DETECTION AND DIAGNOSIS OF OUT-OF-SPECIFICATION FAILURES IN MIXED-SIGNAL CIRCUITS

A Dissertation

by

PARIJAT MUKHERJEE

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Peng Li
Committee Members,	Gwan S. Choi
	Jose Silva-Martinez
	Vivek Sarin
Head of Department,	Chanan Singh

December 2014

Major Subject: Computer Engineering

Copyright 2014 Parijat Mukherjee

ABSTRACT

Verifying whether a circuit meets its intended specifications, as well as diagnosing the circuits that do not, is indispensable at every stage of integrated circuit design. Otherwise, a significant portion of fabricated circuits could fail or behave correctly only under certain conditions. Shrinking process technologies and increased integration has further complicated this task. This is especially true of mixed-signal circuits, where a slight parametric shift in an analog component can change the output significantly. We are thus rapidly approaching a proverbial wall, where migrating existing circuits to advanced technology nodes and/or designing the next generation circuits may not be possible without suitable verification and debug strategies. Traditional approaches target accuracy and not scalability, limiting their use to high-dimensional systems.

Relaxing the accuracy requirement mitigates the computational cost. Simultaneously, quantifying the level of inaccuracy retains the effectiveness of these metrics. We exercise this accuracy vs. turn-around-time trade-off to deal with multiple mixed-signal problems across both the pre- and post-silicon domains. We first obtain approximate failure probability estimates along with their confidence bands using limited simulation budgets. We then generate "failure regions" that naturally explain the parametric interactions resulting in predicted failures. These two pre-silicon contributions together enable us to estimate and reduce the failure probability, which we demonstrate on a high-dimensional phase-locked loop test-case.

We leverage this pre-silicon knowledge towards test-set selection and post-silicon debug to alleviate the limited controllability and observability in the post-silicon domain. We select a set of test-points that maximizes the probability of observing

failures. We then use post-silicon measurements at these test-points to identify systematic deviations from pre-silicon belief. This is demonstrated using the phase-locked loop test-case, where we boost the number of failures to observable levels and use the obtained measurements to root-cause underlying parametric shifts.

The pre-silicon contributions can also be extended to perform equivalence checking and to help diagnose detected model-mismatches. The resultant calibrated model allows us to apply our work to the system level as well. The equivalence checking and model-mismatch diagnosis is successfully demonstrated using a high-level abstraction model for the phase-locked loop test-case.

DEDICATION

To all my teachers over the years,

my parents being the first and foremost among them.

I dedicate this dissertation to every individual who has made it possible for me to be here today. While naming every person who has touched my life in some way is impossible, a few notable individuals deserve special mention.

First and foremost are my family, especially my parents and sister. They have backed me up and given me their complete support with every decision I have made in life. I would particularly like to thank my parents for instilling in me the thirst for knowledge and the drive to continually challenge myself.

Next of course are all my teachers for being a source of inspiration, motivation, and knowledge. I would especially like to mention Dr. B. Venkataramani for encouraging me to pursue my interests and in the process, motivating me towards my PhD. And of course, Dr. Peng Li for all the support and guidance he has provided ever since, be it within or outside the domain of academics and research.

Also deserving a special mention are all my friends over the years for their support, encouragement, and for making every facet of my life much more pleasant. Of these, I would especially like to thank Sapna for always standing by me through thick and thin.

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Peng Li for being a strong guiding force throughout the course of my graduate studies. He has not just been a constant source of support, guidance and motivation; he has encouraged me to grow into a researcher in my own right by exhorting and enabling me to follow my own ideas. For this and more, he has my everlasting gratitude.

I would like to take this opportunity to thank all the industry liaisons who have helped shape the course of my research over the years. Notably, I would like to thank Dr. Chirayu Amin, Dr. Chenjie Gu, Dr. Eli Chiprout and Dr. Noel Menezes from Intel Corp. for not just keeping me aligned with the current needs of the industry through numerous inspiring discussions, but also personally guiding me during my summers at Intel.

Special thanks are also due to my committee members - Dr. Jose-Silva Martinez, Dr. Gwan Choi and Dr. Vivek Sarin - for their constant support and guidance. Besides their academic input, the constructive comments I have received at every stage of my dissertation have been invaluable.

No acknowledgement would be complete without recognizing the numerous faculty members from both the Electrical and Computer Engineering Department and the Computer Science Department who have equipped me with the knowledge and tools I needed to work on diverse research problems. Nor would it be complete without thanking all the technical and administrative staff for providing an environment so conducive to learning and research.

Last but not the least of course, I would like to thank my friends and colleagues both at Texas A&M University and outside for all their encouragement and feedback.

NOMENCLATURE

A/MS	Analog / Mixed-Signal
BART	Bayesian Additive Regression Trees
CAD	Computer Aided Design
DC	Direct Current
DfT	Design-for-Test
DUT	Device Under Test
IC	Integrated Circuit
LHS	Latin Hypercube Sampling
KDE	Kernel Density Estimation
PVT	Process, Voltage, Temperature
RIPPER	Repeated Incremental Pruning to Produce Error Reduction
SAT	Simulation After Test
SBT	Simulation Before Test
SoC	System on Chip
SPICE	Simulation Program with Integrated Circuit Emphasis
SRAM	Static Random-Access Memory

TABLE OF CONTENTS

Page

ABSTRACT . . . ii

DEDICATION . . . iv

ACKNOWLEDGEMENTS . . . v

NOMENCLATURE . . . vi

TABLE OF CONTENTS . . . vii

LIST OF FIGURES . . . x

LIST OF TABLES . . . xv

1. INTRODUCTION . . . 1

1.1 Failure probability estimation . . . 5
1.2 Failure region based diagnosis . . . 8
1.3 Equivalence checking and diagnosing model mismatch . . . 10
1.4 Test set selection to maximize observed failures . . . 11
1.5 Identifying systematic shifts in pre-silicon belief . . . 13
1.6 Organization of the dissertation . . . 13

2. BACKGROUND AND RELATED WORK . . . 16

2.1 Analog/mixed-signal circuits . . . 16
2.1.1 Design . . . 19
2.1.2 Modelling and simulation . . . 21
2.1.3 Yield estimation and optimization . . . 22
2.1.4 Analog and mixed-signal testing . . . 23
2.2 Mixed signal circuit as a statistical model . . . 26
2.2.1 Parameters, properties and models . . . 28
2.2.2 Probability of failure . . . 29
2.2.3 Controllability and observability . . . 31
2.3 Objective of this dissertation . . . 32

3. FAILURE PROBABILITY ESTIMATION * . . . 33

3.1 Interval learner . . . 36
3.1.1 Implementation details . . . 39
3.1.2 Tuning the failure model . . . 40
3.1.3 Enhancing accuracy around sampled points . . . 42
3.1.4 Exercising the failure model . . . 42
3.2 Bias compensation . . . 43
3.3 Adaptive sampling . . . 45
3.3.1 Implementation details . . . 48
3.4 Results . . . 50
3.4.1 The test case . . . 50
3.4.2 Failure probability interval . . . 52
3.4.3 Adaptive sampling . . . 55
3.5 Summary . . . 56

4. FAILURE REGION BASED DIAGNOSIS * . . . 57

4.1 Region primitives . . . 58
4.1.1 Decision trees . . . 58
4.1.2 Ensemble learning . . . 62
4.1.3 Implementation details . . . 63
4.2 Post processing . . . 66
4.2.1 Ranking regions . . . 67
4.2.2 Pruning regions . . . 68
4.2.3 Aggregating regions . . . 68
4.3 Parameter ranking . . . 70
4.4 Results . . . 72
4.4.1 Failure regions . . . 72
4.4.2 Parameter ranking . . . 72
4.4.3 Failure region based pre-silicon diagnosis . . . 73
4.5 Summary . . . 78

5. EQUIVALENCE CHECKING AND DIAGNOSING MODEL MISMATCH 79

5.1 Equivalence checking as a property checking problem . . . 79
5.1.1 Practical considerations . . . 81
5.2 Results . . . 82
5.2.1 Equivalence checking . . . 84
5.2.2 Diagnosing model mismatch . . . 85
5.3 Summary . . . 87

6. TEST SET SELECTION TO MAXIMIZE OBSERVED FAILURES . . . 88

6.1 The test plan . . . 89
6.2 Leveraging failure regions for test set selection . . . 91

6.2.1 Implementation details 93
6.2.2 Failure region based test coverage 95
6.3 Results 96
6.4 Summary 98

7. IDENTIFYING SYSTEMATIC SHIFTS IN PRE-SILICON BELIEF . . . 99

7.1 Identifying shifts in importance of “failure regions” 100
7.1.1 Kernel density estimation 101
7.1.2 Implementation details 103
7.2 Results 104
7.3 Summary 106

8. CONCLUSIONS AND FUTURE WORK 108

8.1 Future work 109

REFERENCES 113

LIST OF FIGURES

FIGURE Page

1.1 Integrated circuit design flow. Dotted lines signify that debug & fix might require additional upstream effort. Verification after the last step (fabrication) is usually referred to as test and suffers from limited controllability and observability into the actual integrated circuit. . . 2

1.2 Multiple distributions in the parameter space interact with each other in the circuit in a complex non-linear fashion to give rise to other distributions in the property space. Failures marked in red in the property space can be mapped back to multiple combinations of parameters (also marked in red) as shown. 3

1.3 High level landscape of failure probability estimation via statistical techniques. The line between system and circuit level techniques is somewhat arbitrary and determined by the cost of transistor level simulation and the dimensionality that can be handled by existing failure probability estimation techniques. 6

1.4 High level overview of "failure region" based diagnosis. Only the pre-silicon part of the picture has been highlighted here. Their possible applications towards post-silicon debug will be listed in Fig. 1.6 . . . 9

1.5 System level techniques depend on circuit level techniques by means of abstraction (models). The accuracy of these system level techniques is thus directly tied to the accuracy of these models, making model validation very important. 11

1.6 Proposed framework that leverages pre-silicon information for post-silicon debug. Boxes marked in green correspond to previously discussed pre-silicon information, while boxes in red correspond the post-silicon aspects of this work. 12

1.7 High level overview of the organization of this dissertation. The pre-silicon data and the failure regions generated in Sections 3 and 4 are used to assist with post-silicon debug in Sections 6 and 7. Simultaneously, the contributions made towards property checking are also extended to equivalence checking in Section 5. 15

2.1 Block diagram of a communication system. Certain amount of effort is always required in the analog domain anytime when dealing with real-world signals. Moreover, generating stable voltage, current and clock references also requires analog functionality. 17

2.2 A single step in the high-level design flow detailed in Fig. 1.1. Verifying functionality and debugging observed failures are integral components at every stage of the design flow. Ensuring correct functionality in the current stage reduces amount of rework that may be required if a problem is discovered in downstream stages. 20

2.3 A black box view of a mixed-signal circuit. The circuit can be thought of as a parametric model where the parameters in question refer to only process parameters at this point. This picture will be further enhanced in Fig. 2.4. 26

2.4 A completely parameterized view of a mixed-signal circuit. Both the inputs to the circuit and the model itself have now been parametrized. The combined parameter space can now be thought to contain input signal parameters, process parameters, sources of noise etc. 27

3.1 Failure regions in property distribution. The area under the probability distribution curve marked in red gives us the probability of failure that we wish to estimate. 34

3.2 High level overview of proposed approach. The interval learner is described in Section 3.1, the bias compensation in Section 3.2 and the adaptive sampling built around it in Section 3.3. 35

3.3 Point wise prediction and error. For every point $x \in X$, we obtain a distribution over $y|x$ instead of a single y. Both a pass-fail prediction and probability of misprediction can thus be obtained. 37

3.4 Distribution of y from Fig. 3.1 overlaid with probability of misclassification. This probability of misclassification is highest at the failure boundary and tapers off as we move away from it. 39

3.5 Adaptive sampling demonstration. Fig. 3.5(a) shows the concept we wish to capture and Fig. 3.5(d) how well we managed to do so. As evident from the final set of sampled points in Fig. 3.5(c), sampling concentration is dependent on both proximity to boundary and how important the boundary is. 46

3.6 Adaptive sampling framework: The failure model and how it may be exercised has already been discussed in Section 3.1. The role of K-fold sub-sampling and bias compensation have also been discussed previously in Section 3.2. 47

3.7 Failure probability interval convergence during adaptive sampling. Fig. 3.7(a) shows the original bias compensated interval. Fig. 3.7(b) shows how the interval converges during the course of adaptive sampling. Fig. 3.7(c) shows the final interval. 49

3.8 High level block diagram of the PLL test case. Two variants of this circuit, $PLL1$ and $PLL2$ will be used in Sections 3, 4, 6 & 7. $PLL1$ will in turn be compared to a behavioral model with the same block diagram but a smaller set of parameters in Section 5. 51

3.9 PLL1 adaptive sampling demonstration. Since the cost of SPICE simulation is dominant, compute time is reported in terms of number of SPICE simulations required. We are able to get to a stable failure probability estimate within a few hundred simulations. 55

4.1 Region primitives : two dimensional example. $X1$ and $X2$ are two independent Gaussian parameters. Regions that do not satisfy specifications are marked in red. Each rectangle corresponds to a failure region primitive we wish to generate. 59

4.2 Rule learning from decision trees. Each node in the tree refers to a split along a given parameter. A path traced from the root node to a leaf rich in failure cases gives us a failure rule. There may be more than one such leaf as shown above. 60

4.3 Decision tree learning : two dimensional example demonstrating the over-fitting vs. generalization trade-off when performing rule-induction using a single decision tree. All naming conventions are the same as in Fig. 4.1. 61

4.4 Sum of trees model. Solid lines: interaction effects; Dotted lines: additive effects. Paths leading to a failure are marked in red. The green circle represents how complex non-linear failure mechanisms may be explained by combining simpler rules. 62

4.5 Ensemble learning : two dimensional example demonstrating how multiple overlapping rules can guarantee sufficient coverage of the non-linear concept we try to explain. Picking a subset of these rules while ensuring sufficient coverage is the purpose of the post-processing step discussed in Section 4.2. 63

4.6 Failure region discovery on adaptively sampled data. The above figure assumes that we are able to query a failure model as in Section 3. Fig. 4.6(a) is thus used to construct the ensemble and Fig. 4.6(b) to determine the probabilities associated with each region. The final set of processed rules are shown in Fig. 4.6(c). 69

4.7 Sample Rule : $P(R)$ represents the probability that the region is excited based on pre-silicon knowledge. $P(F|R)$ is the probability of failure within that region. The boundaries listed correspond to only the critical parameters. The non-critical parameters default to their entire legal range of values. 73

4.8 Failure regions : PLL1. The naming of all parameters and properties are self-explanatory but for the following exceptions : (1) Filter R1, Cx as shown in Fig. 4.10, (2) Transistor channel lengths in the charge pump block (LCPxx) and VCO block (LVxx). All other naming conventions are the same as Fig. 4.7. 74

4.9 Failure regions : PLL2. Naming conventions are the same as in Fig. 4.7 and Fig. 4.8. 75

4.10 Charge pump and filter blocks within the PLL described in Fig. 3.8. $PLL1$ and $PLL2$ end up differing mainly in the values of LCPxx and R1, Cx due to the nature of the diagnosis information obtained for $PLL2$. 77

5.1 Redefining the equivalence checking problem as a property checking problem. Instead of dealing with each model individually, we operate across a joint parameter space defined by the union of both parameter spaces and over a joint property defined by the difference between shared properties. 80

5.2 Difficulty when adaptively sampling the equivalence checking problem as compared to the property checking problem. Though Fig. 5.2(a) and Fig. 5.2(b) are very similar to each other, on comparing Model 1 with Model 2, we get a doubly complex failure manifold in Fig. 5.2(c). 82

5.3 Failure regions : PLL1 vs Verilog-AMS model mismatch. Naming conventions are the same as in Fig. 4.7 and Fig. 4.8. 86

6.1 Failure region based test selection methodology. The uncertainty information $p(\bar{c}|c)$ can be fed back from the fabrication process. The test-engineer's input is used in determining the test plan (and thus $p(c)$) as discussed in Section 6.1. The failure regions serve as a source of pre-silicon knowledge for $P(F|c, \bar{c})$. 89

6.2 Selecting values for controllable test parameters. X1 is the controllable parameter. For every sampled point on X1, the expected uncertainty along X2 is demonstrated. The expected uncertainty is based on pre-silicon assumptions. The actual post-silicon distribution may be different as shown in Fig. 7.1. 92

6.3 PLL1 Test selection. Probabilities of regions demarcated by one or more controllable parameters are effectively boosted. If the region is bounded by uncontrollable elements alone, no test set can perform better than randomly sampling the test plan. 97

6.4 PLL2 Test selection. Regions with failure probability above 0.5% when excited with chosen test parameters will be considered statistically significant and used for post-silicon debug. 97

7.1 Post silicon debug. X1 is controllable and can be selected during test. Change in distribution of the unobservable parameter X2 causes $P(R_i)$ to change for both clusters. 100

7.2 Complete post-silicon debug flow. The estimated failure region probabilities over both pre and post-silicon data are compared to identify systematic shifts from the pre-silicon belief in the post-silicon domain. 102

7.3 PLL1 debug results. The importance of all regions except region R_2 seems to go up in the post-silicon domain. Studying region R_2 in more detail actually allows us to identify that the mean value of $C1$ increased in the post-silicon domain. 105

7.4 PLL2 debug results. Regions $R_{1.1}$ and $R_{1.14}$ show substantial deviations and will be examined further. 106

LIST OF TABLES

TABLE Page

3.1 Failure rate and interval estimates for PLL1. All probabilities expressed in %. MC+LHS refers to data sampled according to the parameter distribution though using Latin Hypercube Sampling. Adaptive refers to the data sampled adaptively as per our proposed approach. 53

3.2 Failure rate and interval estimates for PLL2. All probabilities expressed in %. MC+LHS refers to data sampled according to the parameter distribution though using Latin Hypercube Sampling. Adaptive refers to the data sampled adaptively as per our proposed approach. 54

4.1 Top 5 parameters in $PLL1$ ranked based on the importance metric described in Section 4.3. 74

4.2 Top 5 parameters in $PLL2$ ranked based on the importance metric described in Section 4.3. 76

5.1 Model mismatch probability and interval estimates for PLL1 when compared against high level Verilog-AMS model. All probabilities expressed in %. 84

5.2 Top 5 parameters in explaining mismatch between PLL1 and Verilog-AMS model based on the importance metric described in Section 4.3. 85

1. INTRODUCTION

Real world signals are analog in nature. So it shouldn't be surprising that analog circuits are all around us. Even when dealing with purely digital systems, analog circuits are responsible for providing stable and reproducible voltage and clock references so that the digital core can do its job. Moreover, in today's brave new world of increased integration and System-on-Chips (SoCs), analog and digital systems reside side by side on the same chip making modern day Integrated Circuits (ICs) truly mixed-signal in nature. In fact, such mixed-signal systems are here to stay as discussed further in Section 2.1.

Unfortunately, the design of such analog/mixed-signal components is inherently complex in nature. Since we are forced to deal with transistor level dynamics when designing analog circuits, an enormous number of design considerations must be accounted for simultaneously to allow the circuit to perform its intended function even under nominal operating conditions. And the nominal operating condition is just the starting point. As CMOS technologies continue to scale, we are not just dealing with bigger and more complex circuits, but also a lot of design uncertainties such as startup conditions; signal & power noise; process variation etc. The parameter space that we need to deal with when designing mixed-signal systems has thus grown dramatically, even over the last few years.

Given the number of conditions the circuit has to operate correctly across, it is impossible to think of every possible scenario when designing the circuit. Automated verification tools to ensure correct operation of such mixed-signal systems is thus of prime importance in both the pre-silicon and post-silicon domains. In fact, irrespective of which stage of the design flow (detailed further in Section 2.1.1) we may

be at, verifying whether the circuit performs as intended, and debugging observed functional failures, is essential as shown in Fig. 1.1.

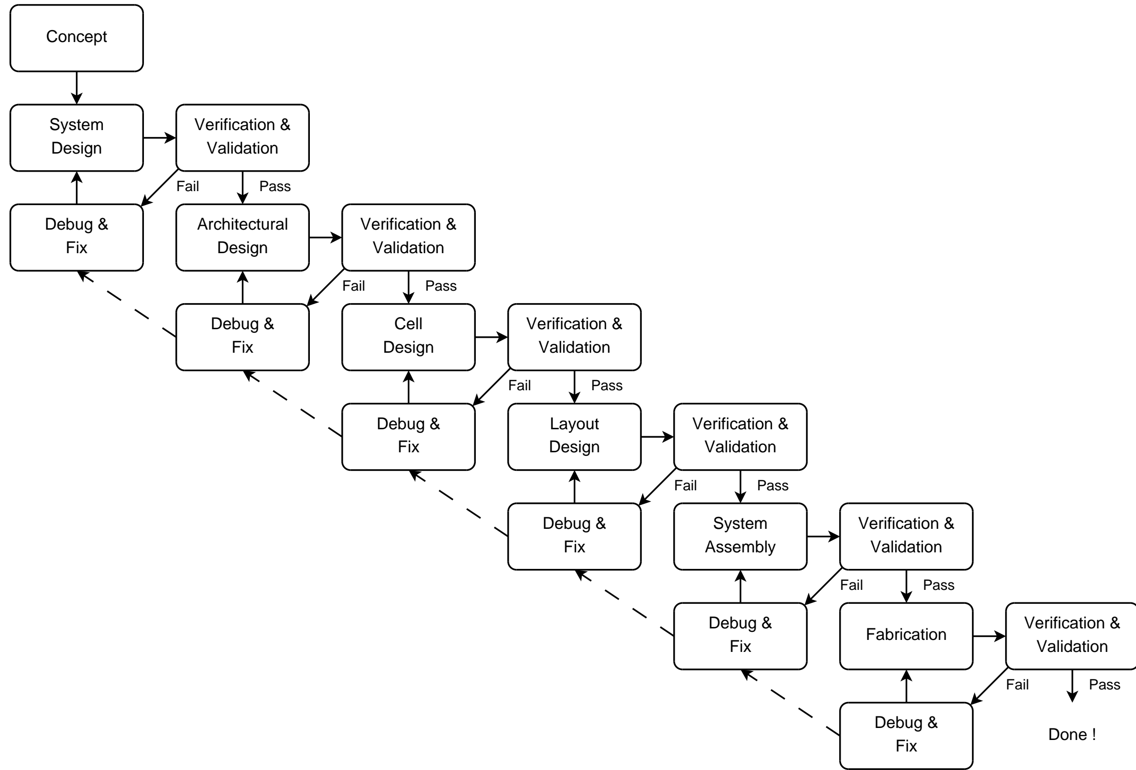


Figure 1.1: Integrated circuit design flow (modified with permission from [1] © 2000 IEEE). Dotted lines signify that debug & fix might require additional upstream effort. Verification after the last step (fabrication) is usually referred to as test and suffers from limited controllability and observability into the actual integrated circuit.

Unfortunately, even verifying that the circuit operates as expected is only slightly easier than designing it. The dimensionality of the parameter space makes it impossible for us to simulate and/or measure every possible scenario. Additionally, both the parameter distributions and the mapping between input parameters and output properties may be arbitrarily complex, forcing us to deal with multiple (and possibly

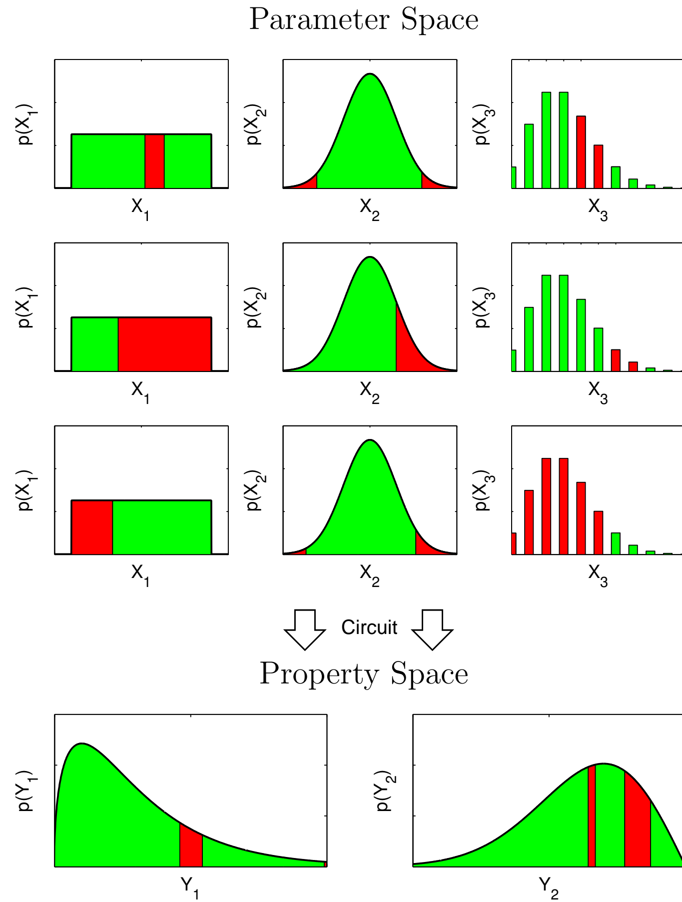


Figure 1.2: Multiple distributions in the parameter space interact with each other in the circuit in a complex non-linear fashion to give rise to other distributions in the property space. Failures marked in red in the property space can be mapped back to multiple combinations of parameters (also marked in red) as shown.

correlated) failure regions across multiple properties. In fact, the scenario can get extremely complex even for a low dimensional system as depicted in Fig. 1.2.

Predicting whether a circuit meets target property specifications has thus grown increasingly difficult over time. Furthermore, simply predicting whether a circuit will meet specifications under all operating conditions is no longer enough. Diagnosing discovered out-of-specification failures has become increasingly challenging,

both due to the multitude of parameters and the fact that failures may be caused by interactions between these parameters.

We are thus rapidly rushing towards a proverbial wall in terms of size and complexity of the mixed-signal circuits that can be designed, unless we devise innovative means to verify and debug such circuits. Estimating how often a circuit will probably fail and diagnosing discovered failures thus form a core constituent of this work.

While the above contributions may address the circuit level problem, state-of-the-art integrated circuits are large mixed-signal systems. It is impossible to simulate these systems at the circuit level. High level models are thus used for system level verification. Verifying whether or not these high level models match the underlying circuit and providing debug information to the designer / model developer if required, indirectly enables system level estimation and diagnosis as well. We thus address the same in this work as well.

Solving these problems in the pre-silicon domain (prior to fabrication) empower the designers with the tools they need for yield optimisation of large mixed-signal systems. Given today's aggressive design margins and turn-around-time requirements, the struggle however does not stop there. A circuit may have "acceptable" yield margins in the pre-silicon domain but may demonstrate unacceptable yields once fabricated.

The mismatch between the pre-silicon expectations and post-silicon (after fabrication) reality may arise from multiple causes but they are all very hard to debug given the lack of observability into the system. By leveraging the large amount of pre-silicon data we collect during the estimation stage and the critical parameters we identify during the diagnosis stage, we further attempt to select test points to boost the probability of observing failures and then identify systematic deviations in the fabrication process using the obtained test measurements.

It is to be noted that the tools and methodologies listed herein are designed to assist the designer / verification engineer and not replace them. A significant amount of domain knowledge is still required to fix observed failures or even determine what experiments may be practical in the post-silicon stage. This holds true whether we are discussing failure probability estimation, diagnosis, test-set selection or root-causing observed post-silicon failures.

We break down each of these problems further in the rest of this section.

1.1 Failure probability estimation

Estimating the probability that a circuit will not meet its intended specifications has received a lot of attention over the years but researchers have been successful in solving only subsets of the overall problem. Formal techniques [2, 3, 4] have difficulty in scaling without grossly approximating the circuit behaviour. On the other hand, the computational complexity of SPICE simulations forces us to compromise when using statistical techniques.

One such compromise requires decomposing a circuit into multiple lower dimensional components. While extremely accurate estimates can be obtained on a per component basis (such as in the yield estimation domain [5, 6, 7, 8]), this divide-and-conquer approach fails to account for correlation between components. As a result, the predicted performance at the system level may be optimistic or pessimistic depending upon whether the components interact with each other to suppress block level failures (such as in compensatory feed-back loops) or further enhance block level failures. Additionally, we still run into scalability issues in the absence of large numbers of repeated components.

An alternate compromise involves relaxing the accuracy requirement. If accuracy isn't a very strong concern and the error rate is high, standard Monte Carlo analysis

[9, 10] can be used directly. However, we additionally need to ensure that the error in this prediction falls within a certain bound. Moreover, it is crucial that these bounds be tight enough to make a design decision. For instance, while knowing that the failure probability lies in the interval $[0.01, 0.02]$ may be sufficient, the same can't be said about $[0.0, 0.5]$. In fact, confidence intervals drawn via bootstrap [11] is a big reason why standard Monte Carlo analysis has remained a strong contender in this area. However, the accuracy of standard Monte Carlo is inversely related to the number of samples and thus the prediction may be extremely biased when trying to estimate low failure probabilities using a limited simulation budget.

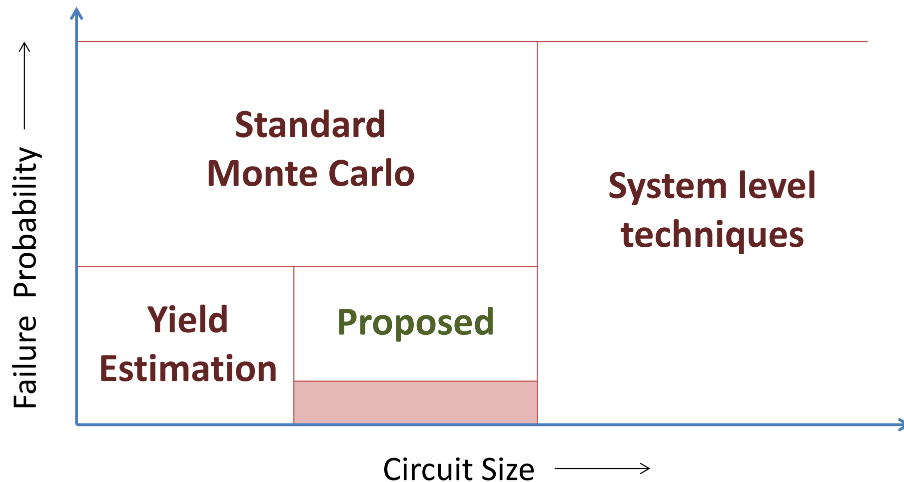


Figure 1.3: High level landscape of failure probability estimation via statistical techniques. The line between system and circuit level techniques is somewhat arbitrary and determined by the cost of transistor level simulation and the dimensionality that can be handled by existing failure probability estimation techniques.

Thus there exists a significant void in the estimation landscape without resorting to divide-and-conquer based system level techniques as shown in Fig. 1.3. We address this void by developing a scheme via which an approximate failure probability

$\bar{P}(F)$ and the corresponding interval $P(F) \in [minP, maxP]$ can be obtained using at most a few hundred SPICE simulations. Since we wish for $\bar{P}(F) \rightarrow P(F)$ as more simulation resources become available, we first need a knowledge representation scheme that returns both $P(F)$ and a pessimistic interval $[minP, maxP]$ when fed with adaptively sampled SPICE data. The designer can then add simulation resources as required until the desired interval accuracy has been achieved.

We describe three main contributions towards these goals as follows:

1. A failure model based knowledge representation scheme that returns both $P(F)$ and a conservative interval $P(F) \in [minP, maxP]$ when fed with arbitrarily sampled SPICE data.

2. Compensation for the bias in the model based estimate from step (1) via an ensemble of learners.

3. An adaptive sampling scheme built around the above bias-compensated learner that not only samples around the failure boundaries, but also takes into account the probability that a boundary is exercised in the first place.

The above contributions allow the designer to start with arbitrarily distributed data and to keep adding simulation resources as required until the desired interval accuracy has been achieved. While the objective is similar to the existing adaptive sampling techniques in the yield estimation domain, the dimensionality of the systems we deal with here means that both the failure probability and the techniques we use to approximate the said failure probability are wildly different.

This work in conjunction with the existing yield analysis and Monte Carlo techniques lays the foundation for a unified approach towards failure probability estimation at the circuit level.

1.2 Failure region based diagnosis

The diagnosis problem can be broken down into two parts:

1. Determining critical parameters whose deviation could cause the circuit to fail.
2. Determining the values (or range of values) for these parameters that result in failure.

Answering the above problems helps in design space exploration and yield optimization in the pre-silicon domain [12, 13, 14]. Existing tools [15, 16] to do so however, mostly assume that a single parameter is capable of causing the circuit to enter a failure region. In reality, interaction effects over a high-dimensional space, defined by input signal parameters and sources of design uncertainty, are more likely to do so. For example, the circuit may not have been designed to meet all specifications in the first place (failures due to interactions between input signal parameters) or multiple design uncertainties interact with each other to give rise to failures (e.g., transistor mismatch). In fact, most design uncertainties manifest a failure only in the presence of specific input conditions.

While techniques exist to deal with all of the above separately, they inherently refer to interaction effects over a high-dimensional space consisting of both the input signal parameters and sources of design uncertainty. As the dimensionality of the parameter space continues to increase, these interaction effects have only become more prominent. Since these interactions refer to a multi-dimensional interval in the parameter space, we can leverage standard rule-induction algorithms that perform both feature selection and clustering of failure information simultaneously. By utilising domain knowledge to rank, prune and cluster these rules, we generate a list of failure regions that can easily be visualised by a designer. This is summarized in Fig. 1.4.

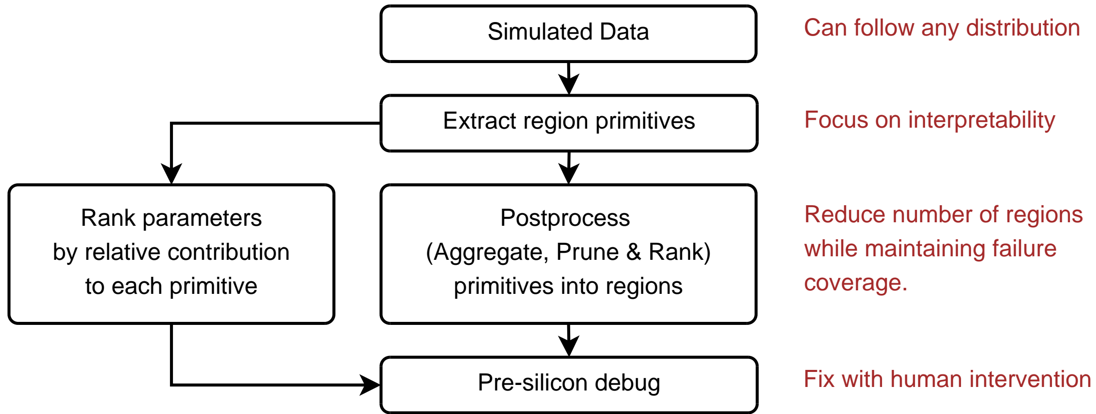


Figure 1.4: High level overview of "failure region" based diagnosis. Only the pre-silicon part of the picture has been highlighted here. Their possible applications towards post-silicon debug will be listed in Fig. 1.6

For the failure regions to be useful, they must first be easy to interpret for the end-user. Since non-linear manifolds are hard to visualise, especially in a high-dimensional space, we propose the use of simple *if-then* rules to serve as region primitives. These simple primitives may then be aggregated to explain complex non-linear manifolds.

However, examining every dimension in a high-dimensional space may not be possible even for simple primitives. Additionally, the limited representational flexibility of interpretable primitives may cause them to overlap with each other or explain very few failures individually. Thus, our complete approach involves the following:

1. Construct failure region primitives using standard rule-induction algorithms.

2. Identify critical parameters and their associated values for each primitive.

3. Prune these primitives to minimise overlaps and maximize predictive accuracy.

4. Aggregate the reduced set of primitives into complex non-linear regions.

5. Rank the above regions based on their probabilities.

6. Rank parameters within each rule to assist with debug.

The real applications of the failure regions of course go further than just pre-silicon diagnosis as discussed here. We shall also outline how they may assist with test-set selection and post-silicon debug in Section 1.4 and Section 1.5 respectively.

1.3 Equivalence checking and diagnosing model mismatch

The failure probability estimation problem we have laid out in Section 1.1 is sometimes referred to as "property checking" wherein we wish to compare a circuit against its specifications. However, there are times when we wish to compare two circuits against each other or two views of the same circuit against each other to see if they are equivalent or not.

The motivation for doing so is more clearly understood by discussing the circuit vs. system-level verification scenario in Fig. 1.5. While we would like to deal with transistor level information as far as possible to avoid abstracting away critical interaction information, it may not be possible to simulate large systems in SPICE within a reasonable turn-around-time.

In such a scenario, we have no choice but to rely on system level techniques which inherently abstract away circuit level information by means of high-level models. Verifying whether these high-level models are equivalent to the underlying circuit under various design uncertainties thus becomes a very important problem.

To deal with this problem, we frame the equivalence checking problem as a property checking problem, which can then be dealt with by using the previously introduced contributions towards failure probability estimation and diagnosis. Forcing two models to be perfectly identical would cause the dimensionality of both models to be similar as well. The higher level abstraction model does not provide any simu-

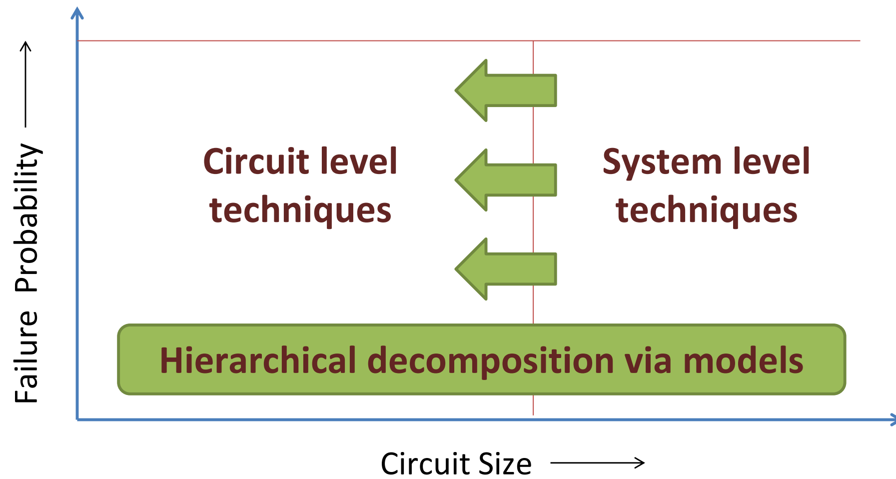


Figure 1.5: System level techniques depend on circuit level techniques by means of abstraction (models). The accuracy of these system level techniques is thus directly tied to the accuracy of these models, making model validation very important.

lation speed-up for system level analysis in this case. Thus, we instead simply ensure that the response of both models are simply within a certain threshold of each other over both sets of model parameters.

1.4 Test set selection to maximize observed failures

Since we have gone through the effort of predicting and optimizing the yield in the pre-silicon domain, we expect failures to be relatively rare in the post-silicon domain. As a result, observing failures when validating the fabricated circuit can be difficult with a very limited measurement budget. At the same time, the same underlying failure mechanisms may lead to failures across hundreds of chips after production, simply because we are dealing with a much larger number of chips.

As a result, some mechanism to boost the number of observed failures in the post-silicon domain is required. Depending upon the device-under-test (DUT), we may be able to directly pick operating conditions or control knobs that affect circuit

functionality. We may also be able to pick a subset of chips by means of process related information obtained during high volume manufacturing. The more the degrees of freedom of course, the larger the set of candidate test-points. The goal of test-set selection is to pick only a few of these test-points, but boost the probability of observing failure to significant enough levels that it can be observed in the post-silicon domain.

To do so, we observe that the failure regions we introduced in Section 1.2 serve as a reasonably confident knowledge representation scheme by demarcating parts of the parameter space where we expect failure to occur. A complete framework that shows all the downstream applications of these failure regions (including test-set selection) is given in Fig. 1.6. For test-set selection in particular, we integrate over the uncontrollable dimensions within each failure region to pick controllable points that have the highest probability of exciting these regions.

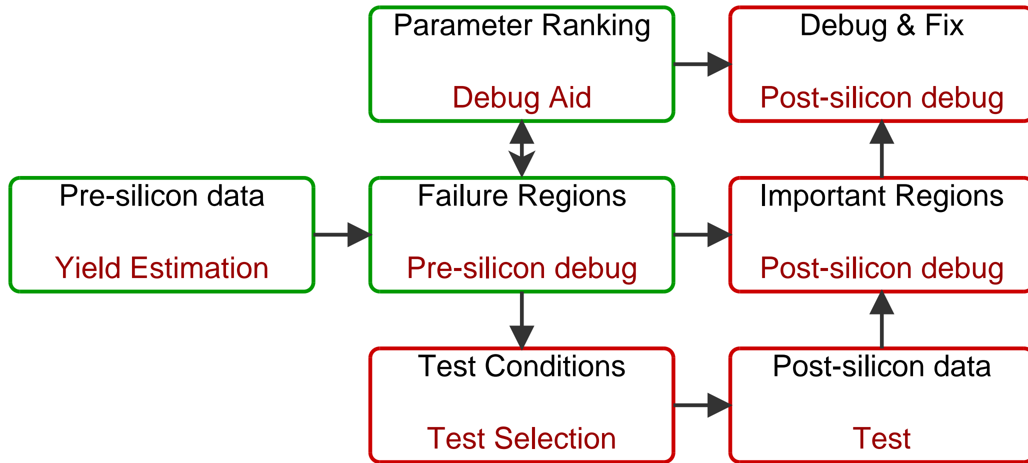


Figure 1.6: Proposed framework that leverages pre-silicon information for post-silicon debug. Boxes marked in green correspond to previously discussed pre-silicon information, while boxes in red correspond the post-silicon aspects of this work.

1.5 Identifying systematic shifts in pre-silicon belief

We have so far discussed failure diagnosis in the pre-silicon domain in Section 1.2. However, the complexity of everything we have discussed so far pales in comparison to the problem of debugging actual silicon, given the limited amount of observability into a real chip. As a result, we try and leverage pre-silicon knowledge captured by the failure regions in Fig. 1.6 to assist with this process.

If the fabrication process replicates the pre-silicon knowledge, then the real yield will be the same as the expected yield and the probability of exercising a given failure region will remain the same over both the pre-silicon and post-silicon domains. However, if this is not the case, the relative importance of specific failure regions might go up or down depending upon deviations in the fabrication process. It is these deviations from the pre-silicon belief in the post-silicon domain that we wish to discover. Since the critical parameters for each failure region is already known from the pre-silicon data, this eventually helps with debug as well.

It is to be noted that this is a significant shift from established post-silicon debug techniques since we are not adding any additional observability into the system. While additional test probes and measurements are naturally supported by our framework, this dissertation treats the design under test as a black box. We do not attempt to add expensive test probes or measurement circuitry but only use existing inputs and outputs. Drawing inferences from limited post-silicon data is still possible because we are not interested in exact parametric shifts anymore, but only the importance of specific failure mechanisms.

1.6 Organization of the dissertation

The organisation of this dissertation is summarised in Fig. 1.7. We first describe the relevant background material in Section 2. We then proceed to address the

property checking problem by estimating the failure probability in Section 3. This technique in conjunction with existing Monte Carlo and yield estimation techniques serve as the basis for our pre-silicon knowledge which we then use to diagnose these violations by identifying failure trends in Section 4.

From this point onwards, the dissertation takes two paths. We first describe how the concepts we used to detect and diagnose property violations in Section 3 and Section 4 can be extended towards equivalence checking and diagnosing model mismatch in Section 5. This proves to be a key enabler for system level detection, diagnosis and debug techniques.

The other path detailed over Section 6 and Section 7 deals with using the pre-silicon knowledge we discover in Section 3 and Section 4 as an aid to deal with test-set selection (Section 6) and identifying systematic shifts in the probability of critical failure mechanisms (Section 7) which help in observing and debugging post-silicon failures.

As mentioned earlier, the key enabler in all of these methods is exercising the accuracy vs. turn-around-time trade-off by means of failure probability intervals, parameter space coverage and probabilities of excitation depending upon the exact nature of the problem - a fact that we summarize in our concluding remarks in Section 8.

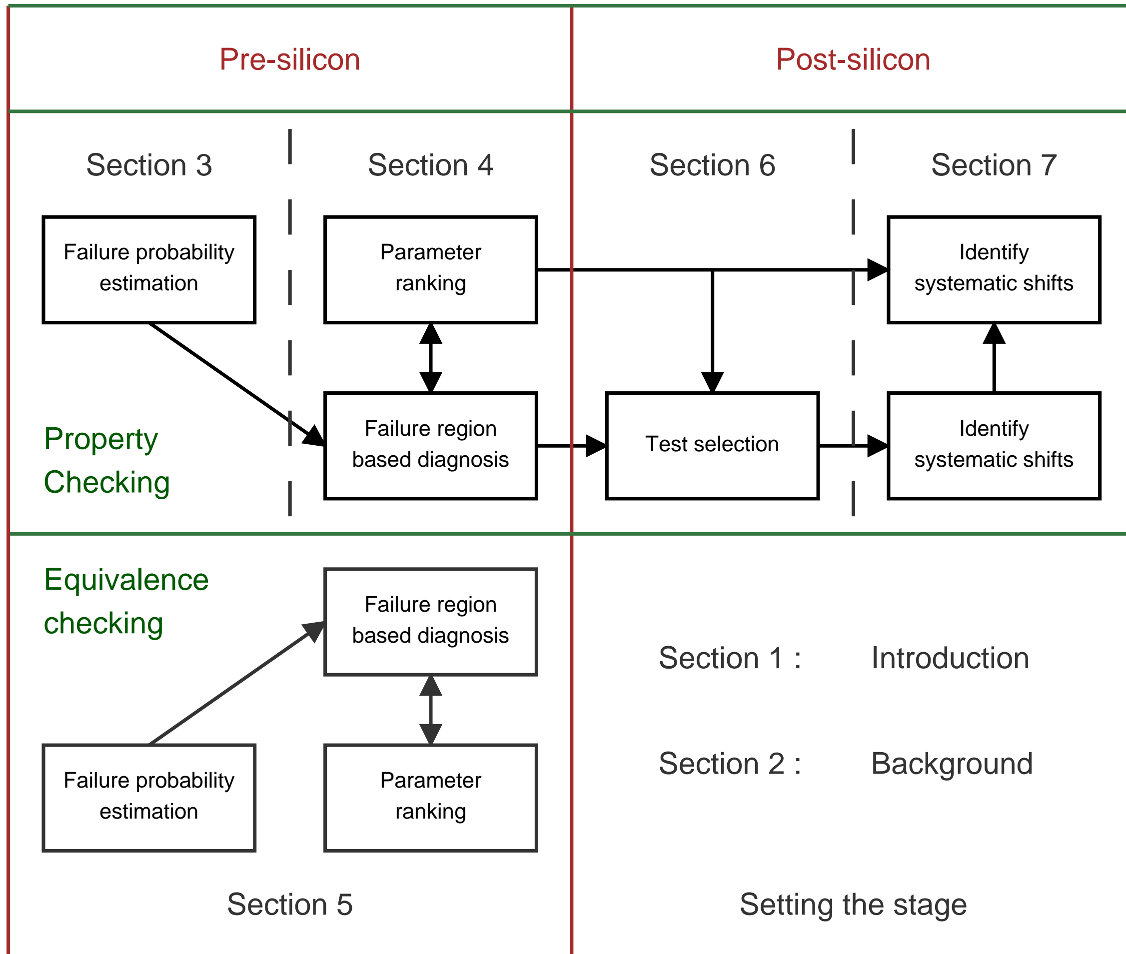


Figure 1.7: High level overview of the organization of this dissertation. The pre-silicon data and the failure regions generated in Sections 3 and 4 are used to assist with post-silicon debug in Sections 6 and 7. Simultaneously, the contributions made towards property checking are also extended to equivalence checking in Section 5.

2. BACKGROUND AND RELATED WORK

2.1 Analog/mixed-signal circuits

Analog/mixed signal circuits are ubiquitous, whether we realise it or not. In fact, as technology continues to shrink and it becomes feasible to integrate more and more onto a single chip, the line between digital integrated circuits and analog integrated circuits continues to grow fuzzier. Instead, we are now dealing with the brave new world of System-on-Chips (SoCs) where both digital and analog functionality reside side by side. The rise of such SoCs has come with its own set of challenges, namely how to design and validate such large mixed-signal systems.

Specifically, this difficulty arises because of the analog components. While we can assume a binary 0/1 response in digital circuits (along with certain setup, hold, fall time restrictions etc.), no such assumptions can be made about analog circuits - making them very difficult to design, simulate and debug. This problem has been further exacerbated by the fact that it is nearly impossible to ensure 100% error-free functionality, given the increase in manufacturing variability [17, 18, 19, 20] in the rush to keep up with Moore's law. While these manufacturing tolerances have hurt digital circuits as well, it has hit analog circuits far worse.

Simultaneously, the tolerances associated with our designs have shrunk with time as well. As our communication needs keep growing, the frequency of operation of related subsystems rises to keep pace with increased bandwidth requirements as well. Additionally, this needs to be achieved while minimizing power consumption as well - both to save energy in mobile form factors such as cellphones and to make sure that the chip doesn't burn itself up from thermal dissipation. As a result, both the time and voltage margins have been shrinking over time for a lot of analog subsystems.

In this world of increased variability, tighter tolerances and ever rising design complexity, it shouldn't be surprising that efforts have been made to replace a lot of analog components with digital ones. Yet, it is nearly impossible to get rid of analog circuits from the modern day Integrated Circuits (ICs). For example, consider the communication subsystem in Fig. 2.1. The core computation may happen in the digital domain but the surrounding circuitry is all mixed-signal or analog in nature.

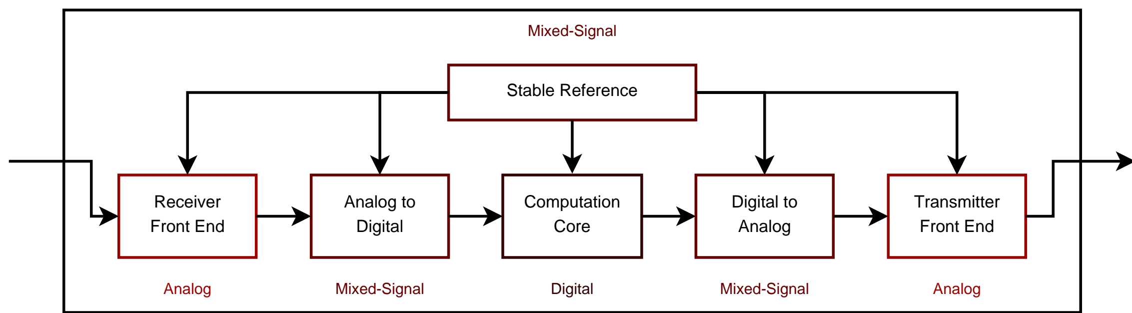


Figure 2.1: Block diagram of a communication system. Certain amount of effort is always required in the analog domain anytime when dealing with real-world signals. Moreover, generating stable voltage, current and clock references also requires analog functionality.

In fact, certain functions will probably always remain in the analog domain [1]:

- Any circuit that directly deals with real world inputs: Real world signals are by nature analog. Even if we wish to process them in the digital domain, we first need to sense, receive, amplify and filter them before digitization. Sensors, amplifiers, filters, oscillators and mixers are thus found in a wide range of SoCs over a wide range of applications.

- On the output side : Even if we are transmitting a sequence of ones and zeros, maintaining the integrity of the sequence over a long lossy channel involves sig-

nificant analog components in order to strengthen, synchronize and buffer the signal so that it can be recovered on the receiver side. The analog components become even more significant when driving signals that are essentially analog in nature. Such signals are commonplace in many applications such as when transmitting audio over a speaker or radio waves over an antenna.

- Connecting the digital and analog worlds : These blocks are the true mixed-signal circuits that interface the above analog circuits with the digital part of the system. Typical circuits used here are the sample-and-hold circuits for signal sampling; analog-to-digital converters for amplitude discretization; digital-to-analog converters for signal reconstruction; and phase-locked loops and frequency synthesizers to generate a timing reference or perform timing synchronization [1].

- Ensuring correct operating conditions : Whether dealing with analog components or digital, all circuits eventually need stable, reproducible and reliable references and bias conditions be it in terms of voltages, currents or clocks. While efforts have been made to move at least part of these circuits to the digital domain, the circuits themselves remain mixed-signal in nature as certain functions must be dealt with in the analog domain.

- Digital circuits : Yes, digital circuits ! The line between analog and digital is simply a matter of abstraction. Voltages close to the ground and VDD in the analog domain are the 0s and 1s in digital. We simply assume that digital circuits operate at either voltage levels 0 or 1 almost all the time. While this is a valid assumption for the most part, as we try and optimize both power and performance in state-of-the-art processors, even the digital circuits are becoming increasingly analog-ish in nature.

As we can see, while the core computation may still happen in the digital domain, analog circuits are indispensable in communicating the necessary data and ensuring correct operating conditions. More importantly, analog dynamics are starting to become important in what were previously purely digital circuits. While the latter is mitigated somewhat by careful design of the underlying standard cell library, increased integration forces us to deal with mixed-signal behavior at some stage of the design cycle. The larger the system we are dealing with of course, the more difficult it is to design, verify and validate.

Dealing with this increased design complexity is nearly impossible without the use of Computer Aided Design (CAD) and verification tools. However, most such tools have traditionally dealt with only parts of the whole and assumed that the system as a whole will work if the individual components are well designed - an assumption that may not be true in practice. More importantly, even if we make this optimal assumption, it is still difficult to design and verify large subsystems under all possible operating / input conditions. Analog components are thus increasingly becoming the bottleneck in terms of design turn-around-time for large mixed-signal integrated circuits.

We now take a more detailed look at this high level picture.

2.1.1 Design

We previously introduced the high level overview of the design flow that goes into manufacturing a modern day integrated circuit [1] in Fig. 1.1. Ensuring correct-by-construction design is difficult even in the digital domain at any of the depicted stages, in spite of the large amount of work that has been put into logic synthesis and automated place & route tools.

The problem is only further exacerbated in the analog domain when dealing with

highly non-linear transistor-level dynamics instead of simple boolean logic as in the case of digital circuits. While there has been work on automated synthesis [21, 22, 23] and transistor sizing [24, 25, 26] in the analog domain as well, the majority of design tasks continue to require a high level of human intervention. Whether human driven, or tool driven, analog design remains extremely prone to error - making verification and validation at every stage even more indispensable.

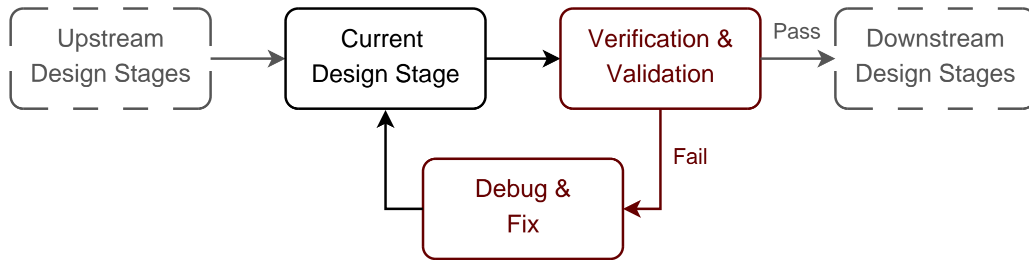


Figure 2.2: A single step in the high-level design flow detailed in Fig. 1.1. Verifying functionality and debugging observed failures are integral components at every stage of the design flow. Ensuring correct functionality in the current stage reduces amount of rework that may be required if a problem is discovered in downstream stages.

Thus every stage in the design flow in Fig. 1.1 is essentially a feedback loop as shown in Fig. 2.2. Irrespective of the level of abstraction we may be working at, verifying that the circuit performs its intended function before we exit this loop is critical. Failure to do so causes the failure mechanism to be cascaded to downstream stages – something that can be very expensive in practice since debug and fix efforts may then incur a significant amount of rework. For example, if the cell level specifications are too aggressive, it may be practically impossible to make a reliable circuit for the same. This may require changing the specifications itself which will in turn require a change in the architecture design. Since process technologies are mostly

not mature when the design cycle is started, an even more commonplace example is where a cell is designed assuming certain process conditions but the actual fabrication process does not meet pre-silicon expectations. Simply tuning the layout may not be sufficient to obtain the required performance in this case.

The more number of steps we have to redo of course, the more expensive the design process becomes. Thus efforts are always made to verify a given level of abstraction by means of high level models before going down to a more detailed view of the same circuit. Modelling, simulation, yield estimation and optimization are thus integral parts of the design cycle.

Of course, once we get to the fabrication stage, we are no longer dealing with models but actual silicon which brings with it a whole new set of problems, as we shall outline shortly.

2.1.2 Modelling and simulation

A key part of the above design flow is the ability to model and simulate the circuit at different levels of abstraction. At the lowest level, we have a transistor level netlist which can be simulated via SPICE [27]. With the rise of available computational power and the development of efficient numerical algorithms, the size of circuits we can deal with using SPICE simulators has grown as well. However, it still may not be possible to simulate transistor level dynamics for large subsystems.

Having a higher level abstraction model thus becomes indispensable for functional verification. Optimally, such an abstraction model should lend itself to co-simulated with digital components as well. With the development of VHDL-AMS [28] and Verilog-A/MS [29], it is indeed possible to build such models [30]. However, ensuring that these models match the circuit response remains a difficult problem, motivating the equivalence checking problem we described in Section 1.3 and that we shall

attempt to address in Section 5. Once again, we assume that both the circuit design and the model generation process are designer driven and develop tools to assist with the same.

Abstraction however, comes at a cost as well. Critical interaction effects between adjoining blocks may be skipped unless the correct parameters are excited during system-level simulation. While this can be done by ensuring that the abstraction model captures lower level parameters as well, doing so for all parameters causes the dimensionality of the abstraction model to grow as well.

2.1.3 Yield estimation and optimization

Assuming we have suitably accurate models at every step of the design cycle, we can solve the real verification problem viz. determine whether the circuit meets specifications under all operating conditions.

As process variability continues to affect circuit performance [17, 18, 19, 20], it is nearly impossible to make sure that the circuit will meet specifications under all conditions. Instead, we are satisfied as long as a certain proportion of circuits meet specifications. This proportion is called the yield and we wish to optimize circuits so that the yield is as close to 100% as possible. Thus the verification problem once it reaches circuit level dynamics is no longer deterministic but stochastic and must be addressed using stochastic techniques [9, 10].

A particularly interesting subproblem that has received a lot of attention is SRAM yield estimation. Given the large number of times an SRAM bit cell is going to be repeated on a die and the fact that they are typically designed using minimum size devices, it is critical to evaluate the failure rate of SRAM cells both efficiently and accurately. In fact, the very design of an SRAM array can be statistical in nature to maximize the yield [31]. This has spurned a lot of work in the area of SRAM yield

estimation [5, 6, 7, 8].

As described in Section 1.1 however, verifying larger circuits remains difficult due to the large number of parameters interacting in a complex non-linear manner. This is what we attempt to address by relaxing the required accuracy in Section 3.

Of course, simply predicting how often a circuit will fail is not sufficient. There must be some mechanism to assist the designer with optimizing the yield as well. Knowing what parameters to optimize and what are the critical interactions between different transistors may not be immediately obvious, especially for large circuits.

While this area has been extensively researched as well, traditionally used diagnosis aids such as [15, 16] are inherently based on sensitivity analysis and work best for linear circuits while most practical circuits are highly non-linear. Other techniques such as [12, 13, 14] may be able to deal with complex non-linear interactions by means of classifying circuit performance or generating Pareto fronts but such information is hard to visually interpret in a high-dimensional space.

Identifying interactions between operating conditions, input signals, process parameters etc. that may result in out-of-specification failures and returning it in a fashion that intuitively makes sense to the designer thus remains a complex problem as described in Section 1.2. This is what we address via the failure region based approach in Section 4.

2.1.4 Analog and mixed-signal testing

Once an IC is fabricated, it must be tested to determine whether it fulfills its intended purpose or not. Although analog circuits may constitute a small portion of the total chip area, they constitute a significant portion of the overall test cost [1]. This is mainly because of two factors : (a) the pin count is limited, meaning that only a limited number of nodes can be accessed from the outside; and (b) the

presence of statistical process parameter fluctuations means that there is a spread of responses around the expected output.

The first factor has been largely offset by the development of Design for Testability (DfT) measures [32, 33, 34, 35], although knowing what to control and observe is starting to become an issue due to the ever increasing design complexity. To deal with the second factor, failures in mixed-signal circuits have been classified into two major types with most techniques targeted towards one factor or the other. The two types are : (a) catastrophic faults or hard faults arising from change in topology (short-circuit and open-circuit failures), and (b) parametric faults or soft faults caused by variations in circuit parameters.

The work we detail in Section 6 and Section 7 notably deals with only the second type of faults i.e. parametric faults. We assume that catastrophic faults can be caught and filtered out using other techniques such as [36, 37, 38, 39, 40].

Even after the DfT structures have been designed and implemented, and the catastrophic faults have been filtered out, selecting the right test points to excite parametric faults remains a challenging problem. Existing approaches [41, 42, 43, 44] are again mostly based on a linear circuit assumption, whereas most practical circuits are highly non-linear.

More importantly, just like in the pre-silicon domain, we don't just want to detect a fault but also diagnose what the root cause(s) of the failures may be. There has already been a large volume of work in fault isolation and diagnosis. Existing approaches towards the same can be divided into two major classes : (a) Simulation before test (SBT), and (b) Simulation after test (SAT). In SBT techniques, the fault-free and faulty circuits are first simulated to build a fault-dictionary. The response of the circuit under test is then compared to this dictionary to find the closest match. In SAT techniques, the circuit response is measured first. The circuit parameters are

then reconstructed from the measurement results.

While SBT techniques prove very useful for catastrophic faults [36, 37, 38, 39, 40], ensuring fault coverage for parametric faults proves to be difficult. Parametric faults are thus usually dealt with using SAT techniques or a mixture of SAT and SBT techniques [45, 46, 47, 48, 49].

It is to be noted that building an exhaustive "fault dictionary" for SBT techniques can be difficult for either type of fault if the circuit size is large. This problem is usually mitigated by using a divide-and-conquer approach. However, having to control and observe many small blocks adds to the DfT cost as well.

SAT techniques on the other hand, are not without their own drawbacks. Depending upon the controllability and observability, it may not always be possible to uniquely solve the value of each element from a given measurement set. Typically, this means that we are forced to introduce additional DfT structures and/or estimate the most likely element values from using optimization techniques on obtained measurements.

With rising design complexity however, these DfT structures are getting more and more expensive and it may not always be straightforward to guarantee that we do have the required controllability or observability. Simultaneously, hardware level security is beginning to become an important concern in many state-of-the-art designs. The more DfT structures we create, the more loopholes we leave into the internal state of the hardware itself.

Thus, the need of the hour is to be able to both excite and diagnose parametric faults while keeping the DfT cost low. While we may not be able to get to exact parameter values, based on the pre-silicon expectations and post-silicon data, we should be able to at least draw statistically significant inferences as to what failure mechanisms may have become important in real silicon. This is the problem that

we introduced in Section 1.4 and Section 1.5 and that we at least partly solve over Section 6 and Section 7. While similar to SBT techniques, the fault dictionary in this case does not have to include every point in the parameter space but simply broad regions of the parameter space that we extract during pre-silicon diagnosis.

2.2 Mixed signal circuit as a statistical model

The rest of this dissertation, we shall be revisiting the concept of "probability" (be it of failure / excitation / coverage etc.) again and again. However, the term "probability" implies that there is an inherent distribution of some sort. If we are to look at a mixed signal circuit as a black box such as in Fig. 2.3, it immediately becomes obvious how the process parameters could form a distribution. The mixed signal circuit is effectively a parameterized model which perturbs the input signals to give rise to output signals depending upon the value of various process parameters.

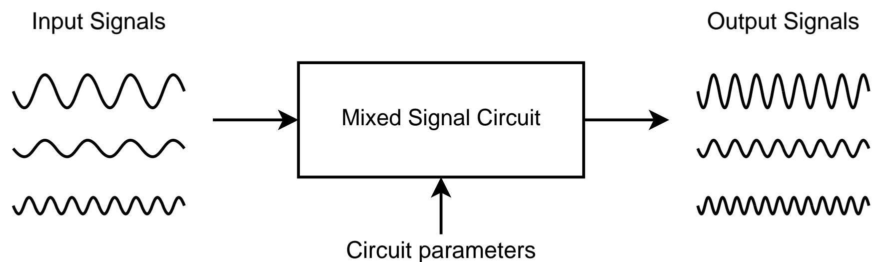


Figure 2.3: A black box view of a mixed-signal circuit. The circuit can be thought of as a parametric model where the parameters in question refer to only process parameters at this point. This picture will be further enhanced in Fig. 2.4.

However, as described in Section 1, we are not just interested in ensuring correct behavior for a fixed set of input signals, voltage etc., but over all such combinations. Thus the real picture looks more like Fig. 2.4 where we don't deal with process

parameters alone. The inputs to the circuit (both supply and signal voltages) are parameterized as well. Depending upon the complexity of the signal in question, the parameters may simply be a voltage (as in the case of a supply voltage), or may be a lot more complex and include parameters such as frequency, duty cycle, rise time and fall time to describe a single signal (such as for a clock).

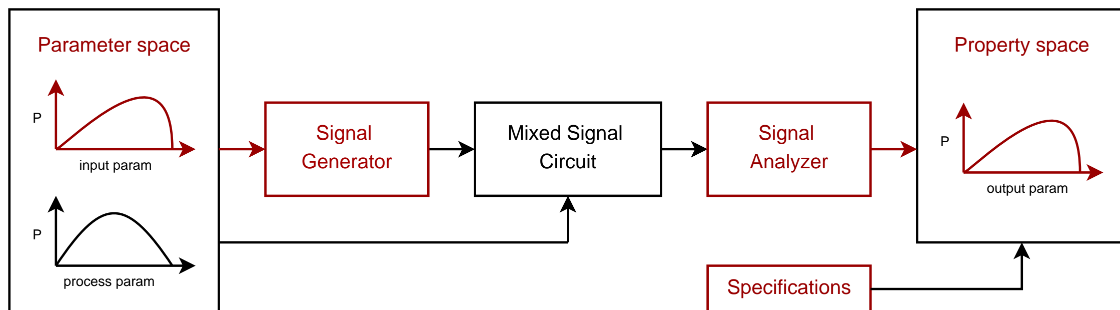


Figure 2.4: A completely parameterized view of a mixed-signal circuit. Both the inputs to the circuit and the model itself have now been parametrized. The combined parameter space can now be thought to contain input signal parameters, process parameters, sources of noise etc.

Note that parameterization does not preclude us from dealing with deterministic behavior either. A fixed value for a given parameter can simply be captured by means of a Dirac delta function at the point where the fixed value lies. The same concept can be generalized to deal with other fixed inputs as well. For example, if we do not wish to parameterize a given signal but instead obtain the signal (via simulation or measurement) from an upstream block, we can simply simulate this upstream block. The parameter in this case becomes a pointer to this signal and will again be a Dirac delta function if we are interested in only one signal. To generalize this concept one step further to deal with a non-deterministic upstream block, we could generate a

list of signals by simulating the upstream block across multiple conditions and use the probability of these signals being generated to form a multinomial distribution across all such pointers.

Parameterizing the output signals when going from Fig. 2.3 to Fig. 2.4, is even easier to understand. While waveforms can be easily visualized once they are simulated, it is very hard to do a waveform-to-waveform comparison. More importantly, the waveform needs to be specified in a language that humans will be able to read, write and comprehend. This is typically achieved by parameterization of the output waveform, which we refer to as "properties". These properties may include voltage levels, overshoot, jitter, frequency, setup and hold times etc.

We will take a deeper look at parameters, properties and the probability of failure for both the pre-silicon and post-silicon domains over the rest of this section. The intent of course is to lay a mathematical foundation for future sections.

2.2.1 Parameters, properties and models

Since the rest of this dissertation revisits the concepts of parameters, properties, probabilities and models repeatedly, we define the same in this section. For every point x in the multi-dimensional parameter space X, there is a point y in the property space Y. The circuit is responsible for the mapping from $X \rightarrow Y$. Assuming that X is completely specified (there are no other dimensions that will affect $Y = f(X)$), we can assume a one-to-one mapping between X and Y.

This mapping $Y = f(X)$ can be an arbitrarily complex non-linear function defined by the interaction between multiple transistors and the input signals. This non-linearity can hold true even when dealing with very simple parameter and property spaces (for example where both the parameters and the properties refer to DC voltages at some node).

The dimensions of X include any parameter that affects the circuit response - be it input signal parameters or design uncertainties such as process variations or noise. The distribution of X, $p(X)$ may be a function of both measurement data (e.g., characterized process variations) and a designer's belief about a system (e.g., what inputs to expect and how often).

It is to be noted that the term "circuit" is in essence a model $f(X)$. Depending upon what level of abstraction we are looking at, $f(X)$ may actually correspond to many different models. In fact, the fabricated circuit can be considered a model as well. However, it is important to note that both the distribution of the parameter space $p(X)$ and the mapping from $X \rightarrow Y$ may have been altered by the actual fabrication process. This is unlike the pre-silicon domain where we can assume just $f(X)$ is changing since $p(X)$ is an assumption anyway.

Also, given the complexity of $f(X)$, simulating the model in the pre-silicon domain or making measurements in the post-silicon domain will always have some amount of noise associated with it such that the observed $Y = f(X) + \epsilon$. While we briefly allude to this in Section 3, for most of the dissertation, we assume ϵ is negligible since the methods we describe are inexact anyway.

2.2.2 Probability of failure

Now, the circuit itself is designed to approximate some ideal function $Y = \bar{f}(X)$ described by a set of specifications. Thus the circuit response $f(x)$ may not always be the same as $\bar{f}(x)$. Failure of the circuit to meet one or more specifications over the entire anticipated range of X gives rise to out-of-specification failures.

Sampling X using the parameter distribution $p(X)$ and mapping $X \rightarrow Y$ via simulation, means empirically observing the desired property distribution $p(Y)$. Computing the proportion of observations failing to meet specifications then gives us an

empirical estimate of the failure probability. This forms the foundation for standard Monte Carlo analysis [9, 10] and is one of the most common sources of pre-silicon data.

However, standard Monte Carlo forces us to sample extensively around the likely case especially in the presence of tailed distributions. Thus, “adaptive sampling” schemes [5, 6, 7, 8] where the sampling distribution focuses on the failure regions, are often used in practice. Obtaining the final failure estimate $P(F)$ from observations $\left\{x^{(1)} \ldots x^{(N)}\right\}$ though, requires knowledge of both the parameter space and sampling distribution. Irrespective of the exact technique used,

$$P(F) = \frac{1}{N} \sum_{i \in 1}^{N} \frac{P\left(F|x^{(i)}\right) p\left(x^{(i)}\right)}{q\left(x^{(i)}\right)} \tag{2.1}$$

where for a given point $x \in X$, $p(x)$ is its probability in the parameter space, $q(x)$ its probability in the sampling distribution and $P(F|x)$ is the probability of observing a failure at that point (0 or 1 if $x \in$ simulated data).

For most adaptive sampling schemes, $q(x)$ is either highest in the failure region [5] or at the failure boundaries [8]. Thus, data sampled using $q(x)$ can be directly used to identify failure regions. $p(x)$ is still required to compute the importance of a given region.

Similar observations can be made regarding techniques aimed at generating Pareto fronts [12, 13] or classifying circuit performance [14]. In fact, equation (2.1) holds irrespective of how the circuit was sampled as long as sufficient coverage of the failure regions can be guaranteed. We use either (a) standard Monte Carlo data [9] or (b) data sampled adaptively (as described in Section 3) for the rest of this dissertation.

2.2.3 Controllability and observability

Everything we have discussed so far holds true in both the pre-silicon and post-silicon domains. However, as we alluded to in Section 2.1.4, limited pin counts and the cost of adding DfT structures means that we may not have access to X or Y in its entirety. We thus additionally need to take the dual concepts of controllability and observability into account.

Every point $x \in X$ is a multi-dimensional vector of the form $x_1 \ldots x_m$. Of these, $k < m$ parameters are going to be controllable in the post-silicon domain. As a result, we can partition $x \in X$ into two components: $c = \{x_1 \ldots x_k\}$ and $\bar{c} = \{x_{k+1} \ldots x_m\}$. The dimensions of c correspond to the controllable parameters such as input signal parameters, supply voltage, operating temperature etc. The dimensions of $\bar{c}$ refer to design uncertainties such as process variations, sources of noise or even input signal parameters such as input jitter that we may not be able to fully control on actual silicon. The set of all legal values of $c \in C$ can be considered as the controllable space. We thus use it for test selection in Section 6.

Seen from another point of view, every point $x \in X$ maps to another point $y \in Y$. If y is another multi-dimensional vector of the form $y_1 \ldots y_q$, the combination of the two $x \cup y \in X \bowtie Y$ encompasses all information about a particular test instance of the circuit. Since not all dimensions of x are observable, we can once again partition the joint vector $x \cup y$ into two components: $o = \{x_1 \ldots x_l, y_1 \ldots y_p\}$ and $\bar{o} = \{x_{l+1} \ldots x_m, y_{p+1} \ldots y_q\}$. Since the output properties are always observable (unless we decide not to measure them), we assume $p = q$ for this work. Thus the only parameters which are unobservable are design uncertainties which have not been characterized using any specialized techniques. Like in the controllable space, the set of all legal values of $o \in O$ can be considered as the observable space which will

be used for post-silicon debug in Section 7.

It is of interest to note that values that are controllable are by definition observable. Thus $c \subseteq o$. This means that we can extract relevant points in C from samples in O, irrespective of how the test points were selected. This implies that the post-silicon debug contributions listed in Section 7 can be used independently of the test selection methodology in Section 6 as long as the probability of observing failures is sufficiently high. Thus the shifts can be identified even from data generated using alternate post-silicon flows.

2.3 Objective of this dissertation

The objective of this dissertation is not to address every problem that we have detailed above. Instead, it is simply to address gaps we described in Section 1 where exercising the accuracy vs. turn-around-time trade-off can benefit scalability. This will hold true, be it in the area of failure probability estimation, diagnosis, test or debug. The key observation in relaxing the accuracy constraint is that we also wish to know the level of confidence. This could be by means of a failure probability interval when estimating the failure probability, or by some coverage metric when doing diagnosis or test. Since we have outlined the high level goals in Section 1 anyway, we do not reiterate the same here.

3. FAILURE PROBABILITY ESTIMATION *

Given a mapping from a parameter space X to a property space Y, our goal is to obtain the probability that the observed properties violate the circuit specifications (henceforth referred to as failure probability). Since it is entirely possible that these failures manifest only when the circuit is biased in a certain manner, the dimensions of X must include operating conditions such as input signal parameters in addition to sources of design uncertainty. The circuit is responsible for the non-linear mapping from $X \rightarrow Y$ where Y is a function of output signal properties.

Since violations can be directly identified in the property space, we define an identification function I that classifies a point $y \in Y$ as pass or fail

$$I(y) = \begin{cases} 0, & \text{if } spec \text{ is met} \\ 1, & \text{if } spec \text{ is not met} \end{cases} \quad (3.1)$$

The probability of failure can then be obtained as

$$P(F) = \int_{y \in Y} I(y)p(y)dy \quad (3.2)$$

The above equation corresponds to the area under the property distribution that results in a failure in Fig. 3.1.

Unfortunately, the property distribution $p(Y)$ is unknown, forcing us to sample

---

* Parts of this section have been reprinted with permission from "Approximate Property Checking of Mixed-Signal Circuits," by P. Mukherjee, C. Amin & P. Li, in *Proceedings of the The 51st Annual Design Automation Conference*, 2014, © 2014 ACM Inc.

http://doi.acm.org/10.1145/2593069.2593091

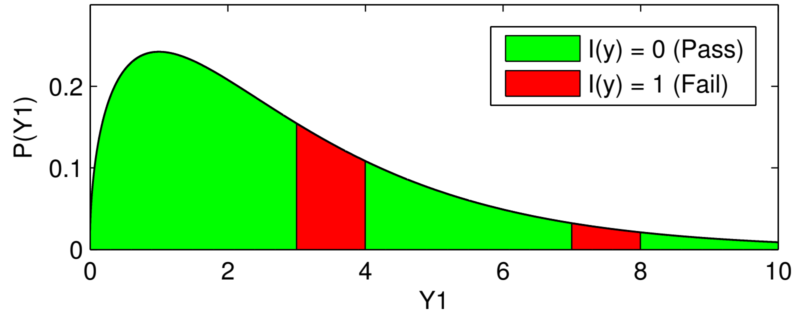


Figure 3.1: Failure regions in property distribution. The area under the probability distribution curve marked in red gives us the probability of failure that we wish to estimate.

in X via $p(X)$ and map $X \rightarrow Y$ via simulations. $P(F)$ can then be empirically ascertained by computing the proportion of samples which cause a property to be violated (standard Monte Carlo [9]).

Sampling based on $p(X)$ however, can cause redundant effort around the most likely case, especially in the presence of tailed distributions. Since our objective is simply determining the failure probability, concentrating on the failure regions proves to be more useful. This is the motivation behind adaptive sampling techniques [5, 6, 7, 8].

Our objective is to similarly leverage knowledge from arbitrary distributions to obtain an approximate estimate. Decoupling the sampling distribution from the parameter distribution however, requires some kind of knowledge representation scheme. In our case, we first train a regression model using SPICE data. The regression model's response to the parameter distribution can then be compared against specifications to predict whether the circuit will meet specifications at a given point $x \in X$ or not. It thus essentially forms a failure model. The remaining elements in Fig. 3.2 either estimate or compensate for the error in this process. The predictions

of course give us the estimated failure probability as per equation (3.2). The error estimates are incorporated into the failure interval giving us a measure of confidence about the predicted failure probability as well. We now take a detailed look at the nature of this error.

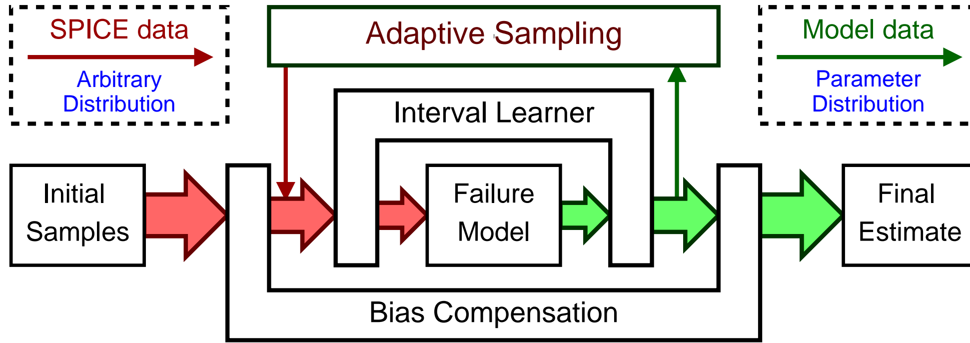


Figure 3.2: High level overview of proposed approach. The interval learner is described in Section 3.1, the bias compensation in Section 3.2 and the adaptive sampling built around it in Section 3.3.

If $y = f(x) + \epsilon$, describes the mapping $X \to Y$ that we wish to capture, the goal of any surrogate model is to find an approximation $\hat{Y} = \hat{f}(X)$. Here $\epsilon \sim N(0, \sigma_\epsilon)$ corresponds to measurement error or random noise in the system and hence, cannot be reduced further. The expected squared prediction error in $\hat{f}(x)$ can now be expressed as

$$\begin{aligned}
\varepsilon(x) &= E\left[\left(y - \hat{f}(x)\right)^2\right] \\
&= \left(E\left[\hat{f}(x)\right] - f(x)\right)^2 + E\left[\hat{f}(x) - E\left[\hat{f}(x)\right]\right]^2 + \sigma_\epsilon^2 \\
&= \text{Bias}^2 + \text{Variance} + \text{Irreducible error}
\end{aligned} \tag{3.3}$$

There is always a trade-off between the above bias and variance. To create estimates that account for both, we leverage the fact that the variance depends on $\hat{f}$ alone and not the actual mapping of $X \rightarrow Y$. We thus propose

1. A (biased) “interval learner” (Section 3.1) that returns $\hat{P}(F)$ and an interval $[minP, maxP]$ enclosing the maximum deviation in $\hat{P}(F)$ due to the model variance.

2. A bias compensation step using an ensemble of such learners to deal with the model bias (Section 3.2).

The final result is an approximation $\bar{P}(F)$ of $P(F)$ and an interval $\left[m\bar{i}nP, m\bar{a}xP\right]$ that captures the expected deviation in $\bar{P}(F)$. This knowledge representation scheme can now be directly used to perform adaptive sampling as described in Section 3.3.

3.1 Interval learner

To begin, let us assume that we obtain $p\left(y|x\right)$ instead of a single prediction $y = \hat{f}(x)$. For demonstrative purposes, $p\left(y|x\right) \sim N\left(E\left[\hat{f}(x)\right], \sigma_{\hat{f}(x)}\right)$ in Fig. 3.3. Specific implementation issues such as choice of surrogate model (Bayesian Additive Regression Trees [50]) are dealt with in Section 3.1.1.

Since $y = \hat{f}(x)$ can take on multiple values at a single $x \in X$ via $p(y|x)$, the probability of failure $P(F|x)$ at x need not be just 0 or 1 but must instead be obtained as

$$P(F|x) = \int\limits_{y=\hat{f}(x)} I(y)p(y|x)dy \tag{3.4}$$

From the above, we can obtain two metrics

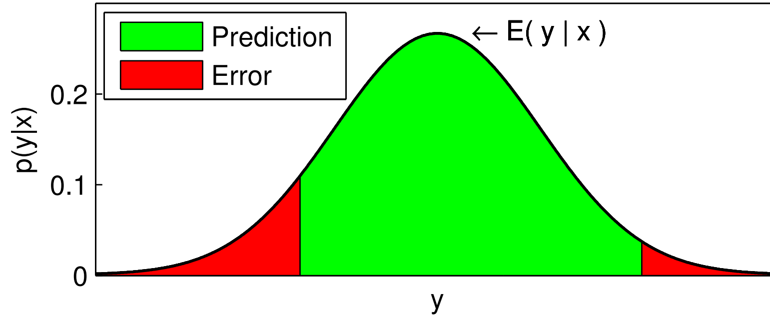


Figure 3.3: Point wise prediction and error. For every point $x \in X$, we obtain a distribution over $y|x$ instead of a single y. Both a pass-fail prediction and probability of misprediction can thus be obtained.

- The prediction

$$\hat{i}(x) \quad = \quad I\left(E\left[y|x\right]\right) \tag{3.5}$$

- Probability of misprediction

$$P(\epsilon|x) \quad = \quad \begin{cases} P(F|x) & \text{if } \hat{i}(x) = 0 \\ 1 - P(F|x) & \text{if } \hat{i}(x) = 1 \end{cases} \tag{3.6}$$

The first metric is the predicted class at the expected value. The second, corresponds to the area under $p(y|x)$ that does not belong to the same class as $\hat{i}(x)$.

Given a one-to-one mapping from $X \to Y$, $P(F)$ can be approximated as follows from equations (3.2) and (3.5)

$$\hat{P}(F) \quad = \quad \int_{x \in X} \hat{i}(x) p(x) dx \tag{3.7}$$

We additionally wish to estimate a conservative interval $[minP, maxP]$ that accounts for the model uncertainty. To do so, we first observe that the total probability of misclassifying points in the parameter space that we have classified as failures ($X_F \subset X|\hat{i}(x) = 1$) is given by

$$P(\epsilon|X_F) = \int_{x\in X|\hat{i}(x)=1} P(\epsilon|x)P(x|X_F)dx \quad (3.8)$$

where $P(x|X_F) = P(X_F|x)p(x) \,/\, P(X_F)$ using Bayes rule.

Since $P(X_F|x) = 1 \quad \forall x \in X_F$, we finally obtain

$$P(\epsilon|X_F) = \int_{x\in X|\hat{i}(x)=1} \frac{P(\epsilon|x)p(x)}{P(X_F)}dx \quad (3.9)$$

If points in X_F are misclassified with probability $P(\epsilon|X_F)$, the minimum bound $minP$ for $\hat{P}(F)$ can be obtained as

$$minP = \hat{P}(F) - P(X_F)P\left(\epsilon|X_F\right) \quad (3.10)$$

By similarly deriving $P(\epsilon|X_P)$ where $X_P \in X|\hat{i}(x) = 0$, we can summarize $[minP, maxP]$ as

$$\begin{aligned} minP &= \hat{P}(F) - \int_{x\in X|\hat{i}(x)=1} P(\epsilon|x)p(x)dx \\ maxP &= \hat{P}(F) + \int_{x\in X|\hat{i}(x)=0} P(\epsilon|x)p(x)dx \end{aligned} \quad (3.11)$$

This deviation is demonstrated in the property space in Fig. 3.4. Since we cannot analytically evaluate equations (3.7) and (3.11), $\hat{P}(F)$, $minP$ and $maxP$ are

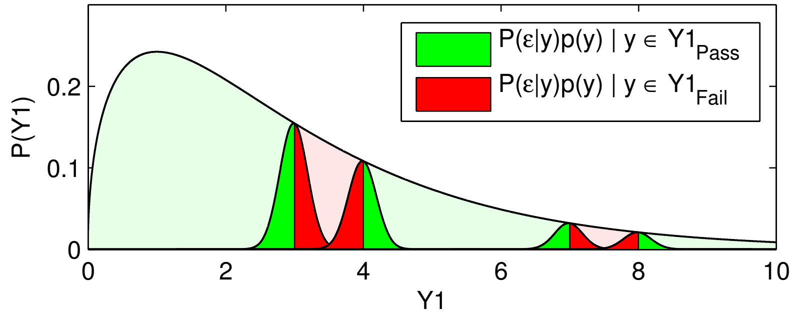


Figure 3.4: Distribution of y from Fig. 3.1 overlaid with probability of misclassification. This probability of misclassification is highest at the failure boundary and tapers off as we move away from it.

estimated via standard Monte Carlo over the failure model. Doing so accurately requires a large number of samples ($\sigma^2 \propto 1/N$). Hence, the cost of evaluating the surrogate model must be low. The bias due to this process can be further reduced by techniques such as Latin Hypercube Sampling [51].

3.1.1 Implementation details

$[minP, maxP]$ is conservative if $P(\epsilon|x)$ is pessimistic on average. This occurs when the model overestimates the error by widening $p(y|x)$ (Fig. 3.3) only in areas where no training samples exist. Ideally, $P(\epsilon|x) = 0 \; \forall x \in$ training dataset. However, this may not be true in practice.

Furthermore, we have glossed over the fact that additional bias may be introduced by the choice of model or the choice of parameters for a given model. Moreover, the representational flexibility of the model may be limited or we may be extremely sensitive to the presence of outliers.

In essence, there is no free lunch and the above framework will always be somewhat heuristic in practice. We minimize this by choosing and tuning the failure

model carefully.

3.1.1.1 Choice of failure model

Having to update $p(y|x)$ over time naturally leads us towards Bayesian inference techniques. Since the mapping between $X \rightarrow Y$ is non-linear, we must further use kernels (which may introduce bias of its own) or a knowledge representation scheme capable of capturing interaction effects naturally. Bayesian trees tend to be suitable in this regard, but perform poorly at capturing additive effects. While efforts to balance the two have been made by using different models at each leaf, a better approach is to simply use a sum-of-trees.

A sum-of-trees model is fundamentally an additive model with multivariate components, each of which can naturally incorporate interaction effects. Specifically, we propose the use of Bayesian Additive Regression Trees (BART) [50] which elaborates the sum-of-trees model by imposing a weak prior that keeps the individual tree effects small. Thus, each regression tree explains a small portion of the outcome that we wish to learn with a certain probability. $p(y|x)$ can be empirically ascertained via Markov Chain Monte Carlo [52], allowing us to estimate $\hat{i}(x)$ and $P(\epsilon|x)$ directly.

3.1.2 Tuning the failure model

While BART is effectively a non-parametric approach, the effect of priors becomes significant when dealing with small datasets (this is true of any Bayesian learning technique). Controlling the number of trees and prior probabilities associated with each tree helps minimize this bias.

While any such technique will be heuristic in nature, we can at least take an informed approach towards developing these heuristics. Specifically, we do the following :

1. Choosing a very restrictive error prior may cause over-fitting and exploration

of irrelevant portions of the parameter space. At the same time, we wish for the training error to be relatively low for the final estimate. As a result, we scale the width of the error prior with respect to the number of samples.

2. Interaction effects are better captured within a tree while additive techniques are better captured by interaction between multiple trees. Given our domain knowledge, we know that failures are usually caused by interaction effects between transistors or by a single transistor in the presence of specific input conditions. The probability that a single parameter will cause failures by itself and/or a large number of parameters (relative to the total number of parameters) will interact to give rise to failures are both equally low. We thus choose the tree priors such that the probability of occurrence of both these extreme events are reduced.

3. Since we have adjusted the priors associated with each tree to prefer longer (than depth 1) trees, we also run into a risk of over-fitting when dealing with limited number of samples. We thus ensure that the number of trees we attempt to fit also grows with the total number of samples such that each tree is explaining at least a few points.

While all the above tuning mechanisms are heuristic in nature, they on average allow our methodology to perform better than what it would have done using the default BART priors. However, a different knowledge representation mechanism targeted towards the specific application of capturing and predicting nonlinear behavior that resembles transistor level response might perhaps be an area of further research.

3.1.3 Enhancing accuracy around sampled points

As mentioned earlier, we wish the failure model to over-approximate the error only in places where no training samples exist. However, optimally $P(\epsilon|x) = 0 \; \forall x \in$ training samples. In other words, the cost of mis-predicting a training sample must be high.

While adjusting the cost function is possible using the BART error prior, it is not the most effective method given that the final spacing between training samples is not previously known. Instead, we increase the cost of misprediction at the sampled points by simply feeding the same data into the model multiple times. This also helps ensure that all sampling points get equal weight and that sampling points arriving later do not overwrite trees built by earlier ones. Since the cost of model construction increases with the number of training points, doing so multiple times can prove to be expensive as well. In addition to this, we run the chance of over-fitting. As a result, the training data is fed back to the model at most one additional time.

3.1.4 Exercising the failure model

The final objective of constructing the failure model is to predict $\hat{P}(F)$, $minP$ and $maxP$ using the actual parameter distribution. Since there is no closed-form expression for the same, they are estimated via standard Monte Carlo over the failure model. As discussed, doing so accurately requires a large number of random samples ($\sigma^2 \propto 1/N$).

In addition, the sampled distribution may not follow the desired sampling distribution (the parameter distribution in this case) leading to additional bias in the estimation process. Fortunately, this is a problem which has been widely researched. Techniques such as Latin Hypercube Sampling (LHS) [51] attempt to ensure that an ensemble of random numbers is representative of the real desired variability.

A square grid containing sample positions is a Latin square if (and only if) there is only one sample in each row and each column of the grid [51]. Generalizing this concept to an arbitrary number of dimensions gives us a Latin hypercube. Each sample in a Latin hypercube is the only one in the axis-aligned hyperplane that contains it. In essence, LHS divides the cumulative probability curve into equal intervals on the cumulative probability scale, and then takes a random value from each interval of the input distribution. As a result, each sample (the data of each simulation) is constrained to match the desired distribution very closely, even when dealing with a modest number of samples.

We thus use LHS to reduce the error arising due to Monte Carlo estimation of $\hat{P}(F)$, $minP$ and $maxP$ from the failure model. The error is further reduced by generating a different sampled distribution for each of the learners we build for bias compensation in Section 3.2.

3.2 Bias compensation

Both $P(F)$ and $[minP, maxP]$ in the previous section are a function of $X \rightarrow \hat{f}(X)$. As a result, they may not accurately reflect the mapping from $X \rightarrow Y$. To draw conclusions about the circuit, we must compensate for this bias.

An ensemble is effectively a collection of a (finite) number of predictors that are trained for the same task. Assuming that we have k predictors $f_k(x)$ attempting to approximate $f(x)$, the weighted ensemble average can be obtained as

$$\bar{f}(x) = \sum_{j=1}^{k} w_k f_k(x) \tag{3.12}$$

where $w_k \geq 0$ and $\sum w_k = 1$.

Given the optimal values of w_k, it has been demonstrated that the quadratic

error of the ensemble estimator is guaranteed to be less than or equal to the average quadratic error of the component estimators [53].

To extend this concept to our failure metrics, we assume we can similarly obtain $\hat{i}_k(x)$ and $P_k(\epsilon|x)$ from k different learners. We can then express the ensemble averaged failure probability $\bar{P}(F)$ as,

$$\begin{aligned} \bar{P}(F) &= \int\limits_{x \in X} \sum_k w_k \hat{i}_k(x) p(x) dx \\ &= \sum_k w_k \int\limits_{x \in X} \hat{i}_k(x) p(x) dx \quad = \quad \sum_k w_k P_k(F) \end{aligned} \tag{3.13}$$

Similarly, the ensemble averaged interval $\left[m\bar{i}nP, m\bar{a}xP\right]$

$$= \left[\sum_k w_k minP_k, \sum_k w_k maxP_k\right] \tag{3.14}$$

To ensure $\left[m\bar{i}nP, m\bar{a}xP\right]$ remains pessimistic, we can minimize $m\bar{i}nP$ and maximize $m\bar{a}xP$ separately. Since both $minP \geq 0$ and $maxP \geq 0$, this occurs when

$$\begin{aligned} w_i = 1, w_j = 0 \quad &| \qquad minP_i < minP_j \quad \forall i \neq j \\ w_u = 1, w_v = 0 \quad &| \qquad maxP_u > maxP_v \quad \forall u \neq v \\ \Rightarrow \quad m\bar{i}nP = min(minP_k) \quad &\& \quad m\bar{a}xP = max(maxP_k) \end{aligned} \tag{3.15}$$

The above framework can compensate for the model bias if certain error "diversity" conditions are met while creating the ensemble learner [54]. To do so, we observe that the bias arises from 3 distinct sources : (1) the sampling distribution used to generate the training data, (2) the sampling method, and (3) the model itself due to model choice, parameter choice etc. We already minimized (3) in Section

3.1.1 and do not address it here. The effect of (1) is discussed further in Section 3.3 leaving us to deal with (2).

K-fold sub-sampling has been previously proposed to deal with the sampling method bias in [53]. To start, assume that the population is divided into k mutually exclusive groups of size $m = n/k$ where n is the total number of SPICE simulated samples. Of these k subsamples, at each step we use $k-1$ subsamples to build a distinct learner $f_k(x)$.

By choosing a specific diversity condition, smarter choices for w_k are also possible. If X_T represents the set of points simulated via SPICE (training set), $P_T(F|x) = I(y|x)$ indicates whether $x \in X_T$ is a failure or not. The generalization error ε_k^G for each learner $f_k(x)$ can then be approximated as

$$\varepsilon_k^G = \|P_T(F|x) - P_k(F|x)\| \qquad \forall x \in X_T \tag{3.16}$$

where $P_k(F|x)$ has been defined in equation (3.4). As per [53], we can then pick w_k such that

$$w_i \varepsilon_i^G = w_j \varepsilon_j^G \qquad \forall i, j \in \{1..k\} \tag{3.17}$$

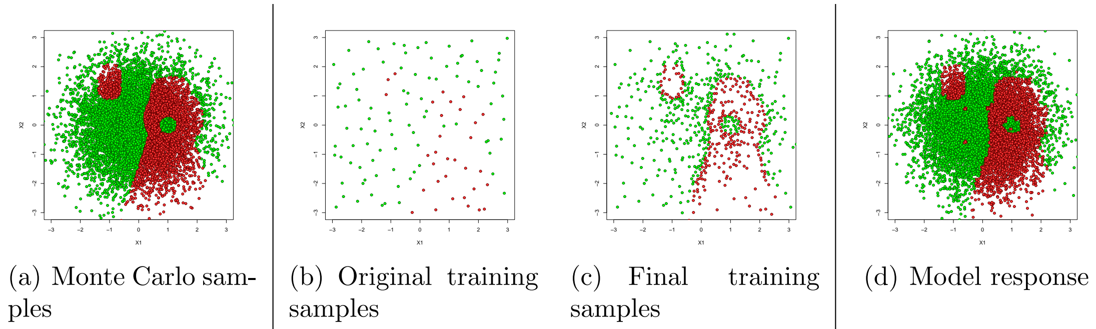
$$\Rightarrow \quad w_k = \frac{\alpha}{\|P_T(F|x) - P_k(F|x)\|} \qquad \forall x \in X_T \tag{3.18}$$

where $\alpha > 0$ is a normalization factor such that $\sum w_k = 1$.

Thus equations (3.13), (3.14) and (3.18) together give us our final estimates of $\bar{P}(F)$ and $\left[m\bar{i}nP, m\bar{a}xP\right]$.

3.3 Adaptive sampling

The bias compensated interval learner we develop is just a tool. Our final objective is to allow the designer to control the accuracy vs. turn-around-time trade-off



(a) Monte Carlo samples (b) Original training samples (c) Final training samples (d) Model response

Figure 3.5: Adaptive sampling demonstration. Fig. 3.5(a) shows the concept we wish to capture and Fig. 3.5(d) how well we managed to do so. As evident from the final set of sampled points in Fig. 3.5(c), sampling concentration is dependent on both proximity to boundary and how important the boundary is.

by adaptively sampling the state space.

The sampled distribution in adaptive sampling is never the same as the parameter distribution. If the sampling distribution is modified to quickly cover the failure regions, both $\bar{P}(F)$ and $\left[\bar{minP}, \bar{maxP}\right]$ may be biased. To minimize this bias, the number of such seed samples is kept to the minimum via the heuristic discussed in Section 3.3.1.1. Points are then adaptively added such that the total uncertainty of the ensemble learner (including the bias) is reduced.

To do so, we first observe the source of the deviation in $[minP, maxP]$ for a given learner by examining equation (3.11) again. Regions in the parameter space where $P(\epsilon|x)p(x)$ is highest contribute more heavily towards the width of this interval. If $p(y|x)$ narrows near the training samples and widens out everywhere else as discussed in Section 3.1.1, the uncertainty for a given learner can be reduced by running further SPICE simulations wherever $P(\epsilon|x)p(x)$ is maximum. By using k-fold sub-sampling similar to Section 3.2 to create these learners additionally addresses the bias since $P_k(\epsilon|x)$ is being reduced for at least one of the k models.

From an implementation standpoint, since we sample the failure model using $p(X)$, the probability of sampling x is already $p(x)$. By simulating the point with the highest $P(\epsilon|x)$ in this dataset, we minimize $\sum P(\epsilon|x)p(x)$. This implies that we do not just sample around the failure boundaries, but the probability of sampling a boundary is proportional to how likely it is. This is shown in Fig. 3.5(c).

The complete framework is detailed in Fig. 3.6. The sub-sampling step differs from Section 3.2 in that we only sub-sample among the N initial samples. The data added adaptively is naturally biased to decrease the model uncertainty and is hence not sub-sampled.

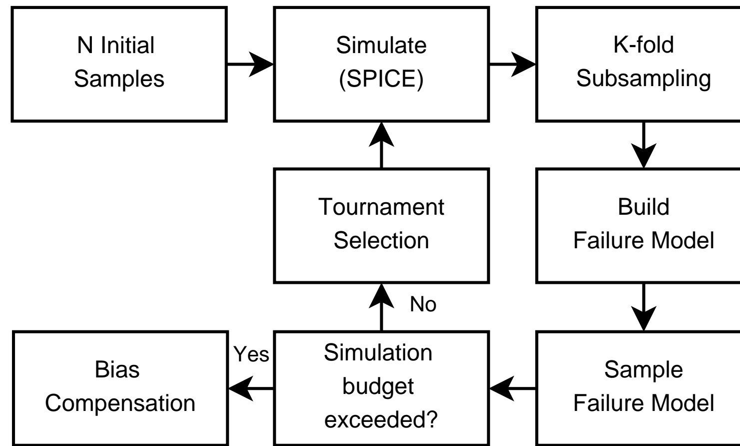


Figure 3.6: Adaptive sampling framework: The failure model and how it may be exercised has already been discussed in Section 3.1. The role of K-fold sub-sampling and bias compensation have also been discussed previously in Section 3.2.

3.3.1 Implementation details

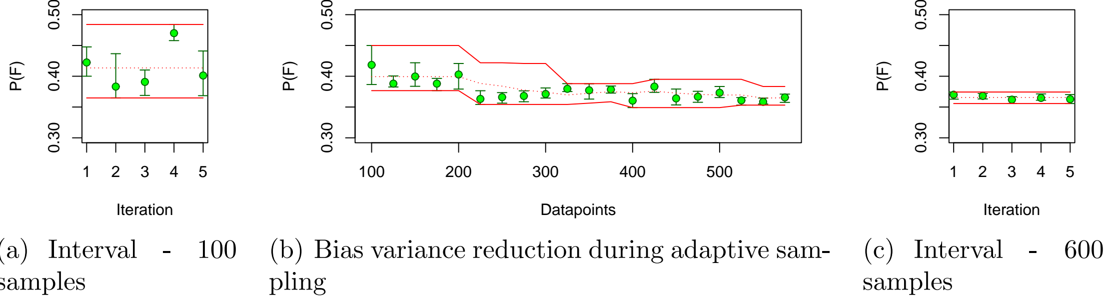
3.3.1.1 Initial samples

In case prior knowledge of likely failure regions exists, this information can be encoded into the sampling distribution. Furthermore, while our algorithm is able to determine information from simulation traces close to the failure boundary, making sure all the failure regions have been sampled provides us with an empirical guarantee of accuracy. However, the number of seed samples must be kept low.

To do so, we use the heuristic detailed in [5] except that we augment it to leverage Latin Hypercube Sampling (LHS) [51]. Since creating or augmenting an LHS set requires a relatively large population size, instead of randomly sampling 1 point at a time until q points in the failure region are found, we iteratively pick p points β times until at least q points in the failure region are found. Thus $N = \beta p$. $\beta = 100$ works quite well for our purposes.

3.3.1.2 Minimum spacing between sampled SPICE data

The irreducible error in equation (3.3) arises from the data generation process viz. SPICE simulation. While this error can be minimized by picking suitable simulation control parameters, no solution is going to be 100% accurate. As such, two points in the parameter space sufficiently close to each other can be assumed to converge to the same solution. Pruning out points which are within a certain threshold ε of each other viz. $||x_i - x_j|| < \varepsilon$ ensures that we do not infinitely refine a given boundary during adaptive sampling. The value of ε is dependent on the nature of the parameter space and optimally requires knowledge of the simulation environment.



(a) Interval - 100 samples

(b) Bias variance reduction during adaptive sampling

(c) Interval - 600 samples

Figure 3.7: Failure probability interval convergence during adaptive sampling. Fig. 3.7(a) shows the original bias compensated interval. Fig. 3.7(b) shows how the interval converges during the course of adaptive sampling. Fig. 3.7(c) shows the final interval.

3.3.1.3 Tournament selection

In practice, SPICE simulations for large circuits must be batched over a cluster to ensure reasonable turn-around-times. q points with high $P(\epsilon|x)$ picked from a single failure model may be in the same vicinity, resulting in a lot of redundant effort. This is similar to getting stuck at a local minima in reward-based selection in genetic algorithms. We thus use deterministic tournament selection [55] to enhance the population diversity. Each tournament is formed by randomly picking w individuals from the Monte Carlo dataset. The individual with the highest $P(\epsilon|x)$ within a given tournament is then chosen unless it has been covered previously. We find $w = 10$ applies the right selection intensity for our purposes.

3.3.1.4 Convergence

Convergence in our framework can be obtained in two ways : (1) We have exhausted our simulation budget (2) We meet some predefined accuracy metric. To define accuracy in in terms of the size of our intervals, we first observe that as long as our model behaves sensibly, for iterations i and $j = i - k$, $(minP_j, maxP_j)$ is going

to be more pessimistic than $(minP_i, maxP_i)$ due to increased amount of available data by iteration i. Thus even if we only build one model per iteration, the following convergence check is going to be conservative.

$$\|max(maxP_{j..i}) - min(minP_{j..i})\| \leq tol \tag{3.19}$$

Where $j = i - k + 1$ and $tol > \alpha$. An example of how this convergence takes place is demonstrated in Fig. 3.7(b).

Note that unless we set tol to an extremely high value, we will run out of simulation budget before achieving this convergence criterion. In fact, this is what happens in the results we present in Section 3.4.

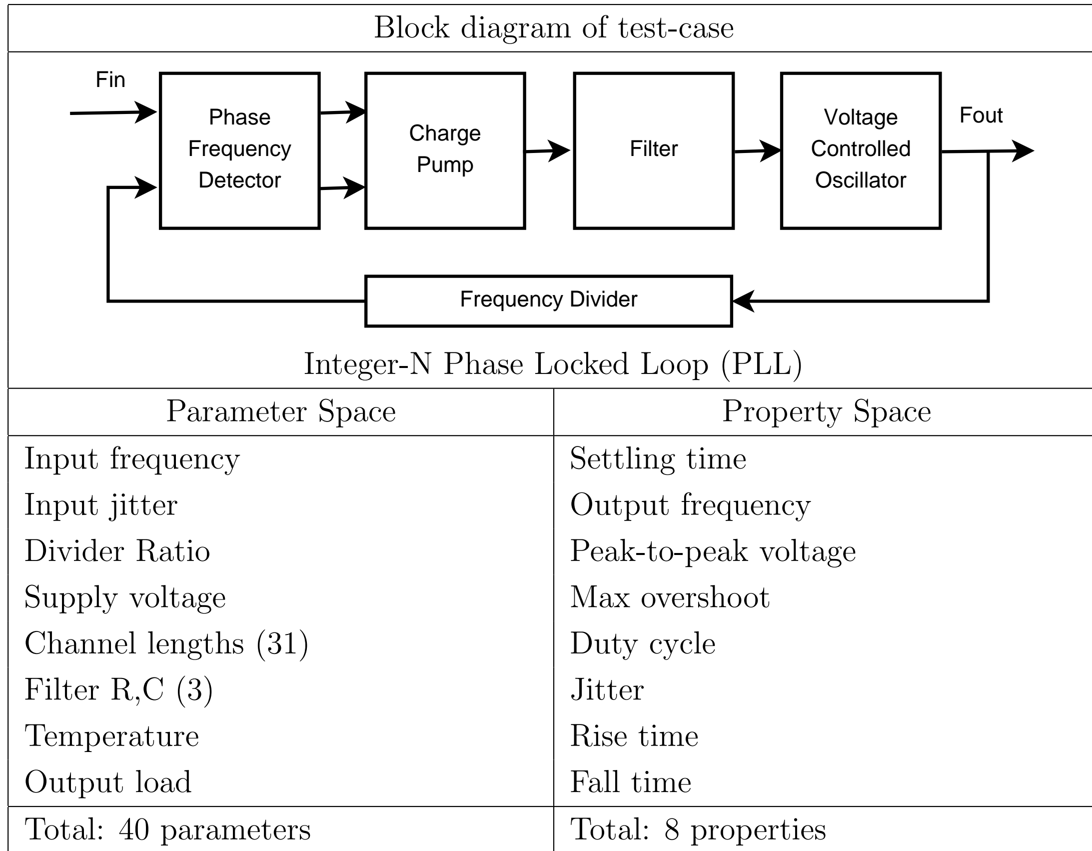
3.4 Results

To present any results, we must first have a test case. Since we use the same test-case through following sections as well, we spend some time describing the same in Section 3.4.1. We then move onto discussing the failure probability interval and how our adaptive sampling performs in Section 3.4.2 and Section 3.4.3 respectively

3.4.1 The test case

While we use a few different circuits to test our proposed methodology, the phase locked loop described in Fig. 3.8 is particularly suited for demonstrative purposes as it is a closed loop feedback system. Failures may be caused (or mitigated due to feedback) by interactions between individual components, which could not have been discovered using a divide-and-conquer approach. This of course has implications on future sections as well which will be discussed in the relevant sections. The versatility of our approach is further demonstrated as the test case has:

- Large system size (spice simulation $\sim$ 30 minutes).



Parameter Space	Property Space
Input frequency	Settling time
Input jitter	Output frequency
Divider Ratio	Peak-to-peak voltage
Supply voltage	Max overshoot
Channel lengths (31)	Duty cycle
Filter R,C (3)	Jitter
Temperature	Rise time
Output load	Fall time
Total: 40 parameters	Total: 8 properties

Figure 3.8: High level block diagram of the PLL test case. Two variants of this circuit, $PLL1$ and $PLL2$ will be used in Sections 3, 4, 6 & 7. $PLL1$ will in turn be compared to a behavioral model with the same block diagram but a smaller set of parameters in Section 5.

- High dimensional parameter and property space (40 parameters & 8 properties).
- Multiple parameter distributions (circuit parameters such as channel lengths and filter R and Cs are Gaussian, everything else is uniform).
- Discrete and continuous parameters (DividerRatio is discrete, everything else: continuous).

- Correlations between parameters (systematic variation in channel lengths. Both inter-die and intra-die variations have been modeled).
- Complex specifications (for example, output frequency = input frequency * divider ratio).

In fact, we shall deal with two variants of the above circuit, $PLL1$ and $PLL2$, to be able to demonstrate conditions under which the failure probability estimation, diagnosis, test and debug methodologies work, or do not work so well as the case may be.

More specifically, $PLL1$ is obtained from $PLL2$ after we use failure probability estimation and diagnosis (Section 4) to perform pre-silicon yield optimization. $PLL1$ thus has a much lower failure probability than $PLL2$ and is more indicative of what we would like the post-silicon yield to be. As a result, the test (Section 6) and debug (Section 7) sections shall focus more heavily on $PLL1$ though we shall provide results for both circuits. How $PLL1$ is constructed from $PLL2$ is something we discuss further in Section 4. Similarly, observations regarding controllability and observability are also deferred until the relevant sections.

3.4.2 Failure probability interval

If $P(F) > 10\%$, Monte Carlo itself converges to a reasonably accurate metric within a few hundred simulations. Thus we are more interested in how our methodology deals with $PLL1$ in Table 3.1 than $PLL2$ in Table 3.2, though of course we provide results for both. The two datasets in the table correspond to:

- MC+LHS: Latin Hypercube sampling [51] over the parameter distribution. $P(F)$ is the Monte Carlo estimate [9, 10] and $minP, maxP$ is the 99% bootstrap [11] confidence interval.

- Adaptive: $P(F)$ and $minP, maxP$ are obtained using the bias compensated interval learner on data sampled adaptively as detailed in Section 3.3.

	PLL1					
	99% Bootstrap CI (MC + LHS)		Bias compensated interval learner			
			MC + LHS		Adaptive	
	$P(F)$	$minP, maxP$	$P(F)$	$minP, maxP$	$P(F)$	$minP, maxP$
200	4.50	1.00, 8.50	0.52	0.34, 2.79	1.99	1.49, 4.21
400	4.75	2.25, 7.75	2.96	2.18, 4.79	3.02	2.36, 4.58
600	4.16	2.16, 6.33	2.84	2.17, 4.33	3.32	2.65, 4.93
800	4.63	2.88, 6.75	3.24	2.50, 4.73	3.52	2.80, 5.07
1000	4.20	2.60, 6.00	3.51	2.76, 4.96	3.56	2.88, 4.95
10000	3.42	2.96, 3.88	← Reference to compare against.			

Table 3.1: Failure rate and interval estimates for PLL1. All probabilities expressed in %. MC+LHS refers to data sampled according to the parameter distribution though using Latin Hypercube Sampling. Adaptive refers to the data sampled adaptively as per our proposed approach.

We first observe from Table 3.1 and Table 3.2 that the intervals returned by our bias compensated interval learner are essentially different from bootstrap [11] confidence intervals. While a detailed discussion of frequentist and Bayesian concepts is outside the scope of this work, our intervals are more similar to credible intervals where: (1) prior information is accounted for, and (2) nuisance parameters are handled differently.

Prior information is already indispensable when using arbitrary sampling distributions. In fact, it is the additional information that allows for quicker convergence to an accurate metric as compared to random sampling. In the absence of adaptive

	PLL2					
	99% Bootstrap CI (MC + LHS)		Bias compensated interval learner			
			MC + LHS		Adaptive	
	$P(F)$	$minP, maxP$	$P(F)$	$minP, maxP$	$P(F)$	$minP, maxP$
200	9.50	4.50, 15.00	12.07	9.39, 17.09	9.79	6.71, 27.71
400	7.00	4.00, 10.00	10.90	8.46, 17.43	7.70	5.41, 23.23
600	6.33	3.83, 9.00	9.54	7.43, 15.39	8.31	6.00, 21.41
800	8.13	5.75, 10.63	9.88	7.80, 15.34	7.81	5.95, 18.26
1000	8.70	6.50, 11.10	9.64	7.45, 15.43	8.00	6.38, 18.65
10000	8.79	8.08, 9.52	← Reference to compare against.			

Table 3.2: Failure rate and interval estimates for PLL2. All probabilities expressed in %. MC+LHS refers to data sampled according to the parameter distribution though using Latin Hypercube Sampling. Adaptive refers to the data sampled adaptively as per our proposed approach.

sampling, the data redundantly captures the prior information and our estimates are no better (or no worse) that standard Monte Carlo estimation.

Prior information also means that the deviation of $minP$, $maxP$ around $\bar{P}(F)$ better captures the true uncertainty about our estimate. If $maxP$ is further from $\bar{P}(F)$ than $minP$, $P(\epsilon|X_P)P(X_P) > P(\epsilon|X_F)P(X_F)$. This means that the uncertainty associated with the pass region is higher than the failure region. No such information can be gleaned from the equi-tailed bootstrap confidence intervals as they rely exclusively on the sampled data.

However, the above benefits come at a cost. While the nuisance parameters in Monte Carlo estimation are relatively simple (bias and variance of sampled distribution), a model based approach can be heavily influenced by a lot more factors. While we minimized these effects in Section 3.1.1, no model is perfect and $[m\bar{i}nP, m\bar{a}xP]$ could be pessimistic to account for these effects (PLL2 in Table 3.2). In our case, the spread of $p(y|x)$ returned by BART is overly pessimistic resulting in pessimistic

intervals. Exploring alternate models that better capture $P(\epsilon|x)$ around $\hat{i}(x)$ could cause the size of $[minP, maxP]$ to reduce more rapidly.

3.4.3 Adaptive sampling

While no model is ever perfect, the goal of our proposed methodology was to reduce either the bias or the variance (or both) in equation (3.3) over time. The initial samples for adaptive sampling are intentionally biased to provide for greater coverage of the failure regions. As a result, both $P(F)$ and $[minP, maxP]$ obtained for this minimum set of samples is heavily biased (200 samples in Table 3.1 and Table 3.2).

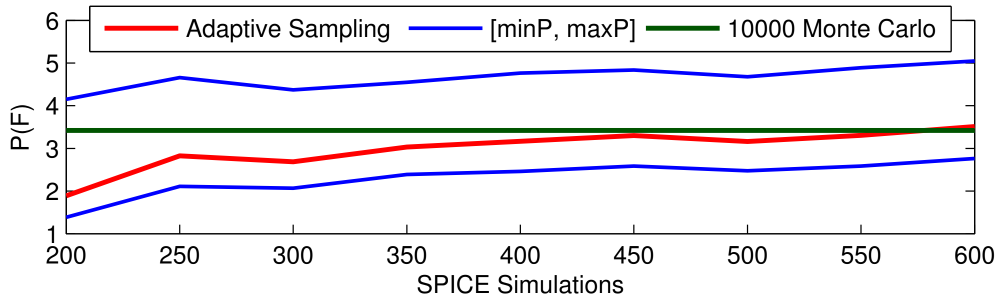


Figure 3.9: PLL1 adaptive sampling demonstration. Since the cost of SPICE simulation is dominant, compute time is reported in terms of number of SPICE simulations required. We are able to get to a stable failure probability estimate within a few hundred simulations.

This bias however, is quickly minimized as demonstrated in Fig. 3.9 to give us an empirical estimate. While either the bias or the variance keeps reducing over time, running further simulations after this initial burn in period (around 600 SPICE simulations) does not provide significant benefits. By using different model parameters, we could have aimed for tighter intervals after the burn-in period. However, our goal

is simply to obtain an approximate failure probability using as few SPICE simulations as possible. 600 SPICE simulations already refers to 300 hours of compute time for this particular test case. As a result, we do not try to improve this estimate any further. As mentioned earlier, exploring alternate failure models might help mitigate this pessimism.

3.5 Summary

As can be seen, the proposed technique fills an important void left by existing work. For one, it allows us to get usable failure metrics for circuits where decomposition into lower dimensional components may not be an option. More importantly, it allows us to obtain reasonable approximations using extremely limited simulation resources when the failure rate is low. Predicting $\left[m\bar{i}nP, m\bar{a}xP\right]$ additionally gives us an empirical estimate of confidence in our prediction. Lastly, the bias compensated interval learner could be extended to other tasks such as classifying circuit performance or system level analysis.

For the results to be meaningful however, significant domain knowledge needs to be leveraged to ensure that our initial random seed has sufficient coverage. Moreover, further work is required to address the model pessimism described earlier if we are to address arbitrary failure rates.

4. FAILURE REGION BASED DIAGNOSIS *

As discussed earlier, the diagnosis problem can be broken down into two:

1. Identify combinations of input signals/design uncertainties that are most likely to violate specifications (Henceforth referred to as failure regions).

2. Identify specific input signal parameters and/or design uncertainties that cause the circuit to enter these failure regions (Henceforth referred to as critical parameters).

A high level overview of the proposed methodology has already been presented in Fig. 1.4. In this section we delve down into the details and discuss the following :

1. How the failure region primitives may be constructed using an ensemble of rule-learning algorithms (Section 4.1).

2. How the primitives may be pruned and aggregated to guarantee maximum coverage in the probabilistic sense (Section 4.2).

3. An information gain based methodology to rank parameters using failure region information (Section 4.3).

Once we have the failure regions and parameter ranking, we show how the failure region information can be used in pre-silicon debug in Section 4.4. Extensions to test-set selection and post-silicon debug will be explored in the following sections.

---

* Parts of this section have been reprinted with permission from "Leveraging Pre-Silicon Data to Diagnose Out-of-Specification Failures in Mixed-Signal Circuits," by P. Mukherjee & P. Li, in *Proceedings of the The 51st Annual Design Automation Conference*, 2014, © 2014 ACM Inc.

http://doi.acm.org/10.1145/2593069.2593154

4.1 Region primitives

Given data that maps the parameter space X to the property space Y, our first goal is to induce simple $if - then$ rules which map region primitives $P_i \subset X$ to out-of-specification failures in Y ($P_i \Rightarrow failure$). The simplest such rule simply bounds each parameter within constant values viz. $P_i = \bigcap_j a_j < x_j < b_j$. From a domain perspective, such rules are not just easy to visualize, they naturally explain interaction effects such as transistor mismatch or input conditions.

Visually, each primitive represents a hypercube in the parameter space as in Fig. 4.1.

While these hypercubes directly solve our problem in a 2-dimensional space, the circuits we wish to diagnose have a lot more parameters. We thus also need to identify the minimum set of features $\{x_1, \ldots, x_m\}$ that is sufficient to explain each failure primitive. The rest of the $N - m$ parameters default to being bounded by their entire legal domain and hence can be considered non-critical for that primitive.

Thus, we are solving two problems simultaneously (1) Clustering the failure points (2) Feature selection within each cluster. Standard rule-induction algorithms such as decision trees [56] and Repeated Incremental Pruning to Produce Error Correction (RIPPER) [57] already do so and hence can be leveraged directly.

4.1.1 Decision trees

A decision tree is essentially a disjunction of conjunctive rules (Fig. 4.2) where each rule is a path traced from the root node to a leaf that identifies a failure case. The majority of tree building algorithms are special cases of (a) Partition your observations recursively using univariate splits (b) Fit a constant model in each cell of the resulting partition. They differ in how exactly the variables are chosen, how the splits are generated and the constant model used in each cell. For example, one

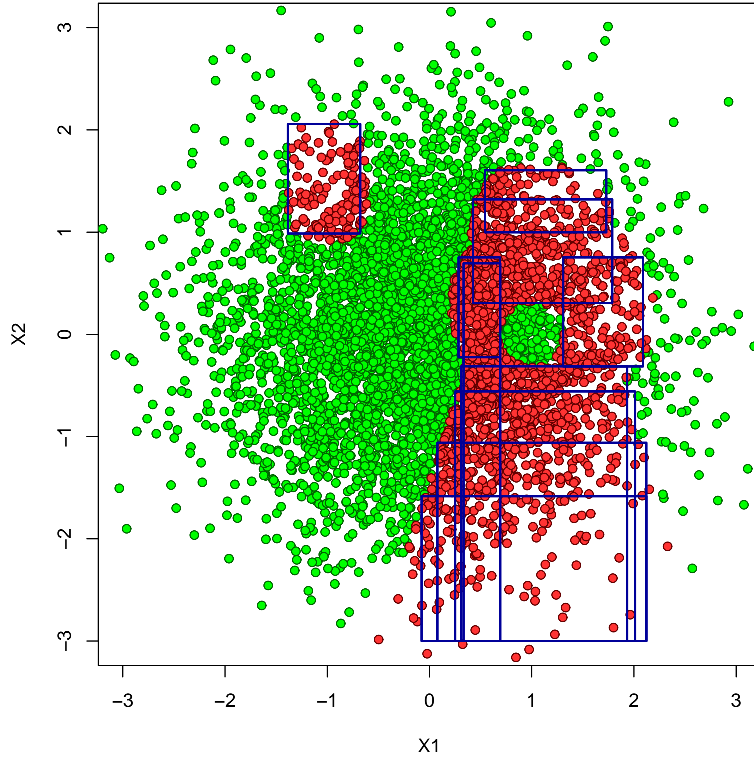


Figure 4.1: Region primitives : two dimensional example. $X1$ and $X2$ are two independent Gaussian parameters. Regions that do not satisfy specifications are marked in red. Each rectangle corresponds to a failure region primitive we wish to generate.

of the most popular decision tree algorithms C5.0 [56], uses entropy as a measure of node impurity that it tries to minimize when generating splits.

$$H(c) = -\sum_{c_i \in (c,\bar{c})} p(c_i) \log_2 p(c_i) \tag{4.1}$$

where c (fail) and $\bar{c}$ (pass) are the two classes of observations in the training dataset indicating whether specifications are met or not. $p(c_i)$ is the proportion of observations belonging to class c_i. This value is highest when $p(c_i) = 0.5$ and lowest when

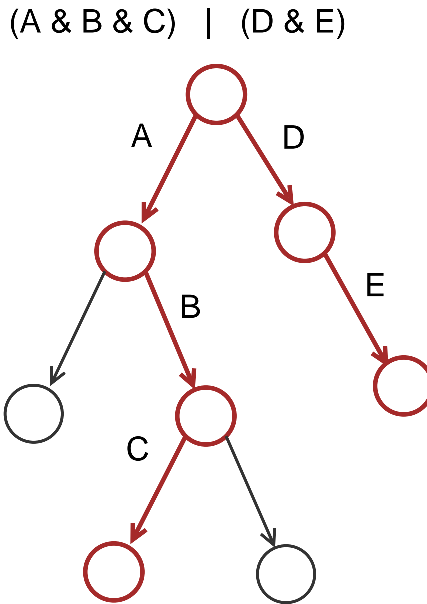


Figure 4.2: Rule learning from decision trees. Each node in the tree refers to a split along a given parameter. A path traced from the root node to a leaf rich in failure cases gives us a failure rule. There may be more than one such leaf as shown above.

$p(c_i) = 1$ or 0. Choosing splits that best decrease entropy thus leads to an increase in homogeneity. Intuitively, at each node of the tree, the parameter that most effectively splits a set of samples into subsets that have more samples belonging to one class than the other is chosen. This is done recursively until no further splits can be identified.

Since we are picking the feature that best separates pass from fail cases at each node, we are implicitly performing feature selection within each primitive as well. In fact, the entropy measure described in equation (4.8) is a popular measure for classifier-independent feature selection as well [58].

Without proper pruning or control over the tree growth however, decision trees tend to over-fit the data making them somewhat poor predictors. In terms of the extracted failure rules, this means that each individual rule may be overly complex,

predicting a very small portion of the data. Both pruning strategies and statistical stopping criterion have been used to solve the over-fitting problem [56].

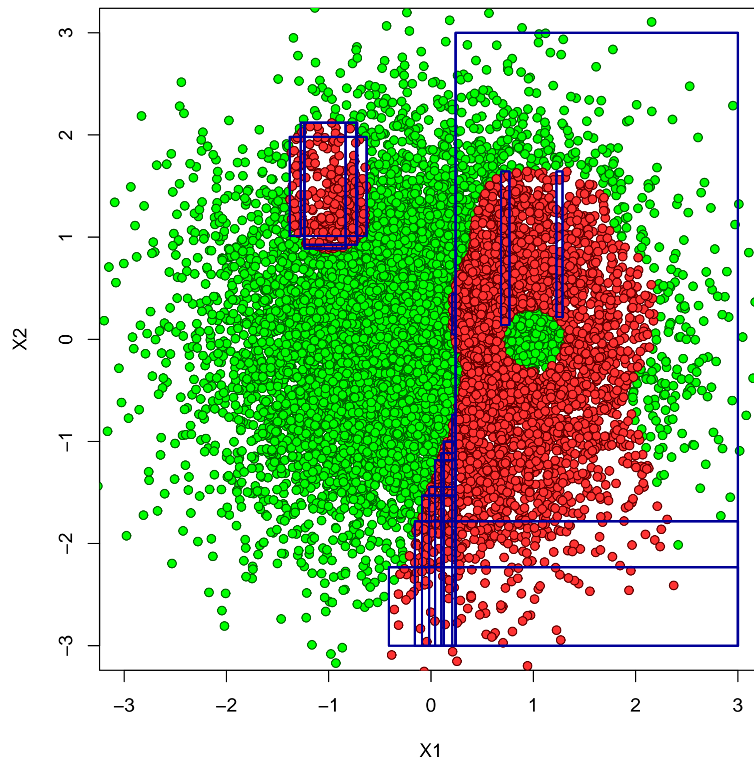


Figure 4.3: Decision tree learning : two dimensional example demonstrating the over-fitting vs. generalization trade-off when performing rule-induction using a single decision tree. All naming conventions are the same as in Fig. 4.1.

However, there is always a trade-off between generalization and over-fitting for any machine learning algorithm and we may end up overcompensating as evidenced by the rules depicted in Fig. 4.3. The rule that covers most of the larger failure region also covers a lot of pass points.

4.1.2 Ensemble learning

One way to get around this is to use multiple trees, each explaining only a subset of the data. Averaging the predictions over all the trees leads to the "sum-of-trees" model demonstrated in Fig. 4.4.

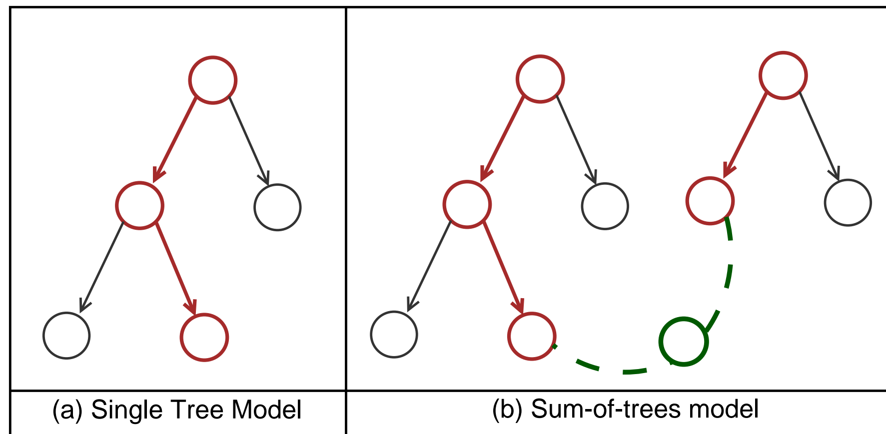


Figure 4.4: Sum of trees model. Solid lines: interaction effects; Dotted lines: additive effects. Paths leading to a failure are marked in red. The green circle represents how complex non-linear failure mechanisms may be explained by combining simpler rules.

From a rule generation perspective, each tree generates rules which may be better at explaining a certain region of the parameter space than others. While this causes an explosion in the number of primitives as evidenced by Fig. 4.5, we assume that this complexity can be handled by the post-processing step (Section 4.2).

The above approach is a specific instance of what is more generally referred to as "ensemble learning". If the post-processing step is classifier independent, we need not limit ourselves to decision trees alone.

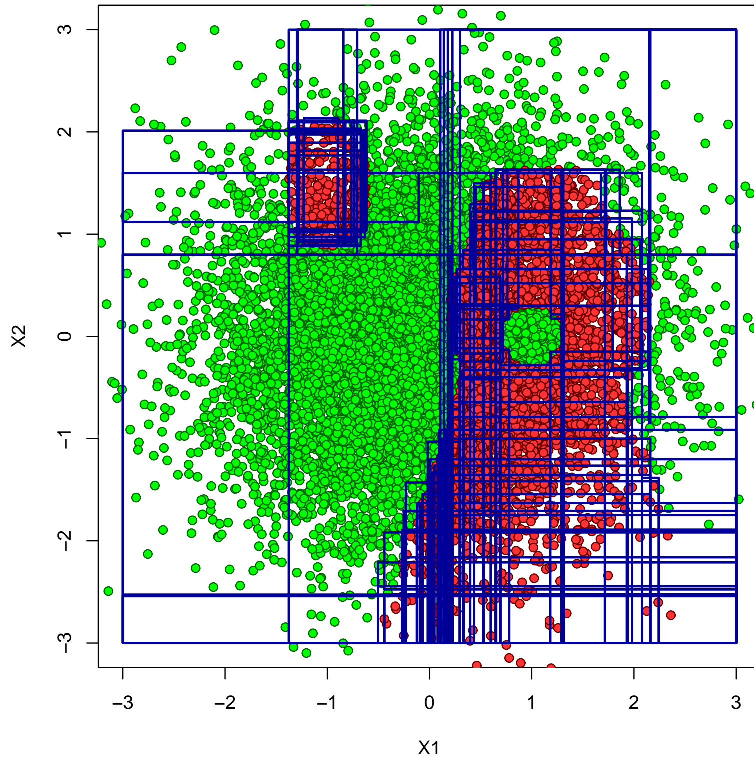


Figure 4.5: Ensemble learning : two dimensional example demonstrating how multiple overlapping rules can guarantee sufficient coverage of the non-linear concept we try to explain. Picking a subset of these rules while ensuring sufficient coverage is the purpose of the post-processing step discussed in Section 4.2.

4.1.3 Implementation details

While intuitively simple, the above methodology is dependent on the quality of the generated rules. If the initial candidate set of rules is bad, no amount of post-processing is going to make the final failure manifold good. As a result, implementation details surrounding choice of algorithms, the data fed into these algorithms and how to deal with failures across multiple properties become important both for interpretability and for the percentage of failures covered by the rules.

4.1.3.1 Algorithmic diversity

Using an ensemble of learners to generate rules means that we have to ensure certain diversity conditions much like we did in Section 3.2. Notably, there are three kinds of diversity we are concerned about : (a) Algorithms used, (b) parameters within these algorithms, and (c) the data fed to each of these algorithms.

We have already discussed how decision trees may be used to generate decision rules in Section 4.1.1. The rules in decision trees are constructed by iteratively separating the data into subsets and then picking all the splits that led to a given leaf node. An alternate approach involves slowly growing regions to cover more points belonging to a given class. A particularly good example of such an algorithm is RIPPER (Repeated Incremental Pruning to Produce Error Reduction) [57].

Depending upon the nature of the sampled data, either of these may produce rules which are better than the other. Both these algorithms however, build rules with constant bounds (in other words, rules of the form $x == a \wedge y == b \Rightarrow fail$). However, models to capture specific interactions (e.g., oblique decision trees [59] to capture linear relationships) may be included at any time. The reason we do not is ease of interpretability as we shall see in Section 4.4.

The second diversity condition has to do with the parameters set within the individual algorithms. However, diversity at this level proves to be less useful for certain restrictions such as the minimum number of points required within a given failure region are already imposed on us given the low number of sampled points. The parameter that shows the largest effect on the quality of rules is the target error rate which we shall discuss in the following section when dealing with rare failures.

The last condition has to do with the data used to actually build the model. While the dataset available for training purposes may be fixed, we can always choose subsets

of this dataset to build rules that better explain a specific portion of the parameter space than others. In practice, boosted decision trees [60] where additional trees are built to better explain data that previous trees misclassified is a good default choice. However, other methods to enforce diversity such as bootstrap aggregation (bagging) [61] and random forests [62] can also be used.

4.1.3.2 Rule induction for rare events

A point that has been skipped over so far but is very critical to consider is sensitivity of various rule-induction algorithms to the relative abundance or rarity of failures. Imagine a case where we have a thousand data points and only one of them is a failure. Even if we generate one rule encapsulating the entire parameter space, the probability of failure within the rule will be 1/1000. In other words the accuracy the rule is 99.9% accurate since the cost associated with mispredicting a pass and a fail are both considered to be the same.

A logical conclusion that can be drawn from the above is that rule-induction algorithms work best when both pass and fail cases are comparable in number. This is usually the case when the data is adaptively sampled. Even if the failure cases happen to be rare for the sampled dataset though, this can be dealt with by assigning a higher cost to mispredicting failure cases. In other words, if the cost associated with mispredicting a pass case is 1, the cost associated with mispredicting a failure case C should be chosen such that,

$$C = \frac{\text{Number of pass cases}}{\text{Number of fail cases}} \tag{4.2}$$

If we use the same example as before, the cost associated with mispredicting the failure point would be now 999 times the cost of mispredicting a pass case. However, this also means that we may include a lot of pass cases in the bounding box we draw

as well for the cost of the one failure point will outweigh the negligible cost of one or more pass points.

If we assume the post-processing step is capable of pruning out the bad rules to keep the good ones, adjusting the cost function can become just another algorithmic diversity criterion. In other words, we run the same algorithm twice : once with the cost function adjustment and once without. We thus generate two sets of rules and let the post-processing step sort it out later.

4.1.3.3 Dealing with multiple properties

None of the arguments presented so far change when dealing with multiple properties if we assume that a violation of any of the properties results in a "failure". For ease of interpretability and to avoid an explosion in the number of regions, this actually ends up being the best approach. Treating each property separately would cause an even larger number of overlapping rules.

4.2 Post processing

In the previous section, we built a lot more P_j than we required without accounting for the predictive accuracy or the difficulty in interpreting a large set of rules. In this section, we keep those P_j which best explain the failures and aggregate them into regions R_i. Each R_i is thus a disjunction of conjunctive rules ($R_i = \bigcup_j P_j$) that explains a distinct (non-overlapping) failure region in the parameter space. The complete set of such regions $R = R_1 \dots R_m$, should explain all the failures discovered via simulation.

Since the distribution of the sampled data does not necessarily follow the parameter distribution (as long as the failure regions have been adequately sampled), we use equation (2.1) as a starting point to develop the post-processing step. If the data does follow the parameter distribution (standard Monte Carlo estimation), $P(F|x)$

is either 0 (pass) or 1(fail), and $p(x)/q(x) = 1$.

To lay the foundation, we first define metrics that give us a notion of importance $P(R_i|F)$ and confidence $P(F|R_i)$ for a given region. Both these values can be easily inferred using Bayesian statistics [63]. Of these, using Bayes rule we get

$$P(R_i|F) \quad = \quad P(F|R_i)P(R_i) \,/\, P(F) \tag{4.3}$$

$P(F)$ can be obtained via equation (2.1). That leaves $P(R_i)$ and $P(F|R_i)$ which we can similarly obtain as

$$P(R_i) \quad = \quad \frac{1}{N} \sum_{x^{(i)} \in R_i} \frac{p\left(x^{(i)}\right)}{q\left(x^{(i)}\right)} \tag{4.4}$$

$$P(F|R_i) \quad = \quad \frac{1}{N} \sum_{x^{(i)} \in R_i} \frac{P\left(F|x^{(i)}\right) p\left(x^{(i)}\right)}{q\left(x^{(i)}\right)} \tag{4.5}$$

To start, assume that we have m regions each of which is composed of a single unique P_i. As a result, the terms P_i and R_i are used interchangeably from this point.

4.2.1 Ranking regions

To rank the regions, we observe that the probability that region R_i was exercised if a failure is observed is given by $P(R_i|F)$. However, if we are to directly use $P(R_i|F)$, it is entirely possible that we end up choosing highly probable regions (large $P(R_i)$) which have very low confidence $P(F|R_i)$. So we propose a custom figure-of-merit,

$$M(R_i) \quad = \quad P(R_i|F)\,(H(\alpha) - H(F|R_i)) \tag{4.6}$$

where H is the entropy function defined in equation (4.8) and α is the minimum allowable $P(F|R_i)$ defined in Section 4.2.2. Thus $(H(\alpha) - H(F|R_i))$ refers to the

information gain of the region R_i as compared to the highest entropy region (least predictive accuracy) that is possible after post-processing. Thus the most important rules with the highest predictive accuracy (richest in failures) will be ranked first.

Since we initialized $R_i = P_i$, we are effectively ranking primitives at this stage. This is in order to speed up the following steps by specifying the order in which primitives must be explored while pruning or aggregating. Lower ranked primitives get pruned / assimilated into higher rank ones.

4.2.2 Pruning regions

The purpose of pruning is two-fold : (a) Enhance the predictive accuracy. (b) Ensure that the final regions generalize effectively. Thus pruning occurs under three conditions:

1. $P(F|R_i) < 0.5$ implies that R_i is richer in pass cases than fail cases. We thus prune whenever $P(F|R_i) < \alpha$ where $\alpha \geq 0.5$.

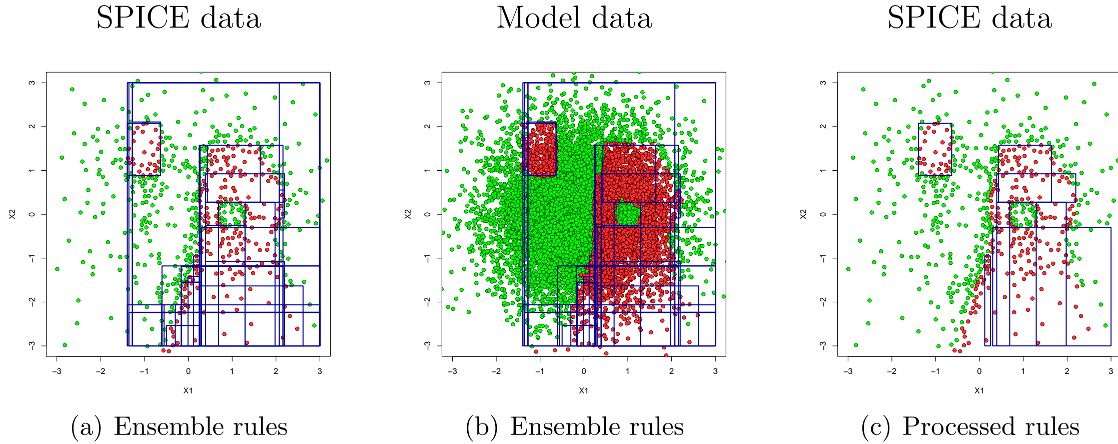
2. $P(R_i|F) \approx 0$ implies that R_i is not important in the first place. We thus prune whenever $P(R_i|F) < \beta$. $\beta = 10^{-2}$ works pretty well.

3. R_i is also unimportant if its failures have already been captured by the remaining regions $(R - R_i)$. We thus prune whenever $P(R_i \wedge \neg \{R - R_i\} |F) < \beta$.

4.2.3 Aggregating regions

Instead of dealing with expensive geometrical computations, we estimate overlaps empirically using the sampled data. We combine two regions R_i and R_j if

$$P(R_i \wedge R_j|F) > \gamma \; P(R_i|F) \quad | \quad i \neq j \tag{4.7}$$



(a) Ensemble rules (b) Ensemble rules (c) Processed rules

Figure 4.6: Failure region discovery on adaptively sampled data. The above figure assumes that we are able to query a failure model as in Section 3. Fig. 4.6(a) is thus used to construct the ensemble and Fig. 4.6(b) to determine the probabilities associated with each region. The final set of processed rules are shown in Fig. 4.6(c).

In other words, a significant number $\gamma = 0.01$ of failures explained by R_i has already been explained by R_j.

The ranking step must be rerun once we are done aggregating. The proposed approach has been demonstrated on data sampled adaptively as in Section 3 in Fig. 4.6 as well. The failure primitives in Fig. 4.6(a) were built using adaptively sampled data. The bias compensated interval learner was then used to obtain pass/fail predictions for the true parameter distribution as visually depicted in Fig. 4.6(b). The importance of using the real parameter distribution of course is that the relative importance of various primitives has now become more clear. The final set of regions obtained via the post-processing steps detailed above is overlaid on the original adaptively sampled data in Fig. 4.6(c).

4.3 Parameter ranking

Each failure region we have described already contains some amount of information regarding a given parameter x_j. If the parameter x_j is critical for region R_i, $[a_{ij}, b_{ij}]$ does not default to $[-\infty, \infty]$ where $a_{ij} \leq x_j \leq b_{ij}$ defines bounds for x_j in R_i. Removing these bounds from region R_i might cause the resulting region $R_{i_{-x_j}}$ to additionally encompass a lot of non-failure points. As a result, the failure information contained in $R_{i_{-x_j}}$ is not as pure.

This concept can be more formally captured by the expected information gain used to generate splits in decision trees [56]. To understand this concept, we must first describe entropy which serves as a measure of impurity within a given region. In our case, this measure reduces to

$$H\left(R_i\right) = -\sum_{f_k \in \left(f, \bar{f}\right)} p\left(f_k | R_i\right) \log_2 p\left(f_k | R_i\right) \tag{4.8}$$

where f (fail) and $\bar{f}$ (pass) are the two classes of observations in the training dataset indicating whether specifications are violated or not. $p\left(f_k\right)$ is the proportion of observations belonging to class f_k. This value of H is highest when $p(f_k|R_i) = 0.5$ and lowest when $p(f_k|R_i) = 1$ or 0. Since we constructed the failure regions using decision trees as well, we already know that if x_j is critical in R_i, the chosen split $a_{ij} \leq x_j \leq b_{ij}$ helped reduce the entropy of $R_{i_{-x_j}}$.

Thus we expect information gain from the introduction of parameter x_j to region $R_{i_{-x_j}}$. This expected information gain

$$IG(R_{i_{-x_j}}, x_j) = H(R_{i_{-x_j}}) - H(R_{i_{-x_j}} | x_j) \tag{4.9}$$

where $H(R_{i_{-x_j}}|x_j)$, the entropy of $R_{i_{-x_j}}$ when split using $a_{ij} \leq x_j \leq b_{ij}$ is given by

$$H(R_{i_{-x_j}}|x_j) = \sum_{v \in vals(x_j)} p(v|R_{i_{-x_j}})H(R_{i_{-x_j}}|v) \tag{4.10}$$

where $vals(x_j)$ can be take one of two values: $[a_{ij}, b_{ij}]$ (bounds for x_j in R_i) or $[-\infty, \infty] - [a_{ij}, b_{ij}]$ (Entire legal range of values excluding the obtained bounds for x_j).

$IG(R_{i_{-x_j}}|x_j)$ also refers to the amount of information lost if parameter x_j is removed from R_i which is what we were after the first place. If we were to obtain $IG(R_{i_{-x_j}}|x_j)\forall i$, we can then obtain $IG(R|x_j)$ over the entire set of regions as

$$IG(R|x_j) \quad = \quad \sum_{R_i \in \{R_1 \ldots R_m\}} p(R_{i_{-x_j}})IG(R_{i_{-x_j}}|x_j) \tag{4.11}$$

Ranking parameters in decreasing order of $IG(R|x_j)$ thus allows us to identify which parameters were most important in describing the set of failure regions. It is to be noted that this manner of ranking is an artifact of the failure rules and not the circuit itself. While information gain is a good measure to decide the relevance of a parameter, it is not perfect. Notably, the information gain may be biased when applied to parameters that can take on a large number of distinct values. In our case, this occurs when the piecewise constant bounds $R_i = \bigcap_j a_{ij} < x_j < b_{ij}$ needs too many regions to capture more complex interactions. Since we are dealing with low failure probabilities, this does not prove to be a problem in our case. Also, should parameter ranking alone be the target, the expected information gain is a generic information-theoretic concept and can be applied to more complex regions discovered via other machine learning algorithms.

4.4 Results

Both the test case (Fig 3.8) and the (adaptively sampled) data we use in this section have been described when discussing failure probability estimation in Section 3. However, the methodology detailed herein is independent of the exact sampling scheme used as long as $P(F|R)$ and $P(R)$ can be estimated with a reasonable degree of accuracy. Thus we could have used Monte Carlo sampled data or data obtained from yield estimation methods here as well.

4.4.1 Failure regions

A sample primitive generated by the proposed methodology is listed in Fig. 4.7. Each primitive lists $P(R)$ and $P(F|R)$ as defined in Section 4.2; the critical parameters and bounding conditions as described in Section 4.1; and the per property failure probabilities $P(F_{property}|R)$. The last metric tells us which of the 8 properties were specifically violated in creating the failure region. Since multiple specifications may be violated simultaneously, $\sum_p P(F_p|R) \geq P(F|R)$.

The failure regions for $PLL1$ and $PLLL2$ are listed in Fig. 4.8 and Fig. 4.9 respectively. These failure regions are used for pre-silicon debug in Section 4.4.3. The naming of both the parameters and properties is obvious from Fig. 3.8 but for the following exceptions : (1) Channel lengths of Charge Pump (LCPxx) & VCO (LVxx) transistors, (2) Filter resistance (R1) and capacitances (Cx). Some of these are shown in Fig. 4.10.

4.4.2 Parameter ranking

The parameter rankings for $PLL1$ and $PLL2$ are listed in Tables 4.1 and 4.2 respectively. The rankings for $PLL2$ are particularly significant since the generated failure regions in Fig. 4.9 are numerous and complex to debug. This diagnosis aid is

```
P(R)   : 0.32%
P(F|R) : 100.00%
Boundaries :
    1.009e+07  <  InputFrequency
    8.400e+01  <  DividerRatio    <= 9.400e+01
    1.141e+00  <  VDD
P(F_property|R) :
       SettlingTime  :  87.50%
    OutputFrequency  :  12.50%
       OutputJitter  :  12.50%
```

Figure 4.7: Sample Rule : $P(R)$ represents the probability that the region is excited based on pre-silicon knowledge. $P(F|R)$ is the probability of failure within that region. The boundaries listed correspond to only the critical parameters. The non-critical parameters default to their entire legal range of values.

just not just desirable but indispensable.

4.4.3 Failure region based pre-silicon diagnosis

Since $PLL1$ is actually obtained from $PLL2$ following the diagnosis stage we describe here, let us first deal with $PLL2$.

Our diagnosis tool detected 15 primitives over 2 regions for the circuit $PLL2$. Instead of going through all the regions, we highlight only the most important trends here:

Input conditions : On extracting only the input conditions listed as important as per the parameter ranking from Table 4.2, we see the following trends in decreasing order of frequency :

1. DividerRatio $\leq$ X

2. Y $<$ InputFrequency $\leq$ Z

Region : 1 $[P(R_1) : 0.60\% \quad P(F|R_1) : 100\%]$

Rule : 1.1 $[P(R_{1.1}) : 0.28\% \quad P(F|R_{1.1}) : 100\%]$

```
2.739e-12  <      C1
1.005e+07  < InputFrequency
```

Rule : 1.2 $[P(R_{1.2}) : 0.28\% \quad P(F|R_{1.2}) : 100\%]$

```
2.806e-12  <      C1
1.811e+01  < Temperature
9.977e+06  < InputFrequency
```

Rule : 1.3 $[P(R_{1.3}) : 0.12\% \quad P(F|R_{1.3}) : 100\%]$

```
2.561e-12  <      C1
1.590e+05  <      R1
1.209e-07  <      LV24
1.246e-07  <      LV44
```

Region : 2 $[P(R_2) : 0.48\% \quad P(F|R_2) : 91.67\%]$

```
            DividerRatio  <= 8.300e+01
1.610e+05  <     R1
7.203e-11  < InputJitter
2.416e-12  <     C1        <= 2.620e-12
```

Region : 3 $[P(R_3) : 0.44\% \quad P(F|R_3) : 100\%]$

```
2.620e-12  <     C1
            DividerRatio  <= 1.090e+02
1.621e+05  <     R1
7.971e-11  < InputJitter
```

Region : 4 $[P(R_4) : 0.12\% \quad P(F|R_4) : 100\%]$

```
2.628e-12  <     C1
                 LV11      <= 1.141e-07
1.160e-07  <     LV23
1.188e-07  <     LV33
```

Figure 4.8: Failure regions : PLL1. The naming of all parameters and properties are self-explanatory but for the following exceptions : (1) Filter R1, Cx as shown in Fig. 4.10, (2) Transistor channel lengths in the charge pump block (LCPxx) and VCO block (LVxx). All other naming conventions are the same as Fig. 4.7.

Parameter	$IG(R|parameter)$
C1	0.0307
DividerRatio	0.0157
R1	0.0129
InputJitter	0.0102
InputFrequency	0.0046

Table 4.1: Top 5 parameters in $PLL1$ ranked based on the importance metric described in Section 4.3.

3. VDD > U

In other words, we expect a higher probability of failure for low Divider Ratios and high Supply Voltage. The trend for Input Frequency is not as clear though most failure regions seem to indicate a high Input Frequency as well.

Design uncertainty : Analyzing design uncertainty variables provides even more clear guidance. Once again, we only analyze the most critical parameters from

Region : 1 $[P(R_1) : 3.46\% \quad P(F|R_1) : 95.66\%]$

Rule : 1.1 $[P(R_{1.1}) : 0.97\% \quad P(F|R_{1.1}) : 97.94\%]$

```
                     DividerRatio    <= 8.800e+01
                     InputFrequency  <= 9.970e+06
5.946e-13  <         C2
                     LCP5            <= 1.251e-07
1.195e+00  <         VDD
1.159e-07  <         LV12
1.145e-07  <         LCP3
1.890e+05  <         R1
```

Rule : 1.2 $[P(R_{1.2}) : 0.24\% \quad P(F|R_{1.2}) : 91.67\%]$

```
                     DividerRatio    <= 9.900e+01
                     InputFrequency  <= 9.985e+06
                     LCP52           <= 1.205e-07
1.146e+00  <         VDD
1.166e-07  <         LV24            <= 1.174e-07
1.167e-07  <         LCP11
                     LCP2            <= 1.248e-07
1.152e-07  <         LV21
```

Rule : 1.3 $[P(R_{1.3}) : 0.24\% \quad P(F|R_{1.3}) : 91.67\%]$

```
                     DividerRatio    <= 1.120e+02
                     InputFrequency  <= 9.977e+06
1.154e+00  <         VDD
1.206e-07  <         LCP3
                     LV24            <= 1.162e-07
5.428e-13  <         C2
                     LCP51           <= 1.235e-07
                     LV13            <= 1.258e-07
                     InputJitter     <= 8.930e-11
```

Rule : 1.4 $[P(R_{1.4}) : 0.41\% \quad P(F|R_{1.4}) : 92.68\%]$

```
                     DividerRatio    <= 8.400e+01
                     InputFrequency  <= 9.972e+06
1.153e-07  <         LCP3
1.157e-07  <         LV2
                     Temperature     <= 5.250e+01
                     LCP1            <= 1.242e-07
1.137e-07  <         LV43
1.138e+00  <         VDD             <= 1.212e+00
1.161e-07  <         LCP5
                     LV12            <= 1.232e-07
```

Rule : 1.5 $[P(R_{1.5}) : 0.15\% \quad P(F|R_{1.5}) : 100\%]$

```
                     DividerRatio    <= 8.200e+01
                     LCP52           <= 1.204e-07
1.939e+05  <         R1
1.178e+00  <         VDD
1.196e-07  <         LCP5
1.195e-07  <         LV22
```

Rule : 1.6 $[P(R_{1.6}) : 0.11\% \quad P(F|R_{1.6}) : 100\%]$

```
                     DividerRatio    <= 8.200e+01
1.178e+00  <         VDD
                     LV5             <= 1.170e-07
                     InputJitter     <= 8.710e-11
1.196e-07  <         LCP5
9.971e+06  <  InputFrequency
```

Rule : 1.7 $[P(R_{1.7}) : 0.20\% \quad P(F|R_{1.7}) : 100\%]$

```
                     DividerRatio    <= 1.060e+02
                     InputFrequency  <= 9.975e+06
1.161e+00  <         VDD
1.238e-07  <         LV5
                     C1              <= 2.171e-12
1.185e-07  <         LV22
                     InputJitter     <= 7.606e-11
1.175e-07  <         LCP3
                     LCP51           <= 1.233e-07
                     C2              <= 6.643e-13
                     LV12            <= 1.220e-07
1.169e-07  <         LV31
```

Rule : 1.8 $[P(R_{1.8}) : 0.37\% \quad P(F|R_{1.8}) : 97.30\%]$

```
                     DividerRatio    <= 1.010e+02
9.895e+06  <  InputFrequency         <= 9.984e+06
                     LCP5            <= 1.161e-07
5.850e-13  <         C2
1.135e+00  <         VDD
                     LCP1            <= 1.242e-07
1.935e+05  <         R1
                     Temperature     <= 5.364e+01
```

Rule : 1.9 $[P(R_{1.9}) : 0.11\% \quad P(F|R_{1.9}) : 100\%]$

```
                     InputFrequency  <= 9.927e+06
                     LCP51           <= 1.186e-07
9.700e+01  <  DividerRatio           <= 1.090e+02
1.164e+00  <         VDD
1.145e-07  <         LCP3            <= 1.202e-07
```

Rule : 1.10 $[P(R_{1.10}) : 0.27\% \quad P(F|R_{1.10}) : 100\%]$

```
                     DividerRatio    <= 9.400e+01
                     InputFrequency  <= 9.982e+06
                     Temperature     <= 9.768e+00
5.848e-13  <         C2
                     LCP1            <= 1.231e-07
1.160e-07  <         LV14
                     LV5             <= 1.238e-07
```

Rule : 1.11 $[P(R_{1.11}) : 0.20\% \quad P(F|R_{1.11}) : 100\%]$

```
8.500e+01  <  DividerRatio           <= 9.500e+01
                     Temperature     <= 3.960e+01
                     LCP5            <= 1.157e-07
                     LCP51           <= 1.169e-07
                     C1              <= 2.157e-12
1.161e-07  <         LV1
9.971e+06  <  InputFrequency         <= 1.008e+07
```

Rule : 1.12 $[P(R_{1.12}) : 0.17\% \quad P(F|R_{1.12}) : 100\%]$

```
                     DividerRatio    <= 8.900e+01
                     LCP52           <= 1.154e-07
1.178e-07  <         LCP3
1.890e+05  <         R1
9.970e+06  <  InputFrequency
```

Rule : 1.13 $[P(R_{1.13}) : 0.11\% \quad P(F|R_{1.13}) : 100\%]$

```
                     DividerRatio    <= 8.900e+01
1.157e-07  <         LV14            <= 1.161e-07
                     LCP5            <= 1.204e-07
```

Rule : 1.14 $[P(R_{1.14}) : 1.17\% \quad P(F|R_{1.14}) : 94.02\%]$

```
                     DividerRatio    <= 8.800e+01
                     InputFrequency  <= 9.970e+06
5.946e-13  <         C2
1.145e-07  <         LCP3
                     LCP5            <= 1.251e-07
                     Temperature     <= 6.164e+01
1.159e-07  <         LV12
1.890e+05  <         R1
1.162e-07  <         LV23            <= 1.241e-07
1.171e-07  <         LCP11
```

Rule : 1.15 $[P(R_{1.15}) : 0.37\% \quad P(F|R_{1.15}) : 100\%]$

```
                     DividerRatio    <= 8.400e+01
9.898e+06  <  InputFrequency         <= 9.966e+06
1.212e+00  <         VDD
                     LV25            <= 1.266e-07
1.179e-07  <         LV33
1.142e-07  <         LCP4
                     Temperature     <= 5.250e+01
```

Region : 2 $[P(R_2) : 0.19\% \quad P(F|R_2) : 94.74\%]$

```
                     DividerRatio    <= 9.700e+01
1.008e+07  <  InputFrequency
1.191e+00  <         VDD
1.211e-07  <         LV22
```

Figure 4.9: Failure regions : PLL2. Naming conventions are the same as in Fig. 4.7 and Fig. 4.8.

Parameter	$IG(R\|parameter)$
DividerRatio	0.0765
InputFrequency	0.0359
VDD	0.0098
LCP5	0.0069
C2	0.0065

Table 4.2: Top 5 parameters in $PLL2$ ranked based on the importance metric described in Section 4.3.

Table 4.2 for simplicity. For PLL2, each of the following observations appears to be significant for a combination of the input conditions listed above (The exact values of X,Y,Z and U may differ from rule to rule):

1. $LCP5 \geq C \,\&\, LCPx < D$ where $x \in \{1, 3\}$.
2. $C2 \geq B$ where B is the nominal value.

Trend 1 can be identified as transistor mismatch effects from Fig. 4.10. This is easily resolved by increasing the channel lengths of the biasing circuit. We thus double LCP1x, LCP3, and LCP5x while adjusting transistor widths accordingly.

For $C2 \leq B$, we take a closer look at the filter block in Fig. 4.10. The impedance of the filter can be obtained as :

$$Z(s) = \frac{\frac{1}{C_2}\left(s + \frac{1}{R_1C_1}\right)}{s^2 + \frac{s(C_1+C_2)}{R_1C_1C_2}} \approx \frac{R_1\left(s + \frac{1}{R_1C_1}\right)}{s} \tag{4.12}$$

For the above approximation to hold, $C_2 << C_1$. However, $C_2 \neq 0$ as it helps smooth out the VCO control voltage ripple. Practically, we wish for $C_2 > C_1/k$ to ensure low jitter. The fact that $C_2 \leq B$ turns up as one of the failure conditions contributing to OutputJitter means that we have insufficient margin for C_1/C_2. Instead

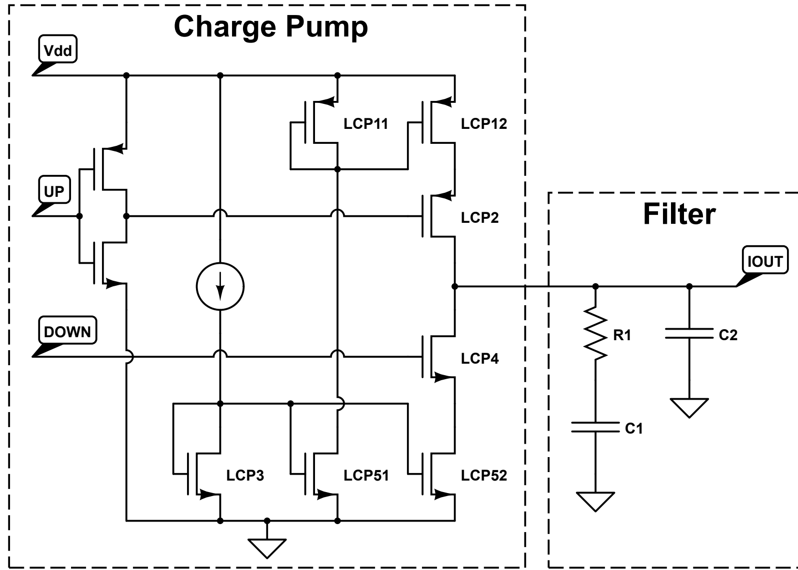


Figure 4.10: Charge pump and filter blocks within the PLL described in Fig. 3.8. $PLL1$ and $PLL2$ end up differing mainly in the values of LCPxx and R1, Cx due to the nature of the diagnosis information obtained for $PLL2$.

of lowering C_2 which is already very small, we raise C_1 instead.

As can be seen from Fig. 4.8, the resulting circuit PLL1 no longer has OutputJitter violations caused by $C2$. However, new failure regions dominated by interactions between $R1$ and $C1$ have been created. The importance of $R1$ and $C1$ is further validated by the parameter ranking in Table 4.1.

It is important to note that the diagnosis information presented above is just an aid. Interpreting the regions and actually fixing the circuit to optimize the yield may require significant domain knowledge. The failure regions simply list the minimum number of features required to explain a given failure region along with their associated intervals. They do not take into account topological information and hence may skip out on important correlations with other parameters. For example, $LCP5 \geq C$ may refer to a mismatch effect by itself if C is large enough as $LCPx$ defaults to

its entire legal domain. Knowing that $LCP5$ cannot be considered independently of $LCPx$ is thus critical in fixing the circuit. The domain knowledge is indispensable in fixing the circuit as well of course. Whether we wish to just resize the transistors or change the circuit altogether, designer intervention is not just desirable but necessary.

4.5 Summary

Our proposed approach does a very good job at discovering failure regions caused by interaction effects in a high-dimensional space. More importantly, we present this information in a manner that intuitively makes sense to the designer by reporting critical parameters and their associated tolerances. While we use standard rule-induction techniques, the framework is conceptually robust enough to leverage multiple such algorithms and still return a single set of failure regions and parameters ranked in order of their importance.

While we specifically demonstrate the effectiveness of our approach in pre-silicon debug, this work can be leveraged in any area that involves identifying the root cause behind out-of-specification failures or exciting such failures; a fact that we shall leverage in test-set selection in Section 6 and identifying systematic shifts in pre-silicon belief in the post-silicon domain in Section 7.

5. EQUIVALENCE CHECKING AND DIAGNOSING MODEL MISMATCH

We matched a circuit against its specifications to detect property violations in Section 3. Though we pushed the envelope of the size of systems we can deal with using such approaches, simulating large systems in SPICE within a reasonable turn-around-time may not be possible in the first place. System level detection and diagnosis techniques involving high level models of component sub-blocks thus remain indispensable.

Whether the model is machine generated or written by a designer, any modeling technique is prone to error and over-generalization. In this section we discuss how we may detect and diagnose mismatch between two models designed for the same purpose. Most often, one of these models will correspond to the real circuit and the other, a high level abstraction model meant to capture the functionality of said circuit. This high level abstraction model could be machine generated or written in System-C, Verilog-A, Simulink or any other high level modeling language. If the circuit and the high level model prove to be equivalent, the high level model can then be used to substitute for the circuit in system level simulation and analysis with a high degree of confidence.

5.1 Equivalence checking as a property checking problem

For two models to be completely equivalent, the complexity of the high level abstraction model will approach the complexity of original SPICE model making it useless for system level simulations. As a result, there is always a certain amount of tolerance involved in the equivalence checking process that captures the level of generalization we can tolerate at the system level.

More formally, given two models f_1 and f_2 and their corresponding param-

eter spaces $X1$ and $X2$, we wish to ensure that their corresponding properties $Y1 = f_1(X1)$ and $Y2 = f_2(X2)$ are always within a certain threshold of each other ($|Y1 - Y2| < \delta$ where δ is defined by the equivalence spec).

To do so, we first observe that $X1$ and $X2$ by definition must share some dimensions if they are trying to capture the same functionality. These shared dimensions will always include the input signal parameters. Depending on the model complexity, they may also include some design uncertainties such as power supply noise, jitter etc. As a result, instead of operating on $X1$ and $X2$ separately, it is simpler to re-define the problem as attempting to detect model mismatch between f_1 and f_2 over a joint parameter space $X = X1 \cup X2$.

Now, if we redefine the property space as $Y = |f_1(X) - f_2(X)|$, the equivalence checking problem is pretty much the same as the property checking problem we described in Section 3 with the probability of model mismatch given by equation (3.2). As a result, we do not reiterate relevant technical contributions here and go onto detail some results in Section 5.2.

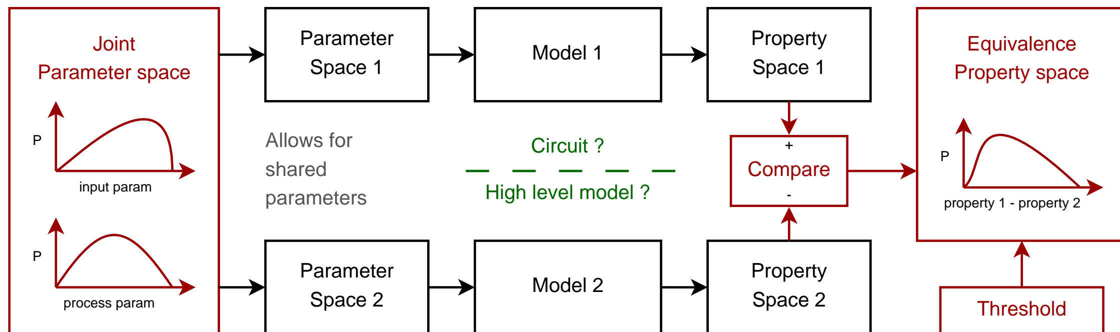


Figure 5.1: Redefining the equivalence checking problem as a property checking problem. Instead of dealing with each model individually, we operate across a joint parameter space defined by the union of both parameter spaces and over a joint property defined by the difference between shared properties.

A more graphical view of the discussion so far is given in Fig. 5.1. Effectively, we can think of the equivalence checking problem as property checking over a system composed of two models (one of which may be the circuit), connected by a comparator. What used to be specifications in property checking of course is now a list of thresholds that indicate the amount of mismatch that we are willing to tolerate between different properties.

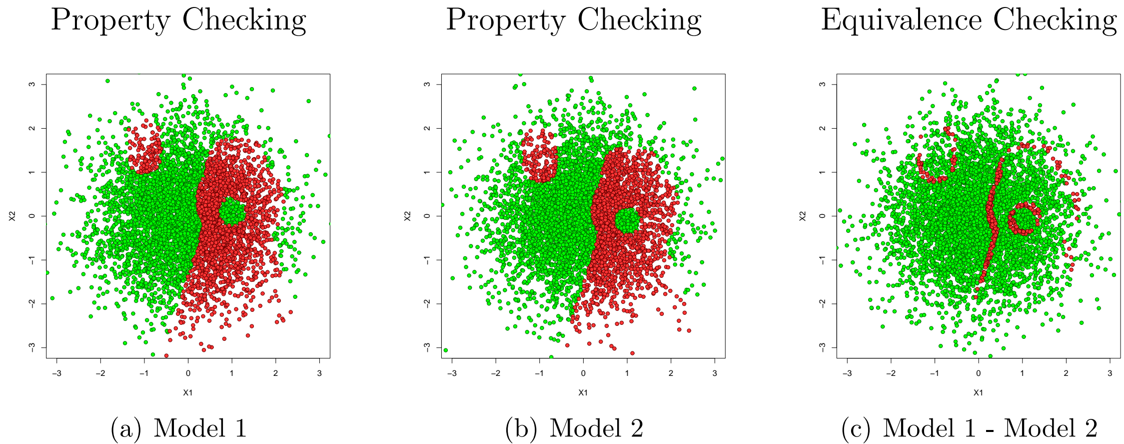
5.1.1 Practical considerations

An important observation to be made here is that both models should share the same set of properties in order to be effectively compared. More importantly, they should share at least part of the parameter space if they are indeed trying to capture the same functionality. These are not conditions imposed by the verification methodology but rather design considerations should the two models be indeed trying to capture the same concept. An example of this is the test-case we describe in the following section.

Another observation to be made is that the failure regions for equivalence checking can be expected to be different from the ones we have dealt with so far in property checking. An important assumption we made in Section 3 is that failure regions were more or less contiguous. As a result, the number of points in the interior of the failure region could be expected to be larger than the points on the boundary. While this is a valid assumption for the property checking case given how failures are usually centered around the extreme values of some parameters, the same cannot be said for equivalence checking.

An extreme example of this is visually demonstrated in Fig. 5.2. Even though the failures regions for property checking (Fig. 5.2(a) and Fig. 5.2(b)) in both the models look similar and the failure regions are highly contiguous, the length of the

boundary for the equivalence checking problem (Fig. 5.2(c)) is twice that of the property checking problem even though the level of mismatch (and thus the probability of mismatch) is tiny.



(a) Model 1 (b) Model 2 (c) Model 1 - Model 2

Figure 5.2: Difficulty when adaptively sampling the equivalence checking problem as compared to the property checking problem. Though Fig. 5.2(a) and Fig. 5.2(b) are very similar to each other, on comparing Model 1 with Model 2, we get a doubly complex failure manifold in Fig. 5.2(c).

Of course this has a direct impact on both the equivalence checking and the model mismatch diagnosis. For the equivalence checking, it means we need a lot more points to achieve the same accuracy. For diagnosis, it means that the failure regions are either going to be very small or together explain only a portion of the actual parameter space that leads to failure.

5.2 Results

We use the same circuit PLL1 (Fig. 3.8) we used in Sections 3 and 4 as the SPICE netlist and compare it against a Verilog-A behavioral model which captures the behavior of the circuit in response to the following parameters :

- Input frequency
- Input jitter
- Divider Ratio
- Supply voltage
- Filter R,C (3)
- Output load

In other words, the following parameters are exercised by the SPICE netlist alone:

- Temperature
- Channel lengths (31)

The Verilog-AMS model takes an order of magnitude less time to simulate as compared to the SPICE model. In place of the 30 minutes it takes for the SPICE level netlist, the Verilog-AMS model takes only 3 minutes to simulate. On the flip side, it reacts to far fewer parameters meaning that it is an over-generalized version of the actual netlist.

Naturally, this over-generalization ensures that the two views of the same circuit do not always give the same response. This probability of model mismatch is simply referred to as $P(F)$ over the next few sections. It is to be noted that this model mismatch can occur because (a) the behavioral model was not designed to capture the nominal response to certain shared parameters, or (b) because the parametric variations not captured by the behavioral model are significant enough to cause the response to exceed the mismatch threshold. The diagnosis information we present in Section 5.2.2 may thus include both kinds of parameters.

5.2.1 *Equivalence checking*

Since the equivalence checking problem is in essence similar to the property checking problem, we would expect our observations regarding Table 5.1 to be much the same as those detailed in Section 3. However, it turns out that the boundaries are harder to characterize in the equivalence checking problem as failure regions may not be very contiguous which was the assumption we made in Section 3. As a result, convergence to a useful probability metric takes longer. This is in line with our expectations as detailed in Section 5.1.1.

	PLL1					
	99% Bootstrap CI (MC + LHS)		Bias compensated interval learner			
			MC + LHS		Adaptive	
	$P(F)$	$minP, maxP$	$P(F)$	$minP, maxP$	$P(F)$	$minP, maxP$
300	2.33	0.33, 5	0.22	0.17, 0.67	17.8	16.1, 19.8
400	3	1, 5.5	0.11	0.094, 0.24	11.3	10.0, 13.2
500	2.2	0.8, 4.2	0.13	0.11, 0.19	8.7	7.65, 10.2
600	3	1.33, 5	0.11	0.098, 0.16	5.7	4.26, 6.66
700	2.14	0.86, 3.71	0.18	0.16, 0.23	5.0	4.21, 6.55
800	3	1.63, 4.75	0.21	0.17, 0.24	4.3	3.57, 5.56
900	2.22	1, 3.55	0.18	0.15, 0.22	3.78	3.13, 5.08
1000	2.3	1.2, 3.6	0.25	0.22, 0.29	3.82	3.15, 5.11
1100	2.09	1.09, 3.27	0.32	0.28, 0.36	3.42	2.76, 4.83
1200	2.52	1.76, 3.36	0.30	0.26, 0.5	3.17	2.52, 4.55
10000	2.58	2.17, 2.98	← Reference to compare against.			

Table 5.1: Model mismatch probability and interval estimates for PLL1 when compared against high level Verilog-AMS model. All probabilities expressed in %.

5.2.2 Diagnosing model mismatch

The failure regions and the parameter ranking to explain mismatch between PLL1 and the Verilog-AMS model are shown in Fig. 5.3 and Table 5.2 respectively.

We first observe from Table 5.2 that 3 of the top 5 parameters (Temperature, LCP2, LV2) are not modeled in the Verilog-AMS model. This implies that there might be a possibility that the Verilog-AMS model is not detailed enough to capture the desired behavior to the desired level of accuracy.

Parameter	$IG(R \mid parameter)$
Temperature	0.0130
DividerRatio	0.0127
InputFrequency	0.0126
LCP2	0.0609
LV2	0.0541

Table 5.2: Top 5 parameters in explaining mismatch between PLL1 and Verilog-AMS model based on the importance metric described in Section 4.3.

Looking over the failure regions in Fig. 5.3 further corroborates this hypothesis. All the failure regions are explaining some combination of parameters modeled by the circuit alone in conjunction with certain conditions on the shared parameters (DividerRatio, InputFrequency etc). Even more importantly, failure region 2 is made up of LV2 and LV21 alone. In other words, it refers to a very specific transistor mismatch which has certainly not been modelled in the Verilog-AMS model.

The easiest way to fix the mismatch is by increasing the error tolerance. However, this may not be desirable for system level performance. Depending upon at which stage of the design process we may be at two more fixes are possible: (a) Fix the

Region : 1 $[P(R_1) : 1.00\% \quad P(F|R_1) : 92\%]$

Rule : 1.1 $[P(R_{1.1}) : 0.28\% \quad P(F|R_{1.1}) : 100\%]$

```
                DividerRatio     <= 1.020e+02
1.239e-07  <         LV44
2.468e-12  <         C1
4.698e+01  <    Temperature
                InputFrequency   <= 1.000e+07
```

Rule : 1.2 $[P(R_{1.2}) : 0.44\% \quad P(F|R_{1.2}) : 90.91\%]$

```
                     LCP2        <= 1.181e-07
4.698e+01  <    Temperature
                InputFrequency   <= 9.941e+06
                DividerRatio     <= 1.020e+02
```

Rule : 1.3 $[P(R_{1.3}) : 0.12\% \quad P(F|R_{1.3}) : 100\%]$

```
                DividerRatio     <= 8.100e+01
1.658e+05  <         R1
                InputFrequency   <= 1.002e+07
```

Rule : 1.4 $[P(R_{1.4}) : 0.12\% \quad P(F|R_{1.4}) : 100\%]$

```
                     LV12        <= 1.151e-07
1.141e-07  <         LV1
4.799e+01  <    Temperature
2.362e-07  <         LCP51
5.369e-13  <         C2
```

Rule : 1.5 $[P(R_{1.5}) : 0.12\% \quad P(F|R_{1.5}) : 100\%]$

```
2.460e-07  <         LCP51
                InputFrequency   <= 1.000e+07
                DividerRatio     <= 1.070e+02
1.827e+01  <    Temperature
```

Rule : 1.6 $[P(R_{1.6}) : 0.20\% \quad P(F|R_{1.6}) : 80\%]$

```
1.183e-07  <         LV21
4.701e+01  <    Temperature
                     LV2         <= 1.176e-07
                InputFrequency   <= 9.994e+06
2.350e-12  <         C1
```

Region : 2 $[P(R_2) : 0.08\% \quad P(F|R_2) : 100\%]$

```
1.277e-07  <         LV2
                     LV21        <= 1.257e-07
```

Figure 5.3: Failure regions : PLL1 vs Verilog-AMS model mismatch. Naming conventions are the same as in Fig. 4.7 and Fig. 4.8.

circuit, and (b) Fix the model.

Fixing the circuit means we have to make sure that the circuit response is much less susceptible to variation. If we think of the Verilog-AMS model as the golden model, then this is indeed the correct approach for it means designing a more robust circuit. This fix would be similar to the one we made to get from $PLL2$ to $PLL1$ in Section 4.

On the other hand, if we are to change the model to match the circuit data (such as when constructing a high level model to explain an existing circuit), the only option may be to update the high-level model. Doing so however, may not always be possible by changing the model parameters alone. The model may not have been constructed to capture the effect of key parameters in the first place (Such as Temperature, LV2 and LCP2 in this example). The only option in that case may be to actually model the effect of these parameters in the high-level model or relax

the mismatch tolerance.

Relaxing mismatch tolerances of course affects the confidence of the system-level simulation. Capturing the effect of key parameters leading to model mismatch in the high level abstraction model proves more desirable. This is even more so when one considers that these parameters may be interacting with other parameters from adjoining blocks to give rise to system level failures. The more the number of low-level parameters modelled by the abstraction model the higher the chances of capturing these interactions. On the flip side, system-level simulation and verification cost may go up as a result of the same.

5.3 Summary

In this section we framed the equivalence checking problem as a property checking problem that we then addressed using the contributions in Section 3. Our observations regarding the estimates are much the same as in Section 3 except that convergence takes longer. This is due to the length of the failure boundary defined by the mismatch between two models.

The efficacy of the failure regions is also demonstrated when applied to diagnosing model mismatch. However, fixing such a mismatch is even more complex than in the property checking case as the high level model may not have sufficient representational flexibility to capture the circuit behaviour. Of course, relaxing the thresholds further does give us 0% $P(F)$ but the threshold is a measure of deviation between the circuit and the model in itself. Relaxing it too far may make the model useless from a system level perspective even if the reported $P(F)$ is very low.

6. TEST SET SELECTION TO MAXIMIZE OBSERVED FAILURES

The high-dimensional "parameter space" we have dealt with so far, contains two types of parameters in reality : controllable and uncontrollable. The goal of test set selection is to pick controllable parameters such that the probability of observing failures is maximized in the presence of uncertainty along the uncontrollable ones.

More formally, we wish to sample a set of t points $\hat{C} \subset C$ such that the probability of exciting a failure, $P(F|\hat{C})$ is maximized. To do so, we observe

$$P\left(F|\hat{C}\right) = \sum_{c \in \hat{C}} \int_{\bar{c} \in \bar{C}} P(F|c, \bar{c}) p(\bar{c}|c) d\bar{c} \tag{6.1}$$

where $P(F|c, \bar{c})$ is either 0 or 1 depending on whether $x = \{c, \bar{c}\}$ leads to a failure or not and $p(\bar{c}|c)$ represents the expected distribution along the uncontrollable dimensions for a given controllable point c.

In the pre-silicon domain, both $p(c)$ and $p(\bar{c}|c)$ are known (based on certain assumptions). That leaves $P(F|c, \bar{c}) = P(F|x)$ which can be directly ascertained from any failure model such as the one we build in Section 3. However, since we cannot actually integrate over $\bar{c} \in \bar{C}$, we will need to sample over both c and $\bar{c}|c$. Given the large number of samples required to characterize both $p(c)$ and $p(\bar{c}|c)$, it makes computationally more sense to use the failure regions we obtain in Section 4 as the knowledge representation scheme instead. A complete framework to do so is shown in Fig. 6.1.

While these failure regions may overgeneralize or skip over a few failure points, the critical tests we wish to pick are the ones that demonstrate the highest probability of exciting a failure. More importantly, while we expect the pre-silicon model to be

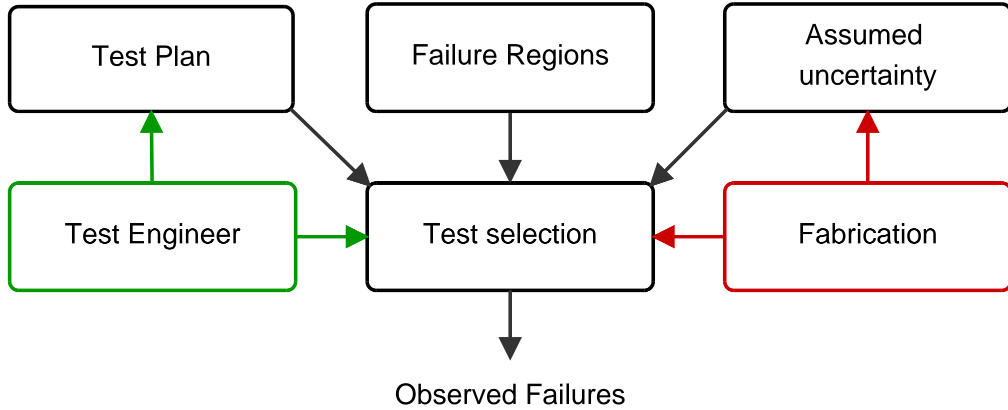


Figure 6.1: Failure region based test selection methodology. The uncertainty information $p(\bar{c}|c)$ can be fed back from the fabrication process. The test-engineer's input is used in determining the test plan (and thus $p(c)$) as discussed in Section 6.1. The failure regions serve as a source of pre-silicon knowledge for $P(F|c, \bar{c})$.

representative of real silicon, the amount of computational effort it will take to use a perfect pre-silicon model far outweighs the benefits. This is more so because the observed $P(F|c)$ using our final measurements will show a high degree of variance due to small sample size (If we assume Monte Carlo sampling over the uncontrollable parameters, variance $\propto 1/m$ where m is the number of measurements taken at a specific test point c).

We thus describe how we may leverage the failure regions from Section 4 for test-set selection in Section 6.2. Since we are dealing with a test-set selection problem and not a test-generation problem however, we first discuss the set of all possible tests in Section 6.1.

6.1 The test plan

A test plan documents the strategy that will be used to verify and ensure that a product or system meets its design specifications. A test plan is usually prepared by

test engineers with significant input from the designers.

It may include testing strategies for multiple scenarios depending upon the product and at which point of the design cycle the product may be. Of these, two are particularly important in the post-silicon domain :

- Design Verification - typically performed on a small sample of units during the development or qualification stages of the product.

- Production test - typically performed in an ongoing manner during product assembly for purposes of performance verification and quality control.

The testing we talk about in this dissertation is suitable only for design verification. Functional tests are usually too expensive for production testing and the pre-silicon data we collect in Section 3 and Section 4 verify the functionality of the circuit against certain performance metrics.

Due to the involvement of the designers and test engineers, a significant amount of domain knowledge may already be available in terms of the what to test, how to test and what resources are available in terms of design for test. This does not mean we cannot make their job easier.

The parameter rankings we obtained in Section 4 already tells us what parameters are important in terms of separating pass from failure cases. The parameters which do nothing to separate pass from fail cases in the pre-silicon domain are likely also to remain unimportant in the post-silicon domain even if we can control them. On the other hand, controlling the parameters which were determined important in the pre-silicon domain, would have a better level of controllability in terms of exciting pass vs. failure cases in post-silicon too.

Even this information however, is just meant to assist the test engineer. A significant amount of domain knowledge may still be required at this stage. For example,

two parameters which are very closely related to each other in pre-silicon may not necessarily be as closely related in post-silicon. From a test-plan perspective, this information becomes suddenly relevant as our pre-silicon failure regions (and thus the parameter rankings) would be able to explain the failures in terms of any one of these parameters.

An even more important consideration is whether or not it is possible to control the chosen parameters in the first place, and if so to what level of granularity. Implementing DfT structures where we can vary the value of a parameter in a continuous fashion is usually very expensive. Practically, DfT structures [32, 33, 34, 35] are implemented to be set to one of a few different modes which can be controlled with a digital knob. Effectively, the continuous space C has now instead been discretized into a finite set of controllable points $S \subset C$.

6.2 Leveraging failure regions for test set selection

Assuming that the pre-silicon failure regions are a pretty good indicator of post-silicon reality, our goal is to effectively boost the probability of observing these failure regions. This is empirically demonstrated in Fig. 6.2.

An added benefit of a failure region based approach is that it now becomes possible to ensure that both R_i and R_j (where $i \neq j$) have a non-zero probability of being covered by $\hat{C}$ instead of just maximizing $P(F|\hat{C})$. This is possible even if the effect of uncontrollable dimensions is higher for R_i than R_j.

For now however, let us assume that our goal is simply to observe as many points lying within any of the failure regions as possible. More formally, we wish to sample a set of points $\hat{C} \subseteq S$ such that the probability of exciting regions $R = \{R_1 \ldots R_m\}$,

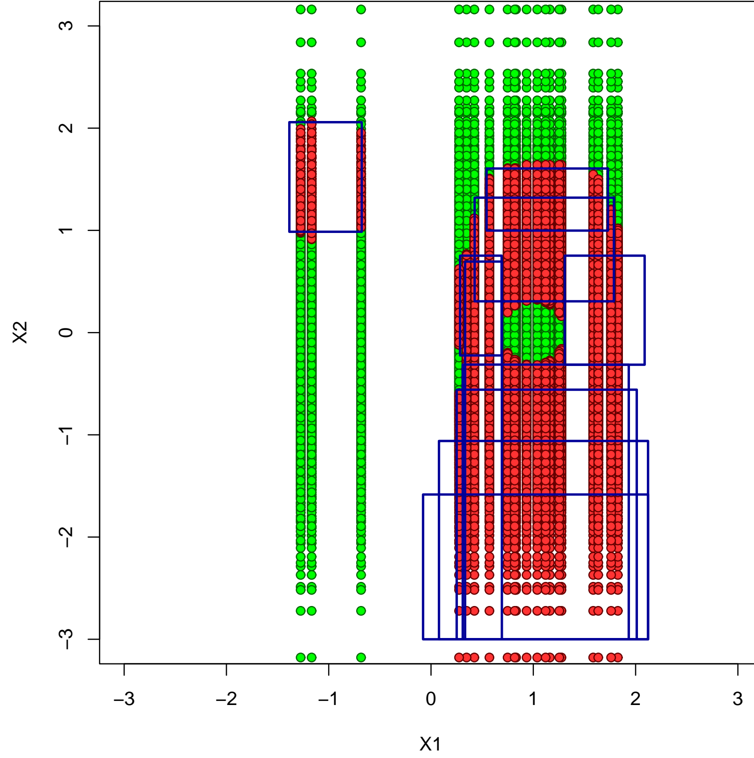


Figure 6.2: Selecting values for controllable test parameters. X1 is the controllable parameter. For every sampled point on X1, the expected uncertainty along X2 is demonstrated. The expected uncertainty is based on pre-silicon assumptions. The actual post-silicon distribution may be different as shown in Fig. 7.1.

$P(R|\hat{C})$ is maximized. To do so, we observe

$$P\left(R|c\right) = \sum_{\bar{c}\in\bar{C}} P(R|c,\bar{c})p(\bar{c}|c) \tag{6.2}$$

$$P\left(R|\hat{C}\right) = \sum_{c\in\hat{C}} P\left(R|c\right) \tag{6.3}$$

where $P(R|c,\bar{c})$ is either 0 or 1 depending on whether $x = \{c,\bar{c}\} \in R$ or not.

Thus, there are in fact, two computations happening here. The first one, is estimating $P\left(R|c\right)$ over all candidate test-points $c \in S$. The second computation is

actually picking t points $c \in S$ to form $\hat{C}$. The simplest way to maximize equation (6.3) by doing so is simply ranking all $c \in S$ by $P(R|c)$ and picking the first t controllable points. However, this may cause $\hat{C}$ to be concentrated in debugging a single failure mechanism. This is discussed further in Section 6.2.1.

6.2.1 Implementation details

To develop an actual algorithm, let's look at equation (6.2) to start with. We first note that $p(c)$ and $p(\bar{c})$ need not be independent. For example, the threshold voltage of a transistor V_t might be influenced by the supply voltage VDD which may be controllable. As a result, $p(\bar{c}|c)$ is not necessarily the same as $p(\bar{c})$. c however is directly controllable by definition, allowing us to sample $c \in C$ without taking $\bar{c}$ into account. The effect of $\bar{c}$ will be captured when trying to pick $\hat{C}$ based on $P(R|c)$. This computation of $P(R|c)$ for every $c \in S$ forms the majority of pseudocode given in Algorithm 1. The inner loop computes $P(R|c, \bar{c})\ \forall\, \bar{c}|c$. The outer loop computes $P(R|c)\ \forall\, c \in S$. Lines 1 and 13 simply generate S or pick $\hat{C}$ from S using $P(R|c)\ \forall\, c \in S$.

$P(R|\hat{C})$ is thus maximized by a two-step sampling process. We first generate a large set of candidate test points S. We then compute and use the probability of exciting R at each point in S to sample within S. Algorithm 1 summarizes this process.

Note that the last step does not just pick the top t points. Instead, we use a weighted sampling scheme [64] where the probability of selecting $c \in S$ is proportional to $P(R|c)$. Points with the highest $P(R|c)$ of course have the highest probability of being selected where as a $P(R|c) = 0$ will ensure that a point is never selected. The goal here is to ensure that $\hat{C}$ is not completely clustered around the same point as that only gives us information on one failure mechanism and is not very interesting

Algorithm 1 GenTestParams$\big(p(C), p(\bar{C}|C), R = \{R_1 \ldots R_m\}\big)$

```
1:  S ← sample c ∈ C with probability p(c)
2:  for c ∈ S do
3:      S̄ ← sample c̄ ∈ C̄ using p(c̄|c)
4:      for c̄ ∈ S̄ do
5:          if {c, c̄} ∈ R then
6:              P(R|c, c̄) ← 1
7:          else
8:              P(R|c, c̄) ← 0
9:          end if
10:     end for
11:     P(R|c) ← Σ_{c̄∈S̄} P(R|c, c̄) p(c̄|c)
12: end for
13: Ĉ ← sample c ∈ S with probability P(R|c) / Σ_{c∈S} P(R|c)
```

from a debug perspective.

Also, all the sampling steps (lines 1, 3 and 13) herein refer to sampling without replacement [65]. They are additionally all weighted [64] based on different probability metrics. Since line 1 is just a surrogate for a good test plan, it is to be noted that this may not always be the case. The test plan may allow for multiple tests with the same controllable parameters. Typically, this is a good idea when the variance in $p(\bar{c}|c)$ is very high. As discussed in Section 6.1, a good test plan will try to ensure that it never happens in practice but we may not be able to add sufficient DfT to guarantee guarantee the same. For the purpose of this work, we assume there is only one of each controllable point in the test-set.

The final number of samples in $\hat{C}$ is determined by the number of test inputs t the verification engineer wishes to generate. The number of candidate points S is heuristically chosen as 25 times that number. The number of points in $\bar{S}$ is chosen as 100 since we integrate it out on line 11 in Algorithm 1 anyway. It is to be noted that no additional SPICE simulations are required as the failure regions have sufficient

knowledge to predict circuit failure.

6.2.2 Failure region based test coverage

Test coverage is a complicated subject when dealing with parametric faults [66, 67]. Since we are dealing with a continuous parameter space, the set of all possible faults is infinite. This is unlike the case of catastrophic faults where failures can be measured in terms of the number of shorts and opens that can be detected. Moreover, there is no guarantee that we will actually hit a parametric fault on testing at c due to the effect of $\bar{c}$.

Since the set of all faults is infinite, it is impossible to come up with a % of faults covered by the test-set. However, we still need some mechanism to be able to quantify how good the generated test-set is. More specifically, we need a metric that will allow us to compare how good the test-set is, as compared to just randomly sampling the test plan.

Assuming that the failure regions from Section 4 form a fault dictionary we wish to exercise, $P(R|\hat{C}) = \sum_{c \in \hat{C}} P(R|c)p(c|\hat{C})$ gives us a good indicator of the number of failures we should expect to see when exercising $\hat{C}$. Comparing $P(R|\hat{C})$ with $P(R|S)$ also gives us an indicator of how good we expect the test-set to be compared to randomly sampling the test plan. Of course, should the parameter space be entirely controllable, we should be aiming for $P(R|\hat{C}) = 1$. However, this value will be a lot lower in practice depending upon the effect of $\bar{c}|c$.

Now, the above metrics correspond to the expected coverage in the pre-silicon domain. Deviations in $\bar{c}$ may cause the actual coverage in the post-silicon domain to be different from the expected coverage as we shall see in Section 7. But for the purpose of this section, $P(R|\hat{C})$ serves as a good indicator of how well our proposed test-set selection algorithm performs.

6.3 Results

Both the test case (Fig 3.8) and the failure regions we use to perform test set selection have been described in Section 3 and Section 4 respectively. Since we are dealing with actual silicon here, it is also relevant to discuss which of these parameters are controllable and observable in the post-silicon domain. Only the input frequency, divider ratio, supply voltage, temperature and output load are assumed to be controllable. In addition to the above, we can also observe Input Jitter and all eight properties listed in the specifications. However, 34 parameters (transistor channel lengths and filter R,Cs) remain unobservable.

It should be noted here that the post-silicon data we use here is actually simulated via SPICE after introducing shifts in various parameters to ensure it doesn't follow the pre-silicon distribution. The reason for using simulated instead of real post-silicon data here is so that the probabilities are actually measurable in both the pre and post-silicon domains.

The effect of applying our test parameter selection methodology on the circuit we have previously described ($PLL1$) is shown in Fig. 6.3. Why some regions are boosted more effectively than others becomes immediately obvious if we are to refer to the parameter ranking in Table 4.1. The most important controllable parameter happens to be the Divider Ratio. As a result, rules demarcated by $DividerRatio$ ($R_{1.1}$ and R_2) are boosted most effectively as well. On the other hand, some rules ($R_{1.3}$ and R_4) are not affected at all since they are not demarcated by any controllable parameter. Thus irrespective of the exact test-signals chosen, their probability is not going to be affected.

A similar trend is seen if we run Algorithm 1 on the failure regions generated for the alternate design $PLL2$. Given the large number of regions required to de-

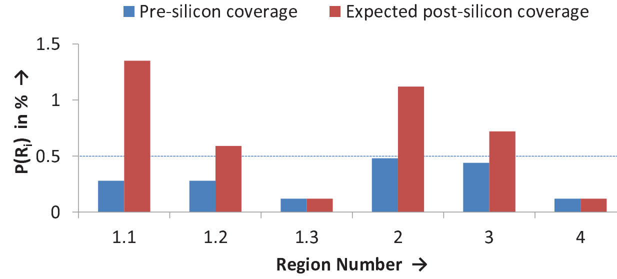


Figure 6.3: PLL1 Test selection. Probabilities of regions demarcated by one or more controllable parameters are effectively boosted. If the region is bounded by uncontrollable elements alone, no test set can perform better than randomly sampling the test plan.

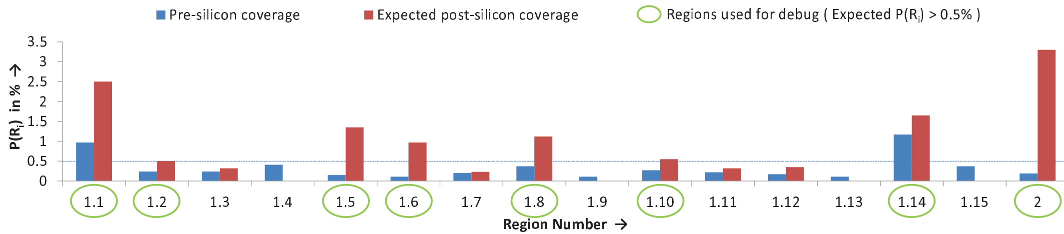


Figure 6.4: PLL2 Test selection. Regions with failure probability above 0.5% when excited with chosen test parameters will be considered statistically significant and used for post-silicon debug.

scribe the failure mechanisms in this circuit, we additionally draw a line as to which regions to consider statistically significant after test selection. Since we are dealing with about 2000 post-silicon data samples in our setup, we draw this line at 0.5% so as to ensure that we are likely to observe at least 10 failures for each significant region. The results from test selection are shown in Fig. 6.4 with the statistically significant regions marked in green. As expected, rules which feature the most important parameters from our parameter ranking in Table 4.2 feature prominently in

the extracted failure regions as well.

6.4 Summary

As circuit sizes continue to grow, our controllability in the post-silicon domain continues to degrade as well. Observing failures during post-silicon debug thus becomes a very difficult problem in itself. Selecting controllable test parameters to boost the probability of observing certain failure mechanisms helps reduce the measurement effort involved in doing so. Additionally, it will help with debug as well as we shall see in Section 7.

In this section we demonstrate a test set selection methodology that best uses (possibly approximate) pre-silicon knowledge to choose a test set to excite real silicon with. We also demonstrate how the parameter ranking we described in Section 4 helps us in determining which parameters we wish to control and observe to do so as well.

7. IDENTIFYING SYSTEMATIC SHIFTS IN PRE-SILICON BELIEF

The dimensions in the high-dimensional “parameter space” we have dealt with so far can also be split into two based on their observability in the post-silicon domain. The goal of post-silicon debug is to identify which (and in which direction) unobservable parameters may have shifted from what we assumed about them in the pre-silicon domain. If we are interested in inferring exact parameter values, we may have no choice but to probe additional measurements which are significantly correlated to said parameter value. But if we are just interested in identifying shifts in specific failure mechanisms, we can do a lot more using existing data.

Since we have already captured these failure mechanisms as part of the “failure regions” in Section 4, identifying shifts in the importance of these failure regions allows us to determine which failure mechanisms might have become more (or less) important. While we are dealing with a coarser level of information (regions not exact values), the inability to determine exact parameter values is offset by the fact that we already know which parameters are important (or not important) for a given failure region based on pre-silicon knowledge (See Section 4 for details).

The reason for looking at the holistic picture instead of looking at each individual response is that the obervability into a real circuit is limited. The cost (design or test time) of probing internal nodes of the circuit once it has been fabricated is prohibitive. And without being able to observe significantly correlated responses or parameters, it may be impossible to determine what parametric shifts may or may not have occurred. However, while we may not be able to reconstruct the exact mapping, it is possible to detect systematic shifts in the probability of a given region of the parameter space even in this scenario.

7.1 Identifying shifts in importance of "failure regions"

If we can observe the probability of occurrence of specific failure mechanisms in the pre and post-silicon domain, we can already narrow the post-silicon deviations down to very few trends. An extreme case of this can be visualized by comparing Fig. 7.1 with Fig. 6.2. The change in distribution manifests itself through the change in probabilities of different failure regions.

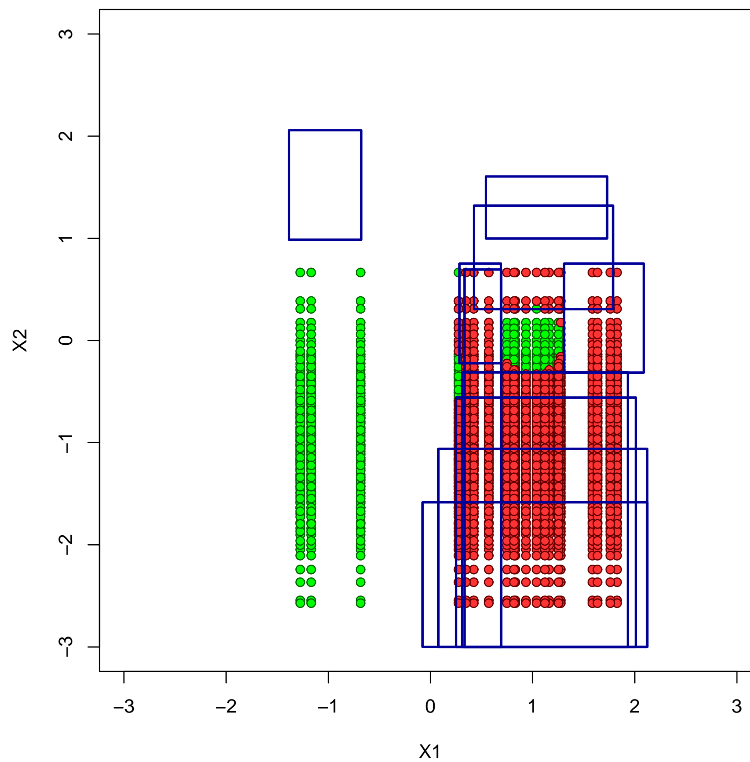


Figure 7.1: Post silicon debug. X1 is controllable and can be selected during test. Change in distribution of the unobservable parameter X2 causes $P(R_i)$ to change for both clusters.

Since we have already identified the critical parameters and associated tolerances

for each failure region, we can then easily determine which parameter to target as well. Additional guidance for this is provided by the parameter ranking which we discussed in Section 4.3.

Thus our first goal is to obtain the probability that a given region R_i has been exercised, given a set of observations and the failure region information that we are privy to. If $\hat{O} \subset O$ refers to a set of post-silicon observations, $P(R_i|\hat{O})$ can be obtained as

$$P\left(R_i|\hat{O}\right) = \sum_{o \in \hat{O}} P(R_i|o)p(o) \tag{7.1}$$

Since $c \subset o$, we can similarly obtain the pre-silicon belief by generating Monte Carlo samples for the uncontrollable parameters around each $c \in \hat{C}$. In other words, for each $c \in \hat{C}$, we generate a set of samples $\bar{c} \in \bar{C}$ with probability $p(\bar{c}|c)$ and simulate $x = \{c, \bar{c}\}$ via SPICE. $\check{O} \subset O$, the set of pre-silicon observations can then be drawn from this simulated SPICE data. Comparing $P(R_i|\hat{O})$ against $P(R_i|\check{O})$ allows us to detect systematic shifts in the importance of region R_i when moving from the pre-silicon to post-silicon domain. The complete framework is summarized in Fig. 7.2.

So the only problem we are left to deal with is estimating $P(R_i|o)$ where o can be an observation in either $\hat{O}$ or $\check{O}$. This is a standard density estimation problem that we can solve using kernel density estimation (KDE) [68].

7.1.1 Kernel density estimation

Kernel density estimation (KDE) [68] is a non-parametric method to estimate the probability density function of a random variable. Inferences about the population can be made using a finite data sample by smoothing over the known points. If

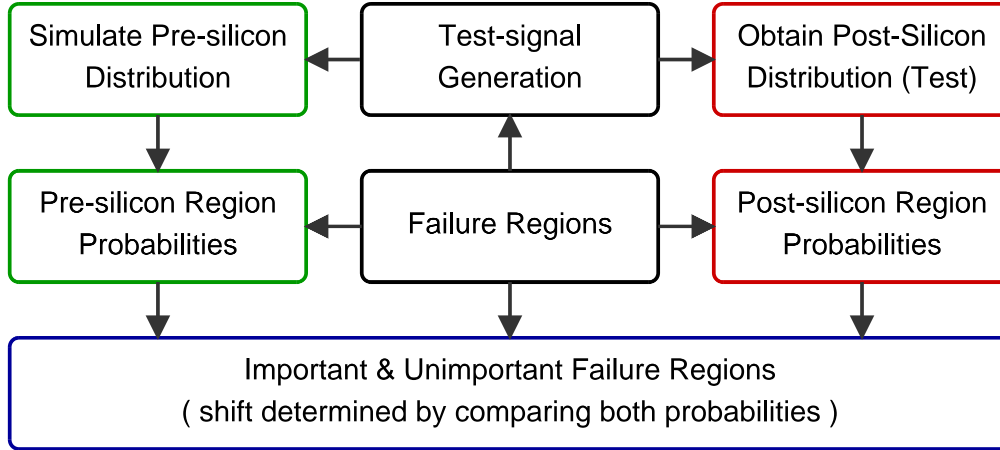


Figure 7.2: Complete post-silicon debug flow. The estimated failure region probabilities over both pre and post-silicon data are compared to identify systematic shifts from the pre-silicon belief in the post-silicon domain.

$\left\{x^{(1)} \ldots x^{(n)}\right\}$ refers to an independent and identically distributed sample drawn from some distribution with an unknown density f, we wish to estimate the value of f at an unknown point x using $\left\{x^{(1)} \ldots x^{(n)}\right\}$. Its kernel density estimator is

$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^{n} K\left(\frac{x - x^{(i)}}{h}\right) \tag{7.2}$$

where $K(.)$ is the kernel - a symmetric but not necessarily positive function that integrates to one - and $h > 0$ is a smoothing parameter called the bandwidth.

The main challenge with this technique is choosing a suitable h since the scale of each axis may differ. This scaling problem is naturally addressed if we can choose different bandwidths h_j along each of the dimensions x_j. By using a "generalized product kernel function" [69], this concept can be extended further to deal with multiple data types as well (continuous, ordered and factors).

Given the dimensionality of the systems we wish to debug, we haven't used cross-

validation techniques to search for the optimal value of h_j at this time. h_j is instead statistically computed as $h_j = 1.06\sigma_j n^{-1/(2P+1)}$ where n is the total number of observations and P is the order of the kernel (by default 2). σ_j is an adaptive measure of spread of the jth continuous variable defined as

$$\sigma_j = min(\text{standard deviation}, \text{interquantile range}/1.349) \tag{7.3}$$

While we may not be able to estimate the real post-silicon probability correctly using the above choice of h, all we need is a comparison and not an exact metric. As long as we make sure that we use the same h in estimating $P(R_i|\hat{O})$ and $P(R_i|\check{O})$, this h proves to be sufficient as shown by our results in Section 7.2. Any bias introduced by the choice of h affects both the pre and post-silicon probability estimates allowing us to still compare the two.

For further insight into the exact KDE method used and for non-parametric extensions that are being considered, we refer the reader to [69].

7.1.2 Implementation details

The above technique works as long as there is sufficient correlation between the observable quantities and the unobservable ones. In our case, the observable quantities include the desired output properties. If a parameter is not significantly correlated to these properties, a deviation in said parameter will not affect the failure region probabilities. While this means that our approach is unable to catch this deviation, it also means that the deviation is not very important in the first place as it is not capable of pushing the output properties into an out-of-specification failure region.

It is also to be noted that there is always a certain amount of noise associated with our estimate of $P(R|\hat{O})$ and $P(R|\check{O})$. This can arise from the estimation technique

used (KDE), the choice of bandwidth h but most importantly, it arises due to the limited number of data points we are forced to deal with in the post-silicon domain. Since the sampled distributions is never going to replicate the true distributions, we cannot directly compare the probability values without taking into account some notion of statistical significance. In our case, we leverage a very simple heuristic to do so. If the post-silicon probability is within $\pm 10\%$ of the pre-silicon probability, we assume the deviation is simply noise and the importance hasn't shifted. Anything below or above that margin is considered to be statistically significant and is used for debug purposes.

7.2 Results

Both the test case (Fig 3.8) and the failure regions we use to perform test set selection have been described in Section 3 and Section 4 respectively. We further assume that the test points have been selected using the test selection methodology detailed in Section 6 and the post-silicon data is similarly simulated as in Section 6.3. The benefit of simulating instead of measuring post-silicon data is that the real values for the region probabilities are known in the post-silicon domain as well thus allowing us to validate how well we estimate pre-silicon vs post-silicon shifts.

As noted in Section 7.1.1, since we we are forced to estimate the probabilities using kernel density estimation, the estimated pre-silicon failure probability described in Fig. 7.3 and Fig. 7.4 may not reflect the real probabilities in Fig. 6.3 and Fig. 6.4. However, as long as we used the same h for both pre and post-silicon estimation, the estimated pre and post-silicon probabilities within Fig. 7.3 and Fig. 7.4 can be compared.

As discussed in Section 4, $R1$ and $C1$ are critical in determining the failure probability for $PLL1$. An increase in $R1$ or $C1$ causes an increase in the overall failure

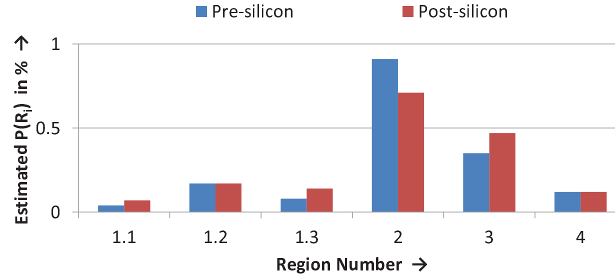


Figure 7.3: PLL1 debug results. The importance of all regions except region R_2 seems to go up in the post-silicon domain. Studying region R_2 in more detail actually allows us to identify that the mean value of $C1$ increased in the post-silicon domain.

probability. This is reflected in Fig. 7.3 as well where the estimated importance of most of the regions goes up in the post-silicon domain. The only exception to this rule is region R_2 which actually decreases in importance.

To understand why this occurs, we observe the bounds for $R1$ and $C1$ in region R_2. $R1 > 1.61e+05$ refers to resistance values higher than the nominal design value of $1.6e + 05$. This implies one possible reason for the decrease in this probability is a systematic reduction in $R1$ values over all our test-chips. On the other hand, $2.4e - 12 \leq C1 \leq 2.6e - 12$ refers to a small band of interest around the nominal $C1$ value of $2.5e - 12$ indicating that the mean value could have either increased or reduced. We can easily reject the first hypothesis as a reduction in $R1$ would have caused the importance of all the other regions to decrease as well. An increase in $C1$ is the only hypothesis that satisfies the shift in probability over our failure regions allowing us to identify the exact cause of the change in yield in the post-silicon domain. Since our post-silicon data is actually simulated, we can verify that this is indeed the correct cause for the observed shift.

However, it may not always be practically possible to identify the exact paramet-

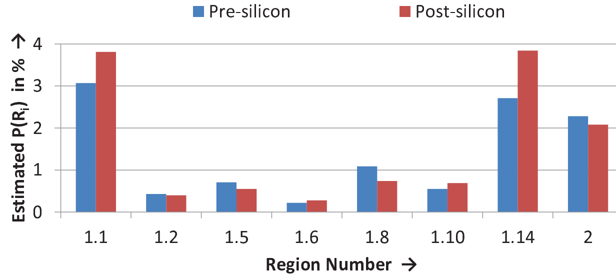


Figure 7.4: PLL2 debug results. Regions $R_{1.1}$ and $R_{1.14}$ show substantial deviations and will be examined further.

ric shift as we did with $PLL1$. We may occasionally have to be content with simply identifying important failure mechanisms and speculating what might be the most "likely" deviation(s) based on the parameter ranking in the pre-silicon domain. The most statistically significant shifts for $PLL2$ are demonstrated by regions $R_{1.1}$ and $R_{1.14}$ in Fig. 7.4. These regions in conjunction with the parameter ranking in Table 4.2 lets us to speculate that the mean value of $C2$ might have gone up or that $LCP5$ may have gone down or both. Since our post-silicon data is actually simulated, we know that an increase in the mean of $C2$ was responsible for this shift. However, being able to shortlist the possible causes of failure down to even two parameters is significant since the designer can just make the circuit more robust to both.

7.3 Summary

As circuit sizes continue to grow, our observability in the post-silicon domain continues to degrade as well. This is especially true with dealing with out-of-specification (soft) faults caused by parametric variations. We thus propose and demonstrate a post-silicon debug flow requiring only as much information as is readily available without the need to add expensive probes / measurement circuitry. While we may

not be able to always identify the exact parametric shift, we are always able to highlight specific failure mechanisms that may have grown in importance in the post-silicon domain.

This information can then be used by the designer in making the circuit more robust. We achieve this by generating failure regions as described in Section 4 and identifying shifts in the importance of these failure mechanisms in the post-silicon domain. Since both critical parameters and interactions with other parameters are known as part of the pre-silicon failure regions, we can use this pre-silicon knowledge to optimize the design around the important failure regions.

The entire process is assisted by the test set selection methodology we detailed in Section 6. However, the process of identifying systematic shifts is independent of the exact test-set selection methodology used, as long as the test-set demonstrates a high probability of exciting failures.

8. CONCLUSIONS AND FUTURE WORK

We described an approximate property checking algorithm in Section 3. We then used the data generated by the algorithm for failure region based diagnosis in Section 4. These failure regions in turn constituted the pre-silicon knowledge that we used for test-set selection and identifying systematic shifts in the probability of these failure regions in Section 6 and Section 7 respectively. In doing so, we effectively leveraged pre-silicon data for post-silicon debug as well.

Last but not the least, we also showed how the pre-silicon aspects of this work (Section 3 and Section 4) can be used for equivalence checking and diagnosing model mismatches in Section 5. Of course if we do manage to build a good system level model that shows good correlation to the component circuits, the contributions listed in Sections 3, 4, 6 and 7 can be applied at a system level as well. Since we are abstracting away a lot of information in building these high level models, such a system will optimally be of similar dimensionality as the circuit level problem we solved here . As a result, we do not specifically demonstrate how Sections 3, 4, 6 and 7 can be applied at a system level.

The key enabler in all of the above is exercising the accuracy vs. turn-around-time trade-off. Instead of trying to aim for exact metrics in any of the methodologies we have detailed during the course of this dissertation, we quantify the confidence in our inexact metrics. In Section 3, we use the failure probability interval instead of the actual failure estimate to make a design decision. In Section 4, we use the importance and confidence of the different failure regions we generate to decide whether or not to address those regions. In Section 6, we simply boost the probability of observing failure mechanisms that depend highly on the controllable parameters. We make no

guarantee as to whether it is possible to do so for all failure regions or not. Similarly in Section 7, we estimate the probabilities for regions that may have been excited in an unexpected fashion by the post-silicon data. We do not attempt to say anything for regions that do not demonstrate any significant deviation.

Of course, if used as an end-to-end solution, this uncertainty is mitigated a whole lot. For example, the parameter selection in Section 4 can be used to assist with designing DfT structures by determining what parameters to control or observe in Section 6 and Section 7 respectively. However, each contribution listed in this dissertation can be used independently of all other contributions as well, leaving a lot of scope for future work that we discuss in the following section.

8.1 Future work

During the course of this dissertation, we have detailed and demonstrated an inexact method for gathering pre-silicon data and using this data in both the pre-silicon and post-silicon domains for the dual purposes of verification and debugging observed failures. However, a lot of scope remains for further research within these individual areas and in figuring out where exactly the contributions detailed herein fit in with other relevant work in the area.

For example, the knowledge representation scheme (BART) that we use in Section 3 and Section 5 for the purpose of failure probability estimation has not been specifically designed to capture non-linear circuit information. It has simply been adapted to the purpose, as a result of which we see a wide variation in quality of results when dealing with different error rates and different circuits. Also, since we deal exclusively with the simulation time in the results we present here, its unclear at what circuit size should we stop approximating and use existing yield estimation techniques to get a more exact solution.

Secondly, the rule-induction techniques we use to build failure regions in Section 4 focuses exclusively on interpretability of said rules. However, we use these same rules for test-set selection and identifying systematic shifts in the importance of these regions in Section 6 and Section 7 respectively. While doing so does not directly hurt us given that overlapping rules still capture highly non-linear concepts, there is a certain amount of generalization and over-fitting happening in the process that might have been mitigated using alternate knowledge representation schemes. It would be interesting to see if using more complex failure manifolds gives better results in downstream activities in the post-silicon domain. However, it also raises the question as to how these more complex regions may be used by the designer for the purpose of fixing the circuit as well.

Additionally, a critical aspect we haven't gone into so far is that the failure regions in Section 4 have been built over the entire parameter space. However, the test plan described in Section 6 may allow only part of this parameter space to be reachable in the post-silicon domain. It may be worthwhile to examine whether generating failure regions that target only the parts of the parameter space that can be reached by the test-plan assists with test set selection or identifying systematic shifts in the importance of these new failure regions. While we do not expect significant advantages in test selection since the failure regions are simply the knowledge representation scheme used to pick the actual tests, given that the regions are being used for actual debug in Section 7, doing so may be beneficial if it reduces the number and complexity of rules that we have to examine.

Of course, all the research vectors we have described above deal with the tools and techniques we have detailed specifically over the course of this dissertation. How they can be used to best effect alongside the large body of work that already exists in all the areas we explore, remains a subject of active research.

Another important observation is that the techniques we describe throughout the course of this dissertation do not take into account any specific topological information. As a result, the contributions we make are not restricted to any specific level of abstraction in the design flow. As long as a quantity affecting the output response can be parameterized, the effect of the same on the circuit can be analyzed for the purposes of failure probability estimation, diagnosis, test or debug. This holds true whether we are analyzing a high level model or a low level SPICE netlist. The only restriction is that the number of parameters cannot grow without bound.

A practical way to deal with this limitation is to perform corner based analysis instead of dealing with all possible transistor mismatch dynamics. We expect failures to be more likely when operating far away from the nominal operating condition that the circuit was designed for. Simulating the furthest such point is thus almost sure to give rise to failures if the output response is sensitive to that parameter. Since corners can be treated as yet another parameter, our contributions can once again be leveraged for the same as well as long as the corner based assumption holds true.

Another way to deal with the number of parameters is to use a hierarchical approach. High level models can be used to predict system-level response while equivalence checking can be used to ensure that these high level models represent circuit-level behavior. Failure probability estimation, diagnosis, test and debug can then be performed at varying levels of abstraction.

This of course opens up a vista of possibilities. Transistor level dynamics can be taken into account when verifying and optimizing smaller blocks. The same transistor level dynamics can be abstracted away and potentially replaced with corners when doing system level analysis. Additionally, what parameters need to be modelled for a given block to enable high-confidence system-level simulation can be guided by the equivalence checking process as well. If it becomes obvious that optimizing a model

to match the circuit response is not possible without adding additional parameters to the high-level model itself, it tells us that the parameter is important for system-level analysis as well. Test and debug can be performed at whatever level of abstraction we may have sufficient controllability and observability at.

The hierarchical approach comes at a cost of course. Every time we abstract away some information, we run the risk of missing out potentially important interaction effects. Additionally, the mismatch tolerance can affect the confidence in the system level error estimate. In essence, there is no free lunch and there is always a trade-off between accuracy and turn-around-time.

Figuring out when to hierarchically decompose and when not to, and what parameters to model at each level, can thus be considered an active research area as well - an area that will no doubt evolve with the scalability of verification techniques. However, this is also an area which requires a significant understanding of the design and functionality of the circuit as well.

To summarize, there are a lot of opportunities worth exploring to further improve as well as apply the contributions listed in this dissertation at both the circuit and system level.

REFERENCES

[1] G. G. Gielen and R. A. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits," *Proceedings of the IEEE*, vol. 88, no. 12, pp. 1825–1854, 2000.

[2] S. Gupta, B. H. Krogh, and R. A. Rutenbar, "Towards formal verification of analog designs," in *Proceedings of the 2004 IEEE/ACM International Conference on Computer-aided design*, pp. 210–217, IEEE Computer Society, 2004.

[3] S. Little, D. Walter, N. Seegmiller, C. Myers, and T. Yoneda, "Verification of analog and mixed-signal circuits using timed hybrid petri nets," in *Automated Technology for Verification and Analysis*, pp. 426–440, Springer, 2004.

[4] W. Hartong, L. Hedrich, and E. Barke, "Model checking algorithms for analog verification," in *Proceedings of the 39th ACM/IEEE Design Automation Conference*, pp. 542–547, ACM, 2002.

[5] R. Kanj, R. Joshi, and S. Nassif, "Mixture importance sampling and its application to the analysis of sram designs in the presence of rare failure events," in *Proceedings of the 43rd ACM/IEEE Design Automation Conference*, pp. 69–72, 2006.

[6] L. Dolecek, M. Qazi, D. Shah, and A. Chandrakasan, "Breaking the simulation barrier: Sram evaluation through norm minimization," in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pp. 322–329, 2008.

[7] S. Sun, Y. Feng, C. Dong, and X. Li, "Efficient sram failure rate prediction via gibbs sampling," *IEEE Transactions on Computer-Aided Design of Integrated*

Circuits and Systems, vol. 31, no. 12, pp. 1831–1844, 2012.

[8] C. Gu and J. Roychowdhury, "An efficient, fully nonlinear, variability-aware non-monte-carlo yield estimation procedure with applications to sram cells and ring oscillators," in *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, pp. 754–761, 2008.

[9] M. H. Kalos and P. A. Whitlock, *Monte carlo methods*. Wiley-VCH, 2008.

[10] G. Fishman, *Monte Carlo: concepts, algorithms, and applications*. Springer, 1996.

[11] B. Efron and B. Efron, *The jackknife, the bootstrap and other resampling plans*, vol. 38. SIAM, 1982.

[12] S. Tiwary, P. Tiwary, and R. Rutenbar, "Generation of yield-aware pareto surfaces for hierarchical circuit design space exploration," in *Proceedings of the 43rd ACM/IEEE Design Automation Conference*, pp. 31–36, 2006.

[13] G. Yu and P. Li, "Yield-aware analog integrated circuit optimization using geostatistics motivated performance modeling," in *Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design*, pp. 464–469, IEEE, 2007.

[14] H. Lin and P. Li, "Classifying circuit performance using active-learning guided support vector machines," in *Proceedings of the 2012 IEEE/ACM International Conference on Computer-Aided Design*, pp. 187–194, IEEE, 2012.

[15] J. Vlach and K. Singhal, *Computer methods for circuit analysis and design*. Springer, 1983.

[16] M. Tian and C.-J. Shi, "Worst case tolerance analysis of linear analog circuits using sensitivity bands," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 8, pp. 1138–1145, 2000.

[17] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer, "High-performance cmos variability in the 65-nm regime and beyond," *IBM Journal of Research and Development*, vol. 50, no. 4.5, pp. 433–449, 2006.

[18] R. Heald and P. Wang, "Variability in sub-100nm sram designs," in *Proceedings of the 2004 IEEE/ACM International Conference on Computer-aided design*, pp. 347–352, IEEE Computer Society, 2004.

[19] D. Sylvester, K. Agarwal, and S. Shah, "Variability in nanometer cmos: Impact, analysis, and minimization," *Integration, the VLSI Journal*, vol. 41, no. 3, pp. 319–339, 2008.

[20] S. Nassif, "Modeling and analysis of manufacturing variations," in *Proceedings of the 2001 IEEE Conference on Custom Integrated Circuits*, pp. 223–228, 2001.

[21] M. Krasnicki, R. Phelps, R. A. Rutenbar, and L. R. Carley, "Maelstrom: efficient simulation-based synthesis for custom analog cells," in *Proceedings of the 36th ACM/IEEE Design Automation Conference*, pp. 945–950, ACM, 1999.

[22] R. Phelps, M. Krasnicki, R. A. Rutenbar, L. R. Carley, and J. R. Hellums, "Anaconda: simulation-based synthesis of analog circuits via stochastic pattern search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 6, pp. 703–717, 2000.

[23] R. A. Rutenbar, G. G. Gielen, and J. Roychowdhury, "Hierarchical modeling, optimization, and synthesis for system-level analog and rf designs," *Proceedings*

of the IEEE, vol. 95, no. 3, pp. 640–669, 2007.

[24] W. Daems, G. Gielen, and W. Sansen, "Simulation-based generation of posynomial performance models for the sizing of analog integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 5, pp. 517–534, 2003.

[25] D. Binkley, C. Hopper, S. D. Tucker, B. C. Moss, J. M. Rochelle, and D. Foty, "A cad methodology for optimizing transistor current and sizing in analog cmos design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 2, pp. 225–237, 2003.

[26] M. Eick and H. E. Graeb, "Mars: Matching-driven analog sizing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 8, pp. 1145–1158, 2012.

[27] L. W. Nagel and D. O. Pederson, *SPICE: Simulation program with integrated circuit emphasis*. Electronics Research Laboratory, College of Engineering, University of California, 1973.

[28] E. Christen and K. Bakalar, "Vhdl-ams-a hardware description language for analog and mixed-signal applications," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 46, no. 10, pp. 1263–1272, 1999.

[29] K. Kundert and O. Zinke, *The designers guide to Verilog-AMS*. Springer, 2004.

[30] D. Potop-Butucaru, C. Lallement, and A. Vachoux, "Vhdl-ams and verilog-ams as alternative hardware description languages for efficient modeling of multidiscipline systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 2, pp. 204–225, 2005.

[31] S. Mukhopadhyay, H. Mahmoodi, and K. Roy, "Modeling of failure probability and statistical design of sram array for yield enhancement in nanoscaled cmos," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, pp. 1859–1880, Dec 2005.

[32] M. Burns and G. W. Roberts, *An introduction to mixed-signal IC test and measurement*, vol. 2001. IET, 2001.

[33] M. Soma, "A design-for-test methodology for active analog filters," in *Proceedings of the 1990 International Test Conference*, pp. 183–192, IEEE, 1990.

[34] K. Arabi and B. Kaminska, "Oscillation-test strategy for analog and mixed-signal integrated circuits," in *Proceedings of 14th VLSI Test Symposium, 1996*, pp. 476–482, IEEE, 1996.

[35] K. Arabi and B. Kaminska, "Oscillation-test methodology for low-cost testing of active analog filters," *IEEE Transactions on Instrumentation and Measurement*, vol. 48, no. 4, pp. 798–806, 1999.

[36] J. A. Starzyk, D. Liu, Z.-H. Liu, D. E. Nelson, and J. O. Rutkowski, "Entropy-based optimum test points selection for analog fault dictionary techniques," *IEEE Transactions on Instrumentation and Measurement*, vol. 53, no. 3, pp. 754–761, 2004.

[37] M. Aminian and F. Aminian, "Neural-network based analog-circuit fault diagnosis using wavelet transform as preprocessor," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 2, pp. 151–156, 2000.

[38] F. Aminian, M. Aminian, and H. Collins Jr, "Analog fault diagnosis of actual circuits using neural networks," *IEEE Transactions on Instrumentation and*

Measurement, vol. 51, no. 3, pp. 544–550, 2002.

[39] V. Prasad and N. S. C. Babu, "Selection of test nodes for analog fault diagnosis in dictionary approach," *IEEE Transactions on Instrumentation and Measurement*, vol. 49, no. 6, pp. 1289–1297, 2000.

[40] G. Devarayanadurg and M. Soma, "Analytical fault modeling and static test generation for analog ics," in *Proceedings of the 1994 IEEE/ACM International Conference on Computer-aided design*, pp. 44–47, IEEE Computer Society Press, 1994.

[41] C. Alippi, M. Catelani, A. Fort, and M. Mugnaini, "Automated selection of test frequencies for fault diagnosis in analog electronic circuits," *IEEE Transactions on Instrumentation and Measurement*, vol. 54, no. 3, pp. 1033–1044, 2005.

[42] T. Golonek and J. Rutkowski, "Genetic-algorithm-based method for optimal analog test point selection," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 2, pp. 117–121, 2007.

[43] N. Nagi, A. Chatterjee, A. Balivada, and J. A. Abraham, "Fault-based automatic test generator for linear analog circuits," in *Proceedings of the 1993 IEEE/ACM International Conference on Computer-aided design*, pp. 88–91, IEEE Computer Society Press, 1993.

[44] S. Mir, M. Lubaszewski, and B. Courtois, "Fault-based atpg for linear analog circuits with minimal size multifrequency test sets," *Journal of Electronic Testing*, vol. 9, no. 1-2, pp. 43–57, 1996.

[45] M. Slamani and B. Kaminska, "Analog circuit fault diagnosis based on sensitivity computation and functional testing," *IEEE Design & Test of Computers*, vol. 9, no. 1, pp. 30–39, 1992.

[46] S. Cherubal and A. Chatterjee, "Parametric fault diagnosis for analog systems using functional mapping," in *Proceedings of the 1999 Design, Automation and Test in Europe Conference and Exhibition*, pp. 195–200, 1999.

[47] C. Alippi, M. Catelani, A. Fort, and M. Mugnaini, "Sbt soft fault diagnosis in analog electronic circuits: a sensitivity-based approach by randomized algorithms," *IEEE Transactions on Instrumentation and Measurement*, vol. 51, no. 5, pp. 1116–1125, 2002.

[48] S. Chakrabarti, S. Cherubal, and A. Chatterjee, "Fault diagnosis for mixed-signal electronic systems," in *Proceedings of the 1999 IEEE Aerospace Conference*, vol. 3, pp. 169–179, IEEE, 1999.

[49] K. Huang, H.-G. Stratigopoulos, and S. Mir, "Fault diagnosis of analog circuits based on machine learning," in *Proceedings of the 2010 Design, Automation & Test in Europe Conference & Exhibition*, pp. 1761–1766, IEEE, 2010.

[50] H. A. Chipman, E. I. George, and R. E. Mcculloch, "Bart: Bayesian additive regression trees," *Annals of Applied Statistics*, vol. 4, no. 1, pp. 266–298, 2010.

[51] M. Stein, "Large sample properties of simulations using latin hypercube sampling," *Technometrics*, vol. 29, no. 2, pp. 143–151, 1987.

[52] T. Hastie and R. Tibshirani, "Bayesian backfitting," *Statistical Science*, vol. 15, pp. 193–223, 1998.

[53] A. Krogh, J. Vedelsby, *et al.*, "Neural network ensembles, cross validation, and active learning," *Advances in Neural Information Processing Systems*, vol. 7, pp. 231–238, 1995.

[54] G. Brown, J. Wyatt, R. Harris, and X. Yao, "Diversity creation methods: a survey and categorisation," *Information Fusion*, vol. 6, no. 1, pp. 5–20, 2005.

[55] T. Blickle and L. Thiele, "A comparison of selection schemes used in genetic algorithms," tech. rep., Gloriastrasse 35, CH-8092 Zurich: Swiss Federal Institute of Technology (ETH) Zurich, Computer Engineering and Communications Networks Lab (TIK, 1995.

[56] J. R. Quinlan, *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., 1993.

[57] W. W. Cohen, "Fast effective rule induction," in *Proceedings of the 12th International Conference on Machine Learning*, 1994.

[58] M. A. Hall, *Correlation-based feature selection for machine learning.* PhD thesis, The University of Waikato, 1999.

[59] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees," *Journal of Artificial Intelligence Research*, vol. 2, pp. 1–32, 1994.

[60] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, pp. 1189–1232, 2000.

[61] L. Breiman, "Bagging predictors," in *Machine Learning*, pp. 123–140, 1996.

[62] L. Breiman, "Random forests," in *Machine Learning*, pp. 5–32, 2001.

[63] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin, *Bayesian data analysis.* CRC press, 2013.

[64] C.-K. Wong and M. C. Easton, "An efficient method for weighted sampling without replacement," *SIAM Journal on Computing*, vol. 9, no. 1, pp. 111–113, 1980.

[65] B. D. Ripley, *Stochastic simulation*, vol. 316. John Wiley & Sons, 2009.

[66] S. Sunter and N. Nagi, "Test metrics for analog parametric faults," in *Proceedings of the 17th IEEE VLSI Test Symposium*, pp. 226–234, IEEE, 1999.

[67] K. Arabi and B. Kaminska, “Parametric and catastrophic fault coverage of analog circuits in oscillation-test methodology,” in *Proceedings of the 15th IEEE VLSI Test Symposium*, pp. 166–171, IEEE, 1997.

[68] B. W. Silverman, *Density estimation for statistics and data analysis*, vol. 26. CRC press, 1986.

[69] Q. Li and J. S. Racine, *Nonparametric econometrics: Theory and practice*. Princeton University Press, 2007.