ar1stPrcntCIB(intNoCICounter, 2), _
   ar1stPrcntCIB(intNoCICounter, 1), intNoCICounter, arCIHolder:=ar2ndPrcntCIB

Get2ndPrcntCI arSampFP(intNoCICounter, 3), ar1stPrcntCIC(intNoCICounter, 2), _
   ar1stPrcntCIC(intNoCICounter, 1), intNoCICounter, arCIHolder:=ar2ndPrcntCIC
```

```
GetBCACI sngConfLevel, arSampFP(intNoCICounter, 1), arSampFP(intNoCICounter, 2), _
   arSampFP(intNoCICounter, 3), intNoCICounter, arRFPHolder:=arRFP, _
   arBCACIAHolder:=arBCACIA, arBCACIBHolder:=arBCACIB, _
   arBCACICHolder:=arBCACIC, arWaferHolder:=arWafer

Next intNoCICounter
'Loop Ends Here
'^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


For i = 1 To NoCI
   With Worksheets("Sheet2")
      .Cells(i * 3, 3).Value = "A"
      .Cells(i * 3 + 1, 3).Value = "B"
      .Cells(i * 3 + 2, 3).Value = "C"
      .Cells(i * 3, 4).Value = arSampFP(i, 1)
      .Cells(i * 3, 23).Value = arSampLY(i, 1)
      .Cells(i * 3, 5).Value = ar1stPrcntCIA(i, 1)
      .Cells(i * 3, 6).Value = ar1stPrcntCIA(i, 2)
      .Cells(i * 3 + 1, 4).Value = arSampFP(i, 2)
      .Cells(i * 3 + 1, 23).Value = arSampLY(i, 2)
      .Cells(i * 3 + 1, 5).Value = ar1stPrcntCIB(i, 1)
      .Cells(i * 3 + 1, 6).Value = ar1stPrcntCIB(i, 2)
      .Cells(i * 3 + 2, 4).Value = arSampFP(i, 3)
      .Cells(i * 3 + 2, 23).Value = arSampLY(i, 3)
      .Cells(i * 3 + 2, 5).Value = ar1stPrcntCIC(i, 1)
      .Cells(i * 3 + 2, 6).Value = ar1stPrcntCIC(i, 2)
      If ar1stPrcntCIC(i, 1) > FPC Or _
         ar1stPrcntCIC(i, 2) < FPC Then
         .Cells(i * 3 + 2, 7).Value = "Failed"
      End If
      If ar1stPrcntCIB(i, 1) > FPB Or _
         ar1stPrcntCIB(i, 2) < FPB Then
         .Cells(i * 3 + 1, 7).Value = "Failed"
      End If
      If ar1stPrcntCIA(i, 1) > FPA Or _
         ar1stPrcntCIA(i, 2) < FPA Then
         .Cells(i * 3, 7).Value = "Failed"
      End If
      .Cells(i * 3, 8).Value = ar2ndPrcntCIA(i, 1)
      .Cells(i * 3, 9).Value = ar2ndPrcntCIA(i, 2)
      .Cells(i * 3 + 1, 8).Value = ar2ndPrcntCIB(i, 1)
      .Cells(i * 3 + 1, 9).Value = ar2ndPrcntCIB(i, 2)
      .Cells(i * 3 + 2, 8).Value = ar2ndPrcntCIC(i, 1)
      .Cells(i * 3 + 2, 9).Value = ar2ndPrcntCIC(i, 2)
      If ar2ndPrcntCIC(i, 1) > FPC Or _
         ar2ndPrcntCIC(i, 2) < FPC Then
         .Cells(i * 3 + 2, 10).Value = "Failed"
      End If
      If ar2ndPrcntCIB(i, 1) > FPB Or _
         ar2ndPrcntCIB(i, 2) < FPB Then
         .Cells(i * 3 + 1, 10).Value = "Failed"
```

```
End If
If ar2ndPrcntCIA(i, 1) > FPA Or _
   ar2ndPrcntCIA(i, 2) < FPA Then
   .Cells(i * 3, 10).Value = "Failed"
End If

If ar2ndPrcntCIC(i, 1) > FPC Or _
   ar2ndPrcntCIC(i, 2) < FPC Then
   .Cells(i * 3 + 2, 10).Value = "Failed"
End If
If ar2ndPrcntCIB(i, 1) > FPB Or _
   ar2ndPrcntCIB(i, 2) < FPB Then
   .Cells(i * 3 + 1, 10).Value = "Failed"
End If
If ar2ndPrcntCIA(i, 1) > FPA Or _
   ar2ndPrcntCIA(i, 2) < FPA Then
   .Cells(i * 3, 10).Value = "Failed"
End If
For k = 1 To 7
   .Cells(i * 3 + 2, 10 + k).Value = arStandCIC(i, k)
   .Cells(i * 3 + 1, 10 + k).Value = arStandCIB(i, k)
   .Cells(i * 3, 10 + k).Value = arStandCIA(i, k)
Next k
If arStandCIC(i, 6) > FPC Or _
   arStandCIC(i, 7) < FPC Then
   .Cells(i * 3 + 2, 18) = "Failed"
End If
If arStandCIB(i, 6) > FPB Or _
   arStandCIB(i, 7) < FPB Then
   .Cells(i * 3 + 1, 18) = "Failed"
End If
If arStandCIA(i, 6) > FPA Or _
   arStandCIA(i, 7) < FPA Then
   .Cells(i * 3, 18) = "Failed"
End If


For k = 1 To 6
   .Cells(i * 3 + 2, 24 + k).Value = arBCACIC(i, k)
   .Cells(i * 3 + 1, 24 + k).Value = arBCACIB(i, k)
   .Cells(i * 3, 24 + k).Value = arBCACIA(i, k)
Next k
If arBCACIC(i, 5) > FPC Or _
   arBCACIC(i, 6) < FPC Then
   .Cells(i * 3 + 2, 31) = "Failed"
End If
If arBCACIB(i, 5) > FPB Or _
   arBCACIB(i, 6) < FPB Then
   .Cells(i * 3 + 1, 31) = "Failed"
End If
If arBCACIA(i, 5) > FPA Or _
   arBCACIA(i, 6) < FPA Then
   .Cells(i * 3, 31) = "Failed"
```

```
            End If

          End With
        Next i


        MsgBox ("OkeyDokey")

    End Sub


    Function FuncNoPoints(ByRef arProb() As Long) As Integer
    Dim upperbound As Long
    Dim RandNumber As Long
    Dim n As Integer

        upperbound = arProb(UBound(arProb, 1), 4)
        Randomize Timer
        RandNumber = Int((upperbound - 1 + 1) * Rnd + 1)

        For n = 1 To UBound(arProb, 1)
          If RandNumber >= arProb(n, 3) And RandNumber <= arProb(n, 4) Then
            FuncNoPoints = arProb(n, 1)
            Exit Function
          End If
        Next n

    End Function
    Function FuncNoBadPoints(ByVal NoPoints As Integer, ByVal FP As Single) As Integer
    Dim intPointsCnt As Integer
    Dim RandNumber As Integer
    Dim n As Integer

        For intPointsCnt = 1 To NoPoints
          Randomize Timer
          RandNumber = Int((10000 - 1 + 1) * Rnd + 1)
          If RandNumber <= CInt(FP * 10000) Then
            n = n + 1
          End If
        Next intPointsCnt

        FuncNoBadPoints = n

    End Function

    Function FuncProb(n As Integer, DD As Double, alpha As Double) As Double
    Dim upperGuy As Double
    Dim lowerGuy As Double

        upperGuy = (Exp(Excel.WorksheetFunction.GammaLn(alpha + n))) _
          * (DD / alpha) ^ n
        lowerGuy = Excel.WorksheetFunction.Fact(n) * _
          (Exp(Excel.WorksheetFunction.GammaLn(alpha))) _
    * (1 + DD / alpha) ^ (n + alpha)
  FuncProb = upperGuy / lowerGuy

End Function

Sub CreateProbArray(ByVal DD As Double, ByVal alpha As Double, _
  ByRef arFinalProb() As Long)
Dim n As Integer
Dim Row As Integer
Dim arProb(1 To 100, 1 To 4)

  n = 0
  Do
    arProb(n + 1, 1) = n
    arProb(n + 1, 2) = CLng(10000000 * FuncProb(n, DD, alpha))
    If n = 0 Then
      arProb(n + 1, 3) = 1
      arProb(n + 1, 4) = arProb(n + 1, 2)
    Else
      arProb(n + 1, 3) = arProb(n, 4) + 1
      arProb(n + 1, 4) = arProb(n, 4) + arProb(n + 1, 2)
    End If

    n = n + 1
  Loop While (CLng(1000000 * FuncProb(n, DD, alpha))) > 0

  ReDim arFinalProb(1 To n, 1 To 4)
  For Row = 1 To n
    arFinalProb(Row, 1) = arProb(Row, 1)
    arFinalProb(Row, 2) = arProb(Row, 2)
    arFinalProb(Row, 3) = arProb(Row, 3)
    arFinalProb(Row, 4) = arProb(Row, 4)
  Next Row

End Sub
Function FuncBinNumber(ByVal ANoBadPoints As Integer, ByVal BNoBadPoints As Integer, _
  ByVal CNoBadPoints As Integer) As Integer

  If ANoBadPoints > 0 Or BNoBadPoints > 0 Or CNoBadPoints > 0 Then
    FuncBinNumber = 8
  Else
    FuncBinNumber = 1
  End If

End Function

Function FuncTA(ByVal strDefType As String, ByRef arWaferHolder() As Integer) _
  As Integer
Dim intCol As Integer
Dim i As Integer
Dim TA As Integer

  If strDefType = "A" Then
```

```
        intCol = 3
    ElseIf strDefType = "B" Then
        intCol = 5
    Else
        intCol = 7
    End If

    For i = 1 To UBound(arWaferHolder, 1)
        If arWaferHolder(i, intCol) > 0 Then
            TA = TA + 1
        End If
    Next i

    FuncTA = TA
End Function

Function FuncTGA(ByVal strDefType As String, ByRef arWaferHolder() As Integer) _
    As Integer
Dim intCol As Integer
Dim i As Integer
Dim TGA As Integer

    If strDefType = "A" Then
        intCol = 3
    ElseIf strDefType = "B" Then
        intCol = 5
    Else
        intCol = 7
    End If

    For i = 1 To UBound(arWaferHolder, 1)
        If arWaferHolder(i, intCol) > 0 And arWaferHolder(i, 9) = 1 Then
            TGA = TGA + 1
        End If
    Next i

    FuncTGA = TGA

End Function


Function FuncTG(ByRef arWaferHolder() As Integer) _
    As Integer
Dim i As Integer
Dim TG As Integer

    For i = 1 To UBound(arWaferHolder, 1)
        If arWaferHolder(i, 9) = 1 Then
            TG = TG + 1
        End If
    Next i

    FuncTG = TG
```

```
End Function
Function FuncFP(ByVal T As Double, ByVal TG As Double, ByVal TA As Double, _
   ByVal TGA As Double, ByVal strDefType As String, _
   ByRef arWaferHolder() As Integer) As Double
Dim intCol As Integer
Dim i As Integer
Dim TotDefs As Integer
Dim TotDie As Integer

   If strDefType = "1" Then
      intCol = 1
   ElseIf strDefType = "A" Then
      intCol = 3
   ElseIf strDefType = "B" Then
      intCol = 5
   Else
      intCol = 7
   End If

   TotDie = T

   For i = 1 To TotDie
      TotDefs = arWaferHolder(i, intCol) + TotDefs
   Next i

   FuncFP = -Log(TG * (T - TA) / (T * (TG - TGA))) / (TotDefs / TotDie)

End Function
Function FuncLY(ByVal T As Double, ByVal TG As Double, ByVal TA As Double, _
   ByVal TGA As Double) As Double

   FuncLY = TG * (T - TA) / (T * (TG - TGA))
   If FuncLY > 1 Then
      FuncLY = 1
   End If

End Function
Sub CreateRFPArray(ByVal intNoReplicates As Integer, ByVal SampFPA As Single, _
   ByVal SampFPB As Single, ByVal SampFPC As Single, boolZ As Boolean, _
   arWaferHolder() As Integer, ByRef arRFPHolder() As Single)
Dim arWaferbootstrap() As Integer
Dim upperbound As Integer
Dim i As Integer
Dim T As Integer
Dim TG As Integer
Dim TAdefA As Integer
Dim TAdefB As Integer
Dim TAdefC As Integer
Dim TGAdefA As Integer
Dim TGAdefB As Integer
Dim TGAdefC As Integer
Dim intNoReplicatesCnt As Integer
Dim RandNumber As Integer

  If boolZ = True Then
    ReDim arRFPHolder(1 To intNoReplicates, 1 To 6)
  Else
    ReDim arRFPHolder(1 To intNoReplicates, 1 To 3)
  End If

  upperbound = UBound(arWaferHolder, 1)
  T = upperbound

  For intNoReplicatesCnt = 1 To intNoReplicates
  ReDim arWaferbootstrap(1 To upperbound, 1 To 9)

  Randomize Timer
  For i = 1 To upperbound
    'Randomly pick a die number between 1 and Number of Total Die
    RandNumber = Int((upperbound - 1 + 1) * Rnd + 1)
    arWaferbootstrap(i, 1) = arWaferHolder(RandNumber, 1)
    arWaferbootstrap(i, 2) = arWaferHolder(RandNumber, 2)
    arWaferbootstrap(i, 3) = arWaferHolder(RandNumber, 3)
    arWaferbootstrap(i, 4) = arWaferHolder(RandNumber, 4)
    arWaferbootstrap(i, 5) = arWaferHolder(RandNumber, 5)
    arWaferbootstrap(i, 6) = arWaferHolder(RandNumber, 6)
    arWaferbootstrap(i, 7) = arWaferHolder(RandNumber, 7)
    arWaferbootstrap(i, 8) = arWaferHolder(RandNumber, 8)
    arWaferbootstrap(i, 9) = arWaferHolder(RandNumber, 9)
  Next i

  TAdefA = FuncTA("A", arWaferHolder:=arWaferbootstrap)
  TAdefB = FuncTA("B", arWaferHolder:=arWaferbootstrap)
  TAdefC = FuncTA("C", arWaferHolder:=arWaferbootstrap)

  TGAdefA = FuncTGA("A", arWaferHolder:=arWaferbootstrap)
  TGAdefB = FuncTGA("B", arWaferHolder:=arWaferbootstrap)
  TGAdefC = FuncTGA("C", arWaferHolder:=arWaferbootstrap)

  TG = FuncTG(arWaferHolder:=arWaferbootstrap)


  arRFPHolder(intNoReplicatesCnt, 1) = FuncFP(T, TG, TAdefA, TGAdefA, "A", _
                    arWaferHolder:=arWaferbootstrap)
  arRFPHolder(intNoReplicatesCnt, 2) = FuncFP(T, TG, TAdefB, TGAdefB, "B", _
                    arWaferHolder:=arWaferbootstrap)
  arRFPHolder(intNoReplicatesCnt, 3) = FuncFP(T, TG, TAdefC, TGAdefC, "C", _
                    arWaferHolder:=arWaferbootstrap)
  'If arRFPHolder(intNoReplicatesCnt, 1) = 0 Or arRFPHolder(intNoReplicatesCnt, 2) = 0 _
    Or arRFPHolder(intNoReplicatesCnt, 3) = 0 Then
    'intNoReplicatesCnt = intNoReplicatesCnt - 1
  'End If

  If boolZ = True Then
    arRFPHolder(intNoReplicatesCnt, 4) = ZFunc("A", SampFPA, arRFPHolder _
        (intNoReplicatesCnt, 1), arWaferbootstrapHolder:=arWaferbootstrap)
      arRFPHolder(intNoReplicatesCnt, 5) = ZFunc("B", SampFPB, arRFPHolder _
        (intNoReplicatesCnt, 2), arWaferbootstrapHolder:=arWaferbootstrap)
      arRFPHolder(intNoReplicatesCnt, 6) = ZFunc("C", SampFPC, arRFPHolder _
        (intNoReplicatesCnt, 3), arWaferbootstrapHolder:=arWaferbootstrap)
    End If

    Next intNoReplicatesCnt

  End Sub

  Sub GetStandCI(ByVal strDefType As String, ByVal sngConfLevel As Single, _
    ByVal SampFP As Single, ByVal intCINoCnter As Integer, _
    ByRef arRFPHolder() As Single, ByRef arStandCIHolder)
  Dim SampFPrnd As Double
  Dim SampFPfaults As Double
  Dim AvgBootFP As Single
  Dim StDevFP As Single
  Dim alpha As Single

    alpha = 1 - sngConfLevel
    AvgBootFP = FuncAvgBootFP(strDefType, arRFPHolder:=arRFPHolder)
    StDevFP = FuncStDevBootFP(strDefType, AvgBootFP, arRFPHolder:=arRFPHolder)

    arStandCIHolder(intCINoCnter, 1) = SampFP
    arStandCIHolder(intCINoCnter, 2) = AvgBootFP
    arStandCIHolder(intCINoCnter, 3) = StDevFP
    arStandCIHolder(intCINoCnter, 4) = AvgBootFP - SampFP
    arStandCIHolder(intCINoCnter, 5) = (sngConfLevel) * 100
    arStandCIHolder(intCINoCnter, 6) = SampFP - (AvgBootFP - SampFP) - _
      StDevFP * (Excel.WorksheetFunction.NormSInv(1 - alpha / 2))
    arStandCIHolder(intCINoCnter, 7) = SampFP - (AvgBootFP - SampFP) - _
      StDevFP * (Excel.WorksheetFunction.NormSInv(alpha / 2))


  End Sub
  Function FuncStDevBootFP(ByVal strDefType, ByVal AvgBootFP As Single, _
    arRFPHolder() As Single) As Double
  Dim Sum As Double
  Dim intCol As Integer
  Dim i As Integer
  Dim upperbound As Integer

    If strDefType = "A" Or strDefType = "1" Then
      intCol = 1
    ElseIf strDefType = "B" Then
      intCol = 2
    Else
      intCol = 3
    End If

    upperbound = UBound(arRFPHolder, 1)
    For i = 1 To upperbound
        Sum = (arRFPHolder(i, intCol) - AvgBootFP) ^ 2 + Sum
    Next i

    FuncStDevBootFP = (Sum / (upperbound - 1)) ^ (0.5)

End Function

Function FuncAvgBootFP(ByVal strDefType, ByRef arRFPHolder() As Single) As Double
Dim Total As Double
Dim Sum As Double
Dim intCol As Integer
Dim i As Integer
Dim upperbound As Integer

    If strDefType = "A" Or strDefType = "1" Then
        intCol = 1
    ElseIf strDefType = "B" Then
        intCol = 2
    Else
        intCol = 3
    End If

    upperbound = UBound(arRFPHolder, 1)

    For i = 1 To upperbound
        Total = arRFPHolder(i, intCol) + Total
    Next i

    FuncAvgBootFP = Total / upperbound

End Function

Sub CreateSortedArray(ByVal strDefType As String, ByRef arRFPHolder() As Single)
Dim i As Integer
Dim j As Integer
Dim tmp As Single
Dim intCol As Integer

    If strDefType = "A" Or strDefType = "1" Then
        intCol = 1
    ElseIf strDefType = "B" Then
        intCol = 2
    Else
        intCol = 3
    End If

    For i = LBound(arRFPHolder, 1) To UBound(arRFPHolder, 1) - 1
        For j = (i + 1) To UBound(arRFPHolder, 1)
            If arRFPHolder(i, intCol) > arRFPHolder(j, intCol) Then
                tmp = arRFPHolder(i, intCol)
                arRFPHolder(i, intCol) = arRFPHolder(j, intCol)
                arRFPHolder(j, intCol) = tmp
            End If
        Next j
    Next i

End Sub
Sub CreateSortedTArray(ByVal strDefType As String, ByRef arRFPHolder() As Single)
Dim i As Integer
Dim j As Integer
Dim tmp As Single
Dim intCol As Integer

    If strDefType = "A" Or strDefType = "1" Then
        intCol = 4
    ElseIf strDefType = "B" Then
        intCol = 5
    Else
        intCol = 6
    End If

    For i = LBound(arRFPHolder, 1) To UBound(arRFPHolder, 1) - 1
        For j = (i + 1) To UBound(arRFPHolder, 1)
            If arRFPHolder(i, intCol) > arRFPHolder(j, intCol) Then
                tmp = arRFPHolder(i, intCol)
                arRFPHolder(i, intCol) = arRFPHolder(j, intCol)
                arRFPHolder(j, intCol) = tmp
            End If
        Next j
    Next i

End Sub
Sub Get1stPrcntCI(ByVal strDefType As String, ByVal sngConfLevel As Single, _
    ByVal intCINoCnter As Integer, ByRef arRFPHolder() As Single, _
    ByRef arCIHolder() As Single)
Dim intLRow As Integer
Dim intURow As Integer
Dim intCol As Integer
Dim bootFPlow As Single
Dim bootFPhigh As Single
Dim alpha As Single

    If strDefType = "A" Then
        intCol = 1
    ElseIf strDefType = "B" Then
        intCol = 2
    Else
        intCol = 3
    End If

    CreateSortedArray strDefType, arRFPHolder:=arRFPHolder

    alpha = 1 - sngConfLevel

    intLRow = Int(UBound(arRFPHolder, 1) * alpha / 2) + 1
    intURow = Int(UBound(arRFPHolder, 1) * (1 - alpha / 2)) + 1

    bootFPlow = arRFPHolder(intLRow, intCol)
    bootFPhigh = arRFPHolder(intURow, intCol)

    arCIHolder(intCINoCnter, 1) = bootFPlow
    arCIHolder(intCINoCnter, 2) = bootFPhigh

End Sub

Sub Get2ndPrcntCI(ByVal SampFP As Single, ByVal FP1stPrcnthigh As Single, _
    ByVal FP1stPrcntlow As Single, intCINoCnter As Integer, ByRef arCIHolder() As _
    Single)
Dim bootFPlow As Single
Dim bootFPhigh As Single


    bootFPlow = 2 * SampFP - FP1stPrcnthigh
    bootFPhigh = 2 * SampFP - FP1stPrcntlow

    arCIHolder(intCINoCnter, 1) = bootFPlow
    arCIHolder(intCINoCnter, 2) = bootFPhigh

End Sub

Sub GetWaferArray(Aupper As Integer, Alower As Integer, Bupper As Integer, Blower As _
    Integer, Cupper As Integer, Clower As Integer, BValueA As Single, BValueB As _
    Single, BValueC As Single, lngWaferNo As Integer, FPA As Single, _
    FPB As Single, FPC As Single, arWaferHolder() As Integer)
Dim A(1 To 52, 1 To 52) As Integer
Dim B(1 To 52, 1 To 52) As Integer
Dim C(1 To 52, 1 To 52) As Integer
Dim ANoOfPoints As Integer
Dim BNoOfPoints As Integer
Dim CNoOfPoints As Integer
Dim NoA As Integer
Dim NoB As Integer
Dim NoC As Integer
Dim Row As Integer
Dim Col As Integer
Dim i As Integer
Dim j As Integer
Dim k As Integer
Dim m As Integer
Dim n As Integer
Dim p As Integer
Dim lngWaferCounter As Long
Dim DiePointsA(1 To 10, 1 To 10) As Integer
Dim DiePointsB(1 To 10, 1 To 10) As Integer
Dim DiePointsC(1 To 10, 1 To 10) As Integer
Dim intDieIDCnt As Integer

    'Initialize Total Points Tracker
```

```
  NoA = 0
  NoB = 0
  NoC = 0

For lngWaferCounter = 1 To lngWaferNo

  'Initialize Arrays to 0
  For i = 1 To UBound(A, 1)
    For j = 1 To UBound(A, 2)
      A(i, j) = 0
      B(i, j) = 0
      C(i, j) = 0
    Next j
  Next i

  'Assign Random Number of Points for each Defect Type
  ANoOfPoints = RandNum(Aupper, Alower)
  BNoOfPoints = RandNum(Bupper, Blower)
  CNoOfPoints = RandNum(Cupper, Clower)

  'Generate Negative Binomial Arrays
  Randomize Timer
  RandomizeArray ArrayHolder2:=A, NoOfPoints:= _
    ANoOfPoints, B:=BValueA
  RandomizeArray ArrayHolder2:=B, NoOfPoints:= _
    BNoOfPoints, B:=BValueB
  RandomizeArray ArrayHolder2:=C, NoOfPoints:= _
    CNoOfPoints, B:=BValueC

  Row = 0
  Col = 0
  For i = 1 To 10
    For j = 1 To 10
      DiePointsA(i, j) = 0
      DiePointsB(i, j) = 0
      DiePointsC(i, j) = 0
    Next j
  Next i

  For i = 2 To 51 Step 5
    Row = Row + 1
    Col = 0
    For j = 2 To 51 Step 5
      Col = Col + 1
      For k = i To i + 4
        For m = j To j + 4
          DiePointsA(Row, Col) = A(k, m) + DiePointsA(Row, Col)
          DiePointsB(Row, Col) = B(k, m) + DiePointsB(Row, Col)
          DiePointsC(Row, Col) = C(k, m) + DiePointsC(Row, Col)
        Next m
      Next k
    Next j
  Next i
```

```
        n = 0
        p = 0
        For intDieIDCnt = (lngWaferCounter - 1) * 100 + 1 To (lngWaferCounter - 1) * 100 + 100
          n = n + 1
          If p < 10 Then
            p = p + 1
          Else
            p = 1
          End If

          arWaferHolder(intDieIDCnt, 1) = Int((intDieIDCnt / 100 - 0.00001) + 1)
          arWaferHolder(intDieIDCnt, 2) = intDieIDCnt
          'Defect Type A
          arWaferHolder(intDieIDCnt, 3) = DiePointsA(p, Int(n / 10 - 0.0001) + 1)
          arWaferHolder(intDieIDCnt, 4) = FuncNoBadPoints(arWaferHolder(intDieIDCnt, 3), FPA)
          'Defect Type B
          arWaferHolder(intDieIDCnt, 5) = DiePointsB(p, Int(n / 10 - 0.0001) + 1)
          arWaferHolder(intDieIDCnt, 6) = FuncNoBadPoints(arWaferHolder(intDieIDCnt, 5), FPB)
          'Defect Type C
          arWaferHolder(intDieIDCnt, 7) = DiePointsC(p, Int(n / 10 - 0.0001) + 1)
          arWaferHolder(intDieIDCnt, 8) = FuncNoBadPoints(arWaferHolder(intDieIDCnt, 7), FPC)

          arWaferHolder(intDieIDCnt, 9) = FuncBinNumber(arWaferHolder(intDieIDCnt, 4), _
                        arWaferHolder(intDieIDCnt, 6), arWaferHolder(intDieIDCnt, 8))
        Next intDieIDCnt

      Next lngWaferCounter

    End Sub
    '****************************************************************************
    Function Test1(ByVal Aij As Integer, ByVal Tot As Integer, _
      ByVal B As Single, i As Integer, _
      j As Integer, intElements As Integer, ArrayHolder() As Integer) As _
      Integer
    Const constA As Single = 0.5
    Dim sngC As Single

      sngC = 1 / intElements

      Aij = Aij + constA * (ArrayHolder(i - 1, j) + _
        ArrayHolder(i + 1, j) + ArrayHolder(i, j - 1) + _
        ArrayHolder(i, j + 1))

      If (Aij * B + sngC) / (Tot * B + sngC * intElements) _
        > Rnd Then
        Test1 = 1
      Else
        Test1 = 0
      End If

    End Function
    Function ArrayTotalPoints(ArrayHolder() As Integer) As _
    Integer
Dim i As Integer
Dim j As Integer
    ArrayTotalPoints = 0
    For i = 1 To UBound(ArrayHolder, 1)
        For j = 1 To UBound(ArrayHolder, 2)
            ArrayTotalPoints = ArrayHolder(i, j) + _
                ArrayTotalPoints
        Next j
    Next i
End Function
Sub RandomizeArray(ArrayHolder2() As Integer, ByVal NoOfPoints _
    As Long, ByVal B As Single)
Dim T As Integer
Dim i As Integer
Dim j As Integer
Dim intElements As Integer

    intElements = UBound(ArrayHolder2, 2) * UBound(ArrayHolder2, 1)
    Do While ArrayTotalPoints(ArrayHolder:=ArrayHolder2) _
        < NoOfPoints
        T = ArrayTotalPoints(ArrayHolder:=ArrayHolder2)
        For i = 2 To UBound(ArrayHolder2, 1) - 1
            For j = 2 To UBound(ArrayHolder2, 2) - 1
                    ArrayHolder2(i, j) = _
                    ArrayHolder2(i, j) + _
                    Test1(ArrayHolder2(i, j), T, B, _
                        i, j, intElements, ArrayHolder2())
            Next j
        Next i
    Loop

End Sub
Function RandNum(ByVal upperbound As Integer, ByVal _
    lowerbound As Integer) As Integer

    If upperbound = 0 And lowerbound = 0 Then
        RandNum = 0
    Else
        RandNum = Int((upperbound - lowerbound + 1) _
            * Rnd + lowerbound)
    End If

End Function

Function ZFunc(strDefType As String, SampFP As Single, BootFP As Single, _
    arWaferbootstrapHolder() As Integer) As Double
Dim arRFPHolder(1 To 25, 1 To 1) As Single
Dim intNoReplicatesCnt As Integer
Dim upperbound As Integer
Dim RandNumber As Integer
Dim arWaferbootstrap() As Integer
Dim i As Integer
Dim T As Integer
Dim TG As Integer
Dim TA As Integer
Dim TGA As Integer
Dim avg As Double
Dim StDev As Double

  upperbound = UBound(arWaferbootstrapHolder, 1)

  For intNoReplicatesCnt = 1 To 25    'Number of Replicates
  ReDim arWaferbootstrap(1 To upperbound, 1 To 9)

  Randomize Timer
  For i = 1 To upperbound
    'Randomly pick a die number between 1 and Number of Total Die
    RandNumber = Int((upperbound - 1 + 1) * Rnd + 1)
    arWaferbootstrap(i, 1) = arWaferbootstrapHolder(RandNumber, 1)
    arWaferbootstrap(i, 2) = arWaferbootstrapHolder(RandNumber, 2)
    arWaferbootstrap(i, 3) = arWaferbootstrapHolder(RandNumber, 3)
    arWaferbootstrap(i, 4) = arWaferbootstrapHolder(RandNumber, 4)
    arWaferbootstrap(i, 5) = arWaferbootstrapHolder(RandNumber, 5)
    arWaferbootstrap(i, 6) = arWaferbootstrapHolder(RandNumber, 6)
    arWaferbootstrap(i, 7) = arWaferbootstrapHolder(RandNumber, 7)
    arWaferbootstrap(i, 8) = arWaferbootstrapHolder(RandNumber, 8)
    arWaferbootstrap(i, 9) = arWaferbootstrapHolder(RandNumber, 9)
  Next i

  TA = FuncTA(strDefType, arWaferHolder:=arWaferbootstrap)

  TGA = FuncTGA(strDefType, arWaferHolder:=arWaferbootstrap)

  TG = FuncTG(arWaferHolder:=arWaferbootstrap)

  T = upperbound

  arRFPHolder(intNoReplicatesCnt, 1) = FuncFP(T, TG, TA, TGA, strDefType, _
                        arWaferHolder:=arWaferbootstrap)


  Next intNoReplicatesCnt


  avg = FuncAvgBootFP("1", arRFPHolder:=arRFPHolder)
  StDev = FuncStDevBootFP("1", avg, arRFPHolder:=arRFPHolder)

  ZFunc = (BootFP - SampFP) / (StDev / 5)


End Function

Sub GetbootstrapT(ByVal strDefType As String, ByVal sngConfLevel As Single, _
  ByVal SampFP As Single, ByVal intCINoCnter As Integer, _
  ByRef arRFPHolder() As Single, ByRef arCIHolder)
```

```
Dim SampFPrnd As Double
Dim SampFPfaults As Double
Dim AvgBootFP As Single
Dim StDevFP As Single
Dim alpha As Single
Dim intCol As Integer
Dim intLRow As Integer
Dim intURow As Integer
Dim boottlow As Double
Dim boothigh As Single

  alpha = 1 - sngConfLevel
  AvgBootFP = FuncAvgBootFP(strDefType, arRFPHolder:=arRFPHolder)
  StDevFP = FuncStDevBootFP(strDefType, AvgBootFP, arRFPHolder:=arRFPHolder)

  If strDefType = "A" Then
    intCol = 4
  ElseIf strDefType = "B" Then
    intCol = 5
  Else
    intCol = 6
  End If

  CreateSortedTArray strDefType, arRFPHolder:=arRFPHolder

  intLRow = Int(UBound(arRFPHolder, 1) * alpha / 2) + 1
  intURow = Int(UBound(arRFPHolder, 1) * (1 - alpha / 2)) + 1

  boottlow = arRFPHolder(intLRow, intCol)
  boothigh = arRFPHolder(intURow, intCol)

  arCIHolder(intCINoCnter, 1) = boottlow
  arCIHolder(intCINoCnter, 2) = boothigh
  arCIHolder(intCINoCnter, 3) = SampFP - boothigh * StDevFP
  arCIHolder(intCINoCnter, 4) = SampFP - boottlow * StDevFP


End Sub

Sub DisplayBR(B As Integer, SampFPA As Single, SampFPB As Single, SampFPC As Single, _
  arRFPHolder() As Single)
Dim colR As New Collection
Dim intR
Dim SampFP(1 To 3) As Single
Dim i As Integer
Dim Sum As Double
Dim arBR() As Single
Dim intRow As Integer
Dim intCol As Integer

  SampFP(1) = SampFPA
  SampFP(2) = SampFPB
  SampFP(3) = SampFPC
```

```
    For i = 1 To B / 20
       colR.Add 20 * i
    Next i

    ReDim arBR(1 To colR.Count, 1 To 4)

    For Each intR In colR
       arBR(intR / 20, 1) = intR
    Next intR

    For intCol = 1 To 3
       For Each intR In colR
          Sum = 0
          For i = 1 To intR
             Sum = Sum + arRFPHolder(i, intCol)
          Next i
          arBR(intR / 20, intCol + 1) = Sum / intR - SampFP(intCol)
       Next intR
    Next intCol

    With Worksheets("BR")
       For intRow = 1 To UBound(arBR, 1)
          .Cells(intRow + 1, 1).Value = arBR(intRow, 1)
          .Cells(intRow + 1, 2).Value = arBR(intRow, 2)
          .Cells(intRow + 1, 3).Value = arBR(intRow, 3)
          .Cells(intRow + 1, 4).Value = arBR(intRow, 4)
       Next intRow
    End With


End Sub

Sub DisplayVariance(B As Integer, arRFPHolder() As Single)
Dim colR As New Collection
Dim intR
Dim i As Integer
Dim Sum As Double
Dim Sum2 As Double
Dim arVR() As Single
Dim intRow As Integer
Dim intCol As Integer
Dim avg As Double

    For i = 1 To B / 20
       colR.Add 20 * i
    Next i

    ReDim arVR(1 To colR.Count, 1 To 4)

    For Each intR In colR
       arVR(intR / 20, 1) = intR
    Next intR

    For intCol = 1 To 3
      For Each intR In colR
        Sum = 0
        For i = 1 To intR
          Sum = Sum + arRFPHolder(i, intCol)
        Next i
        avg = Sum / intR
        Sum2 = 0
        For i = 1 To intR
          Sum2 = Sum2 + (arRFPHolder(i, intCol) - avg) ^ 2
        Next i
        arVR(intR / 20, intCol + 1) = Sum2 / (intR - 1)
      Next intR
    Next intCol

    With Worksheets("VarR")
      For intRow = 1 To UBound(arVR, 1)
        .Cells(intRow + 1, 1).Value = arVR(intRow, 1)
        .Cells(intRow + 1, 2).Value = arVR(intRow, 2)
        .Cells(intRow + 1, 3).Value = arVR(intRow, 3)
        .Cells(intRow + 1, 4).Value = arVR(intRow, 4)
      Next intRow
    End With

End Sub

Sub DisplayRFP(intNoofDisp As Integer, intNoCI As Integer, intNoCICnter As Integer, _
    arRFPHolder() As Single)
Dim intRow As Integer
Dim n As Integer
Dim intN As Integer
Dim i As Integer
    n = intNoCI - intNoCICnter + 1
    intN = intNoofDisp - n

    With Worksheets("RFP")
      For intRow = 1 To UBound(arRFPHolder, 1)
        If n = intNoofDisp Then
          .Cells(intRow + 1, 1).Value = intRow
          For i = 1 To intNoofDisp
            .Cells(1, 2 + 3 * (i - 1)).Value = i
          Next i
        End If
        .Cells(intRow + 1, 2 + 3 * intN).Value = arRFPHolder(intRow, 1)
        .Cells(intRow + 1, 3 + 3 * intN).Value = arRFPHolder(intRow, 2)
        .Cells(intRow + 1, 4 + 3 * intN).Value = arRFPHolder(intRow, 3)
      Next intRow
    End With

End Sub
Sub GetBCACI(ByVal sngConfLevel As Single, _
    ByVal SampFPA As Single, ByVal SampFPB As Single, ByVal SampFPC As Single, _
    intCINoCnter As Integer, ByRef arRFPHolder() As Single, ByRef arBCACIAHolder() _
    As Single, ByRef arBCACIBHolder() As Single, ByRef arBCACICHolder() As Single, _
    ByRef arWaferHolder() As Integer)
Dim SampFPrnd As Double
Dim SampFPfaults As Double
Dim AvgBootFP As Single
Dim StDevFP As Single
Dim alpha As Single
Dim intCol As Integer
Dim upperbound As Integer
Dim upperbound2 As Integer
Dim zo As Double
Dim i As Integer
Dim k As Integer
Dim n As Single
Dim arWaferlessone() As Integer
Dim arSampFP(1 To 3) As Single
Dim arzo(1 To 3) As Single
Dim arSum(1 To 3) As Double
Dim ara(1 To 3) As Double
Dim arAvg(1 To 3) As Double
Dim aralpha1(1 To 3) As Double
Dim aralpha2(1 To 3) As Double
Dim arFPlow(1 To 3) As Double
Dim arFPhigh(1 To 3) As Double
Dim arFP() As Double
Dim TAdefA As Integer
Dim TAdefB As Integer
Dim TAdefC As Integer
Dim TGAdefA As Integer
Dim TGAdefB As Integer
Dim TGAdefC As Integer
Dim TG As Integer
Dim T As Integer
Dim topguy As Double
Dim bottguy As Double
Dim intLRow As Integer
Dim intURow As Integer
Dim colnumb As Integer

  arSampFP(1) = SampFPA
  arSampFP(2) = SampFPB
  arSampFP(3) = SampFPC

  alpha = 1 - sngConfLevel
  upperbound = UBound(arRFPHolder, 1)

  For intCol = 1 To 3
    n = 0
    For i = 1 To upperbound
      If arRFPHolder(i, intCol) < arSampFP(intCol) Then
        n = n + 1
        End If
    Next i
    If n = 0 Then
        n = 0.01
    End If
    arzo(intCol) = WorksheetFunction.NormSInv(n / upperbound)
Next intCol


upperbound2 = UBound(arWaferHolder, 1)

ReDim arFP(1 To upperbound2, 1 To 3)

For i = 1 To upperbound2
    ReDim arWaferlessone(1 To upperbound2 - 1, 1 To 9)
    For k = 1 To upperbound2
        If k <> i Then
            If k > i Then
                For colnumb = 1 To 9
                    arWaferlessone(k - 1, colnumb) = arWaferHolder(k, colnumb)
                Next colnumb
            Else
                For colnumb = 1 To 9
                    arWaferlessone(k, colnumb) = arWaferHolder(k, colnumb)
                Next colnumb
            End If
        End If
    Next k
    TAdefA = FuncTA("A", arWaferHolder:=arWaferlessone)
    TAdefB = FuncTA("B", arWaferHolder:=arWaferlessone)
    TAdefC = FuncTA("C", arWaferHolder:=arWaferlessone)

    TGAdefA = FuncTGA("A", arWaferHolder:=arWaferlessone)
    TGAdefB = FuncTGA("B", arWaferHolder:=arWaferlessone)
    TGAdefC = FuncTGA("C", arWaferHolder:=arWaferlessone)

    TG = FuncTG(arWaferHolder:=arWaferlessone)
    T = UBound(arWaferlessone, 1)

    arFP(i, 1) = FuncFP(T, TG, TAdefA, TGAdefA, "A", _
                            arWaferHolder:=arWaferlessone)
    arFP(i, 2) = FuncFP(T, TG, TAdefB, TGAdefB, "B", _
                            arWaferHolder:=arWaferlessone)
    arFP(i, 3) = FuncFP(T, TG, TAdefC, TGAdefC, "C", _
                            arWaferHolder:=arWaferlessone)

Next i

For intCol = 1 To 3
    For i = 1 To upperbound2
        arSum(intCol) = arFP(i, intCol) + arSum(intCol)
    Next i
Next intCol
```

```
For intCol = 1 To 3
   arAvg(intCol) = arSum(intCol) / upperbound2
Next intCol


For intCol = 1 To 3
   topguy = 0
   bottguy = 0
For i = 1 To upperbound2
   topguy = (arAvg(intCol) - arFP(i, intCol)) ^ 3 + topguy
   bottguy = (arAvg(intCol) - arFP(i, intCol)) ^ 2 + bottguy
Next i
   If bottguy = 0 Then
      ara(intCol) = 0
   Else
      ara(intCol) = topguy / (6 * bottguy ^ 1.5)
   End If
Next intCol

For intCol = 1 To 3
   aralpha1(intCol) = WorksheetFunction.NormSDist(arzo(intCol) + _
   (arzo(intCol) + WorksheetFunction.NormSInv(alpha / 2)) / (1 - ara(intCol) * _
   (arzo(intCol) + WorksheetFunction.NormSInv(alpha / 2))))
   aralpha2(intCol) = WorksheetFunction.NormSDist(arzo(intCol) + _
   (arzo(intCol) + WorksheetFunction.NormSInv(1 - alpha / 2)) / (1 - ara(intCol) * _
   (arzo(intCol) + WorksheetFunction.NormSInv(1 - alpha / 2))))
Next intCol

For intCol = 1 To 3
   intLRow = Int(UBound(arRFPHolder, 1) * aralpha1(intCol)) + 1
   intURow = Int(UBound(arRFPHolder, 1) * aralpha2(intCol)) + 1
   arFPlow(intCol) = arRFPHolder(intLRow, intCol)
   arFPhigh(intCol) = arRFPHolder(intURow, intCol)
Next intCol

arBCACIAHolder(intCINoCnter, 1) = arzo(1)
arBCACIAHolder(intCINoCnter, 2) = ara(1)
arBCACIAHolder(intCINoCnter, 3) = aralpha1(1)
arBCACIAHolder(intCINoCnter, 4) = aralpha2(1)
arBCACIAHolder(intCINoCnter, 5) = arFPlow(1)
arBCACIAHolder(intCINoCnter, 6) = arFPhigh(1)

arBCACIBHolder(intCINoCnter, 1) = arzo(2)
arBCACIBHolder(intCINoCnter, 2) = ara(2)
arBCACIBHolder(intCINoCnter, 3) = aralpha1(2)
arBCACIBHolder(intCINoCnter, 4) = aralpha2(2)
arBCACIBHolder(intCINoCnter, 5) = arFPlow(2)
arBCACIBHolder(intCINoCnter, 6) = arFPhigh(2)

arBCACICHolder(intCINoCnter, 1) = arzo(3)
arBCACICHolder(intCINoCnter, 2) = ara(3)
arBCACICHolder(intCINoCnter, 3) = aralpha1(3)
```

```
    arBCACICHolder(intCINoCnter, 4) = aralpha2(3)
    arBCACICHolder(intCINoCnter, 5) = arFPlow(3)
    arBCACICHolder(intCINoCnter, 6) = arFPhigh(3)

End Sub
```