# ANALOG BEHAVIORAL SIMULATION AND MODELING

by

Edward W.Y. Liu

Memorandum No. UCB/ERL M93/38

25 May 1993

# ANALOG BEHAVIORAL SIMULATION AND MODELING

by

Edward W.Y. Liu

Memorandum No. UCB/ERL M93/38

25 May 1993

**ELECTRONICS RESEARCH LABORATORY**

College of Engineering
University of California, Berkeley
94720

# Abstract

## Analog Behavioral Simulation and Modeling

by

Edward W. Y. Liu

Doctor of Philosophy in Electrical Engineering and Computer Sciences

University of California at Berkeley

Professor Alberto Sangiovanni-Vincentelli, Chair

We propose a top-down, constraint-driven approach to designing complex mixed signal circuits. To support the proposed design methodology, we develop system simulation algorithms and behavioral models for many types of analog systems and components. In analog systems, the nominal circuit functions are usually very simple, and system malfunctions are most often due to second order effects caused by noise and process variations. Therefore, behavioral models at all levels must capture second order effects for constraint translation in top-down design. Using traditional circuit simulation and macromodeling approaches, it is very difficult to simulate frequency domain effects, noise effects, or effects due to process variations because all models are deterministic. As a result, we propose a new strategy for behavioral simulation and modeling for the design and verification of systems in the presence of noise effects and effects due to process variations.

In addition, we propose a behavioral representation for Nyquist data converters. The representation captures the behavior of a memoryless Nyquist data converter, including statistical variations. To describe noise effects, a joint probability density function is used. To describe process variations effects on the converter transfer function, a Gaussian model is used.

Besides applications in verification, the proposed converter behavioral model also provides critical information for design engineers to evaluate the testability of the design at an early design stage and for test engineers to choose the optimum testing strategy after design. To achieve that goal, we propose a strategy for testing all DC performance of Nyquist data converters including offset error, full scale gain error, integral nonlinearity, and differential nonlinearity.

Finally, we focus on noise modeling and simulation for mixed-mode sampled-data systems. We present a "direct" noise analysis approach for mixed mode systems, and compare our approach with the traditional Monte Carlo approach. The approach is approximate and computes noise effects by performing arithmetic on a finite number of moments of distribution functions that characterize electronic noise. One key advantage of this approach is its ability to compute low error probabilities.

Prof. Alberto Sangiovanni-Vincentelli
Thesis Committee Chairman

# Acknowledgments

I would like to thank Prof. Alberto Sangiovanni-Vincentelli and Prof. Paul Gray for their guidance during the course of my Ph.D. studies. I am very fortunate to have been able to work with such outstanding people. My research advisor, Prof. Sangiovanni-Vincentelli, is the smartest and most articulate professor I have ever met. He instilled in me the importance for mathematical rigor in doing my research and good communication skills in promoting my research. The technical, writing, and presentation skills he taught me will certainly be invaluable for the rest of my life. Prof. Gray is the "prince" of professors. Everyone I know who has interacted with him respects him for his integrity and grace. He instilled in me the importance of quality, rather than quantity, in doing research.

I would also like to thank Prof. David Brillinger and Prof. Robert Brayton for teaching me various graduate courses and serving on my qualifying committee.

Of course, many friends deserve special thanks. I was told that it is more polite to list one's friends in alphabetical order. So, many thanks to Wendell Baker, for help with C++; Felice Balarin (my cubicle mate), for reviewing my papers, giving me advice in research and real life, and putting up with the mess in my cubicle; Mansun Chan, for introducing me to ballroom dancing, an activity that I will enjoy for the rest of my life; Henry Chang (my research partner), for being a superb collaborator in many research projects; Eduardo Charbon, for fruitful research discussions; Thomas Cho, for being my prelim exam study mate and good friend; Umakanta Choudhury; Robert Chu, for discussion on analog design; David Cline, for organizing ski trips that enrich our grad student life; Cormac Conroy, for his help in numerous things; Alper Demir, for being a good colleague; Eric Felt, for being a friend whose charisma I admire; David Gates, for helping me out when we were co-TAing EECS 105 and

writing our thesis; Georges Gielen, for being an excellent research collaborator; Ramin Hojati, for lending me books; Harry Hsieh, for popcorn and snacks late at night; Kelvin Hui, for being a friend who made my life enjoyable through bridge and majong games; Gani Jusuf, for being a good friend ever since undergraduate days; Beorn Johnson, for help with SPICE and numerical analysis; Kate and Tim Kam, for their friendship and hospitality; Cindy Keys, for fruitful discussions on Volterra series; Brad Krebbs, for help in computers; William Lam, for being my closest friend ever since we returned from Stanford; Brian Lee, for help with C++; Kathy Lu, for being nice to me; Enrico Malavasi, for informative research discussion; Elise Mills, for making sure I get paid; Dr. Linda Milor, for help in my 219 project and discussion about analog CAD; Rajeev Murgai (who's on Indian time), for chatting and discussions late into the night and sometimes until dawn; Robert Neff, for helpful discussions on D/A converters; Keith Onodera, for organizing ski trips and dinners that enrich our grad student life; Flora Oviedo (the nicest person in the cadgroup), for help in administrative matters; Chris Rudell, for tips on skiing; Jagesh Sanghavi (the entrepreneur in the group), for interesting discussion about business; Dr. Hamid Savoj, for helping me in my 290LS project; Henry Sheng, for teaching me Monte Carlo methods; Narendra Shenoy, for helpful discussions on linear programming; Tom Shiple, for being a friend to chat with during SRC and ILP conferences; Dr. K. J. Singh; Krishnan Sriram; Eric Tomacruz; Greg Uehara (the analog design guru), for his helpful criticism of my talks; Huey-Yih Wang, for being a good friend and teaching me about synthesis and verification; Caesar Wong, for being a close friend who made my life enjoyable through bridge and majong games; Naiyavudhi Wongkomet (Tom), for interesting discussions about Thailand and career goals; Darrin Young (who's crazy about poles and zeros), for interesting discussions about micro-machines and sigma-delta converters.

Finally, I like to thank my parents, Kam Ha and Robert, my sisters, Eva, Helen, and Nancy, and my brother, Alex, for their support and encouragement.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Top-Down Constraint-Driven Design Methodology

System design and verification using traditional simulators such as SPICE is often impossible due to the long simulation times. Because the inner loops of circuit simulators are linear equation solvers, the simulation time is estimated to increase as $O(n^{1.5})$[44], where $n$ is the number of nodes in the circuit. Besides, as the circuit size increases, more simulations are required to estimate *system performances* affected by noise or process variations.

To circumvent the design and verification problems associated with traditional simulators, we propose a **top-down constraint-driven** approach[10, 11] to designing complex mixed signal circuits, where abstraction and successive design refinement are key. Previously, several design methodologies[60, 25] have been proposed for high level synthesis of digital circuits. For example, the System Architect's Workbench[60] takes as input a behavioral description of a system to be designed, along with a set of constraints, and produces a set of register-transfer modules and a control sequence table. The ADAM[25] system is designed to unify a number of design automation programs into a single framework. The goals of ADAM are to produce correct, testable implementations representing a range of tradeoffs, to allow varying degrees of user interaction, to allow design to proceed incrementally starting from a partially complete initial design, and to meet several kinds of constraints. ADAM uses both hard-coded and knowledge-based techniques to achieve these goals. In the analog domain, OASYS[22] is a hierarchically structured framework for analog circuit synthesis. Analog circuit

topologies are represented as a hierarchy of templates of abstract functional blocks each with associated detailed design knowledge. Mechanisms are described to select from among alternate design styles, and to translate performance specifications from one level in the hierarchy to the next lower, more concrete level.

In contrast, our methodology assists the hierarchical generation of the design by providing a rigorous procedure based on interactive and automatic tools. Given a set of circuit specifications (circuit characteristics, design rules, technology information, operating specifications, etc.), a mapping is made to schematics or to layout.

Given a library of $n$ architectures, the first operation that must be performed is architectural selection. Simulators and optimizers are used to aid the decision-making process. For high-level blocks, a behavioral simulator may be employed. For low-level circuits a circuit simulator such as SPICE may be run. For an architecture where no simulators exist (e.g. a pre-made cell) the "simulator" could just be a list of performance specifications. If a suitable architecture cannot be found, then this selector must return to the upper node the fact that the selection has failed. If a standard cell (pre-designed cell) was chosen then a successful return to the upper node is made.

Given the set of specifications, $S$, for the particular architecture chosen we proceed to map $S$ to a set of specifications, $T$, for each of the component blocks or subsystems. At the higher levels of the decomposition when the architectural details of the subsystems have yet to be determined the goal of the mapping is to choose $T$ constrained by $S$ in such a way as to maximize **flexibility** that is a function of the degree of freedom in designing the subsystems. Currently, heuristics are used to characterize this flexibility as a function of $T$. Once the problem has been set up, it is given to a nonlinear optimizer where the constraints are evaluated using the behavioral simulator. The user can, of course, also do the partitioning. If this problem is not found to be feasible, another architecture must be chosen.

These subsystems are then expanded in the same manner, thus recursively expanding the design hierarchy until the complete architecture has been determined (i.e. a full schematic is available). Then these subsystems return a set of actual component specifications based on the full schematic. If the returned specifications fail to meet the criteria, $T$, set by the mapping function, and combined fail to meet $S$, as evaluated by the behavioral simulator, then the flow of control is returned to the

mapping function where a new mapping can be attempted. If all is successful, then two options are available. We can either proceed with the layout or stop here returning only the full schematics.

In creating the layout, the first step is the generation of the constraints for the assembly. This can be done by the user, or accomplished automatically with design tools. As before, if this step fails, the flow control is returned to the mapping function. If it is successful, then constraint-driven physical assembly is performed. If this fails to meet its constraints then an alternate set of assembly constraints is derived. If the physical assembly is successful (i.e. the entire chip, system, or subsystem has been routed and is ready for use), then only the final verification step remains. This step includes full extraction of the circuit including all relevant parasitics. Simulations are then performed with the data using the same simulator used in the architectural selection phase, but this time with the incorporation of all of the extracted layout parasitics to verify subsystem and system level performs in the presence of parasitics. Finally, a summary of the expected performances is generated. If all of the specifications are met, the flow control is returned to the upper node, and the design is complete.

This final verification/extraction phase also aids in testing by sending information to a database which can be used later by the tester. It is also at this stage where additional hardware can be considered to ease the testing or even alternative architectures can be suggested which are less costly to test. This information can be fed back to the mapping function, or even to the architectural selection function.

We believe that our design methodology is significantly different from the ones currently employed by circuit designers. Today, a typical design cycle starts with a set of specifications for an integrated circuit drawn in conjunction with the customer. The designer takes these specifications and performs a first level decomposition based on simulations for nominal behavior. The partitioning is accomplished based mostly on the experience of the designers. Typically, today, designers then resort to a bottom-up approach. Low level circuits are built, tested, verified, and assembled hierarchically from the bottom-up until the first level decomposition blocks are reached. The main problem with this design technique is that if the final blocks fail to meet specifications, the entire circuit has to be redesigned, possibly all the way from the bottom again. This can be very time consuming and costly. A typical solution to this problem is

designing with overconstraints on the lower blocks. This, however, is also costly, because a non-optimal solution is usually reached. Our methodology attempts to prevent these problems. We use behavioral simulations for early verification and design space exploration to leverage the expertise of the designer at its best. Each of the lower blocks is constrained as the tree is descended to match the performance specifications. Thus, at any time, we are reasonably certain that the original specifications are being met. We need not go to the bottom of our design and back up before realizing that a costly mistake has been made. Due to simplifying abstractions such as no wiring and parasitics capacitances and resistances made during top-down design, we perform a final bottom-up verification step in which the circuit is extracted and simulated in the presence of all parasitics.

## 1.2 Behavioral Simulation for Top-Down Design

As shown in Section 1.1, our design methodology rely on accurate circuit modeling at each step of design refinement and the ability to propagate constraints step by step. As a result, we do not need to verify top level system performance from low level implementation details such as SPICE simulation of the entire system. Rather, we need to guarantee correct translation of one level of specification constraints to a set of lower level specification constraints.

In analog systems, the nominal circuit functions are usually very simple, and system malfunctions are most often due to second order effects caused by noise and process variations. As a result, system constraints are usually specified in terms of the maximum amount of second order effects allowed such as signal-to-noise ratio and total harmonic distortion. In turn, component constraints are usually specified in terms of basic statistical effects such as random offsets and mismatches. Therefore, *behavioral models at all levels must capture second order effects for constraint propagation in top-down design.*

The tools for constraint translation are the **behavioral models** at each level, the **behavioral simulator**, and a local **selector** or **designer**, which will produce specification constraints for the lower levels, using the behavioral simulator. A mixed analog/digital system is represented by a hierarchy of behavioral models as shown in Figure 1.1 in which the level of abstraction increases from the implementation details

at the bottom to an abstract mathematical model at the top. Each data node in this **model hierarchy** is a *mathematical* behavioral representation. For the bottom levels, the simulations can use SPICE, logic simulators, or other hardware specific simulators. For higher levels, the simulations are behavioral and not hardware specific.

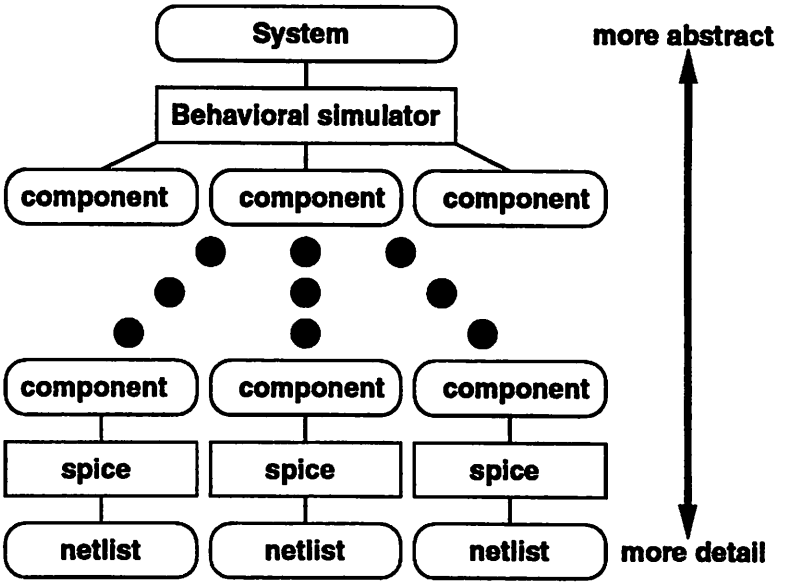


Figure 1.1: Model Hierarchy

To verify the design, we propose a final **bottom-up verification** step. The first phase in this step is **component extraction** where a schematic with parasitic resistances and capacitances is produced from layout. Then, in the **component identification/data fitting** phase, performances of individual circuit components will be determined using SPICE simulation. The performances are analyzed through a series of data analysis to fit parameters for behavioral models of the components. This identification/data fitting procedure is then repeated for components higher in the hierarchy until the top level system is verified.

## 1.3 Organization of thesis

Behavioral simulation algorithms and behavioral representations for many types of mixed analog/digital circuits have been researched[33, 31, 30, 32]. This thesis presents *a behavioral representation for the class of Nyquist data converters, an analog*

*system verification strategy using behavioral simulation, a testing strategy for data converters using behavioral simulation,* and *noise models and simulation algorithms for mixed-mode sampled-data systems.* With the experience gained from the work presented in this thesis, we plan to develop a comprehensive software environment for the behavioral modeling and simulation of mixed-mode systems.

Previous work in circuit simulation and modeling is presented in Chapter 2, followed by some background material on statistics in Chapter 3. Then, a behavioral representation for the class of Nyquist data converters is presented in Chapter 4, followed by an analog system verification strategy using behavioral simulation (Chapter 5), a testing strategy for data converters using behavioral simulation (Chapter 6), and noise models and simulation algorithms for mixed-mode sampled-data systems (Chapter 7). Finally, a conclusion is presented in Chapter 8.

## 1.4 Conclusion

System design and verification using traditional simulators such as SPICE is often impossible due to the long simulation times. To circumvent the design and verification problems associated with traditional simulators, we propose a **top-down, constraint-driven** approach[10, 11] to designing complex mixed signal circuits, where abstraction and successive design refinement are key. To support the proposed top-down design methodology, we are developing system simulation algorithms and behavioral models for many types of analog components such as converters, phase-locked loops, and filters that capture second order effects.

# Chapter 2

# Previous Work

## 2.1 Circuit simulation

Most popular circuit simulators today are based on SPICE[44] developed at U. C. Berkeley. The user inputs a **circuit netlist** that describes the connectivity of the components. The components are modeled with a network of basic circuit elements such as inductors, capacitors, resistors, and controlled sources that are described by differential equations. The circuit is simulated either in the time or frequency domains. In time domain simulation, the set of nonlinear differential equations are solved numerically given some initial conditions. In frequency domain simulation, the circuit is linearized at the operating point and the resulting (frequency dependent) linear equations are solved for the small signal circuit behavior as a function of frequency. The SPICE simulator analyze analog circuit noise in the frequency domain. SPICE linearizes the circuit at the operating point, adds sinusoidal sources in parallel to the noisy elements, and analyze the resulting AC equivalent circuit.

Although small circuits can be verified using circuit simulation, verification of larger circuits is infeasible due to the need for longer simulation time. Because the inner loops of circuit simulators are linear equation solvers, the simulation time is estimated to increase as $O(n^{1.5})$[44], where $n$ is the number of nodes in the circuit. Besides, as the circuit size increases, more simulations are required to estimate *system performances* affected by noise or process variations.

To address the problem of long simulation time, a new generation of circuit simulators have been developed in the 1980's. Rather than solving the linear equa-

tions directly, simulators such as SPLICE1[47], solves the equations *iteratively* using relaxation techniques. The idea of the iterative method can also be applied at the waveform level (voltage as a function of time). In simulators such as RELAX2.1[61], initial guesses are made for waveforms on each circuit node, then the waveforms are modified iteratively until they converge to the correct waveforms. Although iterative solutions provide good performance for specific types of circuits such as digital MOS circuits, they are deficient for analog simulation because of tightly coupled feedbacks in analog circuits. The reason is that although each iteration is fast, a large number of iterations are needed for convergence in tightly coupled circuits.

## 2.2 Macromodeling

### 2.2.1 Macromodeling for circuit simulation

One approach to address the problem of long simulation time is macromodeling in circuit simulation[13, 40, 2, 36, 4, 18, 56]. **Macromodels** constructed from a set of basic circuit elements in circuit simulators such as resistors, capacitors, inductors, and controlled sources approximate the transient behavior of circuits. Because a macromodel is simpler than the original circuit, simulation time is reduced at the expense of accuracy.

For example, the operational amplifier (opamp) circuit shown in Figure 2.1 is modeled by the macromodel[8] shown in Figure 2.2. The opamp with nine nonlinear transistors is modeled by a simpler network of a nonlinear-controlled current source, a DC voltage source, a resistor, and a capacitor. The nonlinear-controlled current source, $gm1$, represents the transconductance of the nonlinear input differential pair. The DC voltage source sets the DC output bias. The resistor, $rlm$, and capacitor, $clm$, represent the output resistance and capacitance at the output node, respectively.

The elements values are tuned to minimize the difference between macromodel and actual circuit outputs. In [8], the difference is defined as

$$c(m, i_w) = \frac{1}{2} \int_0^T \| v_2(t) - v_1(t) \|^2 \, dt \tag{2.1}$$

where $m$ is a macromodel, $i_w$ is an input waveform, $T$ is the time duration for the comparison, $v_2$ is the output from the macromodel $m$, and $v_1$ is the output from the

actual circuit. This difference makes it possible to quantify the accuracy of a macromodel. In particular, the best macromodel in a collection $M$ is the one that minimizes such a difference for worst case input. Mathematically, the task of finding the optimal macromodel is equivalent to solving a min-max problem

$$\min_{m \in M} (\max_{i_w \in U} c(m, i_w)) \tag{2.2}$$

where $M$ is a collection of macromodels and $U$ is a set of input waveforms.

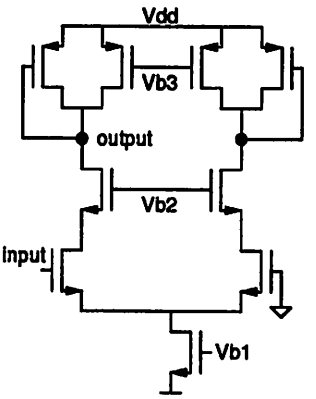


Figure 2.1: Schematic of a CMOS operational amplifier

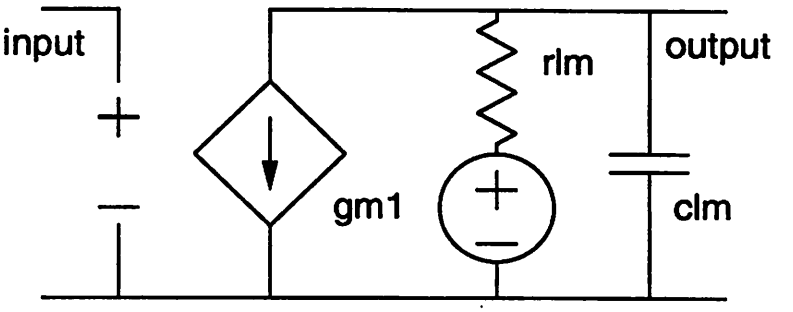


Figure 2.2: Macromodel of a CMOS operational amplifier

The architecture of macromodels are developed by designers based on experience. Macromodel component values are tuned either manually or automatically. Manually tuned macromodels have been available for SPICE for specific types of circuits such as opamps by Boyle[4], comparators by Getreu[18], and phase-locked loops by Tan[56]. In these cases, the macromodel values are hand tuned to minimize the error. As a result, the accuracy of the macromodel can neither be quantified nor controlled. On the other hand, Casinovi[8] proposed algorithms for solving (2.2) based on

an extension of the Hamiltonian formulation of a classical optimal control problem. In this approach, the accuracy of the macromodel can be quantified and controlled.

Macromodeling has been implemented in commercial circuit simulators such as PSPICE[13] and HSPICE[40]. In PSPICE, the "Analog Behavioral Modeling" (macromodeling) option allows for flexible description of electronic components in terms of a transfer function described by a formula or table. For non-linear components, the transfer function describes the instantaneous relation between the input and output. For linear components, the transfer function describes the behavior in the frequency domain. This macromodeling capability is simply an extension of the basic nonlinear controlled voltage and current sources in traditional circuit simulators.

In HSPICE[40], macromodels offer a higher level of abstraction and a speedup over the lower level description of an analog function. As in PSPICE, these elements are voltage or current sources described by functions. In HSPICE, the functions can include nodal voltages, element currents, time, or user defined parameters.

### 2.2.2 Macromodeling for Mixed Analog/Digital Simulators

Simulators such as [17, 50, 9] have a simulation engine for mixed analog/digital circuits, as well as a more flexible macromodeling capability. The SABER simulator[17] uses the MAST language as input. The circuit is described as a network of templates using MAST. Each template defines a component's behavior using differential equations or equations relating timing of events. The interface nodes satisfy Kirchoff's voltage law (KVL) and Kirchoff's current law (KCL), thus loading between components are handled. As a result, each template is a user defined macromodel similar to traditional macromodeling. The difference is that while a macromodel is defined by a network of basic components, a template is defined by a set of differential equations. However, this distinction is largely irrelevant from a mathematical point of view and is only related to the formalism used to input data.

To illustrate modeling using SABER, the equations representing a comparator[36] is presented below,

$$I_{M1} + I_{M2} = C_T \frac{dV_S}{dt} + \frac{V_S}{R} + I_{BIAS} \tag{2.3}$$

$$F(I_{M1}, I_{M2}) - G_1(V_1) = C_1 \frac{dV_1}{dt} \tag{2.4}$$

$$G_2(V_1) - G_3(V_0) = C_L \frac{dV_0}{dt} \tag{2.5}$$

where

$$F(I_{M1}, I_{M2}) = I_{M1} - I_{M2} + k(I_{M1} + I_{M2}) \tag{2.6}$$

$$I_{M1} = f(V_{in1}) \tag{2.7}$$

$$I_{M2} = f(V_{in2}) \tag{2.8}$$

and the functions $G_1$, $G_2$, and $G_3$ are nonlinear functions whose values are stored in tables. The output voltage is $V_0$. To show the fact that modeling in SABER is equivalent to macromodeling, a traditional macromodel representing the same circuit is shown in Figure 2.3.

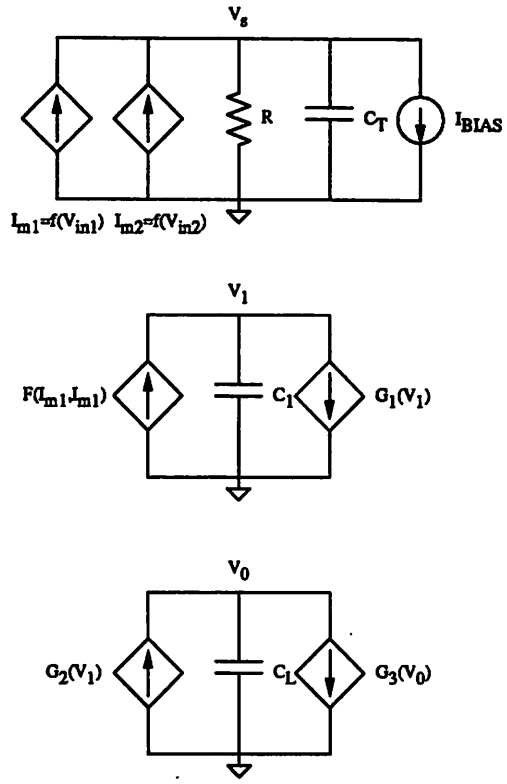


Figure 2.3: Schematic of a comparator macromodel

iMACSIM[50] is a multi-level, mixed-domain simulator. At the highest level is "behavioral" simulation which is used when the function of a block is known, but its detailed structure is undefined yet. The individual blocks can be described in terms of s-domain or z-domain transfer functions and their interaction described using signal

flow diagrams that include summers, multipliers, etc. Blocks can also be described in the "C" language. At the functional level, macromodels such as switches and controlled sources may be used. At the circuit level, electrical simulation is performed using a relaxation based method[48].

### 2.2.3 Limitations of Circuit Simulation and Macromodeling

Although macromodels reduce simulation time, they are not satisfactory to solve analog system problems. For example, the element values for the macromodel shown in Figure 2.2 are optimized for specific input waveforms and output responses. If we desire to model different responses (e.g. opamp slewing) under a new set of input conditions (e.g. fast switching), then the macromodel accuracy is not guaranteed. Error bounds cannot be established during model construction because inputs are unknown.

In addition, a macromodel accurate for a variety of input conditions typically has from a third to a half of the number of elements of the original circuit. Consequently, the simulation time reduction is small. Moreover, great expertise is needed to devise the macromodel. Also, there are limited tools for extracting parameters for macromodels. Casinovi[8] proposed a methodology for automatic macromodel construction and Ma[34] presented model generation and validation tools for iMACSIM.

Furthermore, using the circuit simulation and macromodeling approaches, it is very difficult to simulate frequency domain effects, noise effects, or effects due to process variations. The reason is that the models are *deterministic* and focus on the time domain circuit behavior. Yet, verification of system performance in the presence of noise and process variation effects are crucial.

## 2.3 Special Purpose Simulators

### 2.3.1 Simulators for switched-capacitor networks

For simulating specific classes of circuits such as switched-capacitor networks, special purpose simulators[15, 35] are more efficient than general purpose circuit simulators. SWITCAP[15] is a program for the exact analysis of *linear* networks containing ideal capacitors, independent and dependent voltage sources, and switches

from which macromodels of components are constructed. Its applications include simulation of switched-capacitor filters and charge redistribution systems, such as several types of A/D and D/A and PCM encoders and decoders[39, 58]. The simulation algorithm is based on charge conservation equations before and after switching states and Kirchhoff's voltage law (KVL). As a result, the network variables to be solved for are the voltages and charges. The network equations are formed from the network topology and element values. Then, the network equations can be solved in the time or frequency domain.

The advantage of SWITCAP is its ability to handle both frequency and time domain analyses. The disadvantage, however, is that the network must be linear. Key elements of switched-capacitor circuits such as opamps and comparators are nonlinear. Moreover, only primitive elements are available, so the task of circuit modeling falls entirely on the user. Finally, process variations are not considered.

### 2.3.2 Simulators for sampled-data systems

MIDAS[62] is a behavioral simulator for mixed-mode sampled-data systems. In MIDAS, a sampled-data system is modeled as a graph of primitive component models such as comparators, delays, and adders. Each cycle in the graph must be broken by at least one delay element. After initialization, each component is executed according to their topological order in the graph. At the end of the time domain simulation, special components are available for data analysis such as spectral estimation.

The disadvantages of MIDAS are that couplings between components are not considered, continuous-time domain simulations are not supported, and process variations are not considered. Also, noise effects are estimated only using Monte Carlo techniques which may be inadequate for estimation of rare events (Chapter 7).

## 2.4 Conclusion

Traditional circuit simulators are inadequate for large analog circuits due to the long simulation time. Using the circuit simulation and macromodeling approaches, it is very difficult to simulate frequency domain effects, noise effects, or effects due to process variations because all models are deterministic. Special purpose simulators

are more efficient than general purpose circuit simulators. Some[35] handle frequency domain and noise simulations, while others[62] perform only time domain simulations and rely on spectral estimation techniques and Monte Carlo simulations for estimating frequency response and noise effects, respectively. But, none considers process variation effects. As a result, a new strategy for behavioral simulation and modeling is necessary to verify system performance in the presence of noise effects and effects due to process variations.

# Chapter 3

# Thesis Background

## 3.1 Introduction

This chapter provides some statistical background required to appreciate the contents of the succeeding chapters which constitute the main contribution of this thesis. An overview of sample space, events, probability, random variables, jointly distributed random variables, random vectors, and random processes will be provided below.

## 3.2 Sample Space and Events

Consider an experiment whose outcome is unpredictable. The set of all possible outcomes is known as the **sample space**[45] denoted by $S$. For example, the sample space for the experiment of a roll of a die is

$$S = \{1,2,3,4,5,6\} \tag{3.1}$$

An **event** is any subset $E$ of the sample space, and $E$ occurs if the outcome of the experiment is contained in $E$. For example, the event for odd outcomes in the previous experiment is

$$E = \{1,3,5\} \tag{3.2}$$

The event $E$ occurs if the outcome of the experiment is 1, 3, or 5.

## 3.3 Axioms of Probability

Following [45], we use an axiomatic definition of probability. Consider an experiment whose sample space is $S$. For each event $E$ of the sample space, we define a number $P(E)$, the **probability** of event $E$, that satisfies the following three axioms.

**Axiom 3.3.1** $0 \leq P(E) \leq 1$

**Axiom 3.3.2** $P(S) = 1$

**Axiom 3.3.3** *For any sequence of mutually exclusive events* $E_1, E_2, \ldots$

$$P(\bigcup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} P(E_i) \tag{3.3}$$

The axioms are simple and intuitive. For example, axiom 3.3.1 states that $P(E)$ is between 0 and 1. Axiom 3.3.2 states that the outcome must be in the sample space. Axiom 3.3.3 states that for any sequence of mutually exclusive events the probability of at least one of these events occuring is the sum of their respective probabilities.

A more intuitive meaning of probability is that $P(E)$ is the limiting percentage of event $E$ occuring as the experiment repeats indefinitely.

$$P(E) = \lim_{n \to \infty} \frac{n(E)}{n} \tag{3.4}$$

where $n(E)$ is the number of times in the first $n$ repetitions of the experiment that the event $E$ occurs. The definition is justified in [45] using the Strong Law of Large Numbers and Axioms 3.3.1, 3.3.2, 3.3.3.

## 3.4 Random Variable

Often we are interested in some numerical value associated with experimental outcomes. Therefore, we define a **random variable** as a real-valued function with domain the sample space, $S$. For example, we may be interested in the numerical outcomes of rolling dice. So, we define a random variable, $Y$, that takes on values $\{1, 2, 3, 4, 5, 6\}$. The variable takes on real value $i$ when the outcome of the experiment is $i$. Assuming a fair die, each possible outcomes are equally likely, so the probabilities $P(Y = i) = \frac{1}{6}$.

In the previous example, the random variable is discrete because its set of possible values is either finite or countably infinite. However, there are random variables whose possible values are not countable. We take the definition for a **continuous random variable** from [45],

**Definition 3.4.1** *$X$ is a continuous random variable if there exists a nonnegative function, $f$, defined for all real $x \in (-\infty, \infty)$, having the property that for any set $B$ of real numbers*

$$P\{X \in B\} = \int_B f(x)dx \tag{3.5}$$

The function, $f$, is called the **probability density function** of the random variable, $X$. According to (3.5), the probability that $X$ will be in $B$ can be obtained by integrating the probability density function over the set $B$. For example,

$$P\{a \leq X \leq b\} = \int_a^b f(x)dx \tag{3.6}$$

The probability density function completely characterizes the random variable because all probability statements about $X$ can be answered in terms of $f$. For example, the **cumulative distribution function**[45], $F(\cdot)$, of a random variable, $X$, is defined for all real numbers $b, -\infty < b < \infty$, by

$$F(b) = P\{X \leq b\} = \int_{-\infty}^b f(x)dx \tag{3.7}$$

The **expected value** of $g(X)$, where $g(\cdot)$ is any real-valued function is

$$E[g(X)] = \int_{-\infty}^{\infty} g(x)f(x)dx \tag{3.8}$$

$E[g(X)]$ gives the limiting value of $g(X)$ as the experiment repeats indefinitely. The $n^{th}$ **order moment** is

$$E[X^n] = \int_{-\infty}^{\infty} x^n f(x)dx \tag{3.9}$$

The **mean** of random variable, $X$, is the first order moment

$$E[X] = \int_{-\infty}^{\infty} xf(x)dx \tag{3.10}$$

The **variance** of random variable, $X$, is

$$var(X) = E[X^2] - E[X]^2 \tag{3.11}$$

### 3.4.1 Normal Random Variable

A random variable used throughout the later chapters is the **normal**, or **Gaussian, random variable**. A random variable, $X$, is normally distributed, with parameters $\mu$ and $\sigma$ if the probability density function of $X$ is given by

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}, -\infty < x < \infty \tag{3.12}$$

The cumulative distribution function of a normal random variable is

$$F(b) = \int_{-\infty}^{b} f(x)dx = Q\left(\frac{b-\mu}{\sigma}\right) \tag{3.13}$$

where $Q(\cdot)$ is a standard mathematical function defined as

$$Q(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du \tag{3.14}$$

## 3.5 Jointly Distributed Random Variables

Following [45], we define, for any two random variables $X$ and $Y$, the **joint cumulative probability distribution function** of $X$ and $Y$ by

$$F(a,b) = P\{X \leq a, Y \leq b\}, \quad -\infty < a,b < \infty \tag{3.15}$$

We say that $X$ and $Y$ are jointly continuous if there exists a function $f(x,y)$ defined for all real $x$ and $y$, having the property that for every set $C$ of pairs of real numbers

$$P\{(X,Y) \in C\} = \int\int_{(x,y)\in C} f(x,y)dxdy \tag{3.16}$$

The function $f(x,y)$ is called the **joint probabilitiy density function** of $X$ and $Y$. The joint probability density function completely characterizes the random variables $X$ and $Y$ because all probability statements about $X$ and $Y$ can be answered in terms of $f$. In [45] it is shown that the joint probability density function and joint cumulative distribution function are related by

$$f(a,b) = \frac{\partial^2}{\partial a \partial b} F(a,b) \tag{3.17}$$

The probability density functions of $X$ and $Y$ can be obtained using

$$f_X(x) = \int_{-\infty}^{\infty} f(x,y)dy \tag{3.18}$$

$$f_Y(y) = \int_{-\infty}^{\infty} f(x,y)dx \tag{3.19}$$

The **expectation** of a function $g(X,Y)$ is given by

$$E[g(X,Y)] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} g(x,y)f(x,y)dxdy \tag{3.20}$$

The covariance of random variables $X$ and $Y$, denoted by $Cov(X,Y)$, is defined as

$$Cov(X,Y) = E[(X - E[X])(Y - E[Y])] = E[XY] - E[X]E[Y] \tag{3.21}$$

From [45], if $X$ and $Y$ have a joint probability density function $f(x,y)$, then the **conditional probability density function** of $X$, given that $Y = y$, is defined for all values of $y$ such that $f_Y(y) > 0$, by

$$f_{X|Y}(x|y) = \frac{f(x,y)}{f_Y(y)} \tag{3.22}$$

The use of conditional densities allows us to define conditional probabilities of events associated with one random variable when we are given the value of a second random variable.

The random variables in a joint distribution can be neither jointly continuous nor jointly discrete. For example, $X$ can be a continuous random variable with density function $f_X$, while $Y$ can be a discrete random variable. The joint density function is $f(X,Y)$. In this case, the conditional density of $X$ given that $Y = y$ is given in [45] by

$$f_{X|Y}(x|y) = f(X, Y = y) = \frac{P\{Y = y|X = x\}}{P\{Y = y\}} f_X(x) \tag{3.23}$$

### 3.5.1 Independent Random Variables

The continuous random variables $X$ and $Y$ are said to be **independent**[45] if

$$f(x,y) = f_X(x)f_Y(y) \tag{3.24}$$

in which case for any functions $h$ and $g$

$$E[g(X)h(Y)] = E[g(X)]E[h(y)] \tag{3.25}$$

and $Cov(X,Y) = 0$ from (3.21).

## 3.6 Random Vector

Whereas scalar random variables take on values on the real line (Section 3.4), the values of vector-valued random variables are points in a real-valued higher dimensional space ($R_m$)[49]. In Section 3.5 we considered the case for $m = 2$. Now we consider the case for $m > 2$. The probability law for vector-valued random variables is specified in terms of a joint cumulative distribution function

$$F_{X_1,\ldots,X_m}(x_1,\ldots,x_m) = P\{(X_1 \leq x_1)\ldots(X_m \leq x_m)\} \tag{3.26}$$

The joint probability density function, $f_{X_1,\ldots,X_m}(x_1,\ldots,x_m)$, of an $m$-dimensional random vector is the partial derivative of the cumulative distribution function,

$$f_{X_1,\ldots,X_m}(x_1,\ldots,x_m) = \frac{\partial^m}{\partial x_1 \ldots \partial x_m} F_{X_1,\ldots,X_m}(x_1,\ldots,x_m) \tag{3.27}$$

Expected values of a function of the random vector are evaluated using multiple integrals. For instance, if $m = 4$,

$$E[g(X_1,X_2,X_3,X_4)] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_{X_1,\ldots,X_m}(x_1,\ldots,x_m)dx_1dx_2dx_3dx_4 \tag{3.28}$$

For convenience, the $m$ variables can be represented as components of an $m$ x 1 column vector $X$,

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \tag{3.29}$$

The **mean vector**, $\mu_X$, is defined as

$$\mu_X = E[X] = \begin{bmatrix} E[x_1] \\ \vdots \\ E[x_m] \end{bmatrix} \tag{3.30}$$

The **covariance** matrix, $\Sigma_X$, an $m$ x $m$ matrix is defined as

$$\Sigma_X = E[XX^T] - \mu_X\mu_X^T = \begin{bmatrix} \sigma_{x_1x_1} & \sigma_{x_1x_2} & \cdots & \sigma_{x_1x_m} \\ \sigma_{x_2x_1} & \sigma_{x_2x_2} & \cdots & \sigma_{x_2x_m} \\ \vdots & & & \\ \sigma_{x_mx_1} & \sigma_{x_mx_2} & \cdots & \sigma_{x_mx_m} \end{bmatrix} \tag{3.31}$$

where $\sigma_{x_ix_j} = Cov(x_i,x_j)$. If $\Sigma_X$ is diagonal, then the components of $X$ are **uncorrelated**.

### 3.6.1 Multivariate Normal Distribution

A random vector used throughout the later chapters is the **normal random vector**, or multivariate normal random variable. A random vector, $X$, is normally distributed with parameters $\mu_X$ and $\Sigma_X$,

$$X \sim normal(\mu_X, \Sigma_X) \tag{3.32}$$

if the probability density function of $X$ is given by

$$f_X(x) = \frac{1}{(2\pi)^{\frac{m}{2}}|\Sigma_X|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_X)^T \Sigma_X^{-1}(x-\mu_X)} \tag{3.33}$$

The normal random vector has many useful properties. For example, if $\Sigma_X$ is diagonal, then the components of $X$ are *independent* in additional to being *uncorrelated.* Also, if $A$ is a $k$ x $m$ matrix of rank $k$, then $Y = AX$ has a $k$-variate normal distribution with[49]

$$\mu_Y = A\mu_X \tag{3.34}$$

$$\Sigma_Y = A\Sigma_X A^T \tag{3.35}$$

## 3.7 Random Process

A random variable is a mapping from the sample space, $S$, to a real value. Similarly, a **random process** or **stochastic process** is a mapping from the sample space, $S$, to a waveform. To every outcome $\Lambda \in S$, we assign a waveform using

$$X(t, \Lambda) \tag{3.36}$$

Following the notations in [49], we omit $\Lambda$ and write a random process simply as $X(t)$.

The **mean** of $X(t)$ is the expected value of the random variable $X(t)$,

$$\mu_x(t) = E[X(t)] = \int_{-\infty}^{\infty} \Lambda X(t, \Lambda) d\Lambda \tag{3.37}$$

The **autocorrelation** of $X(t)$, denoted by $R_{XX}(t_1, t_2)$, is the expected value of the product $X^*(t_1)X(t_2)$,

$$R_{XX}(t_1, t_2) = E[X^*(t_1)X(t_2)] \tag{3.38}$$

where * denotes complex conjugate. The **autocovariance** of $X(t)$, denoted by $C_{XX}(t_1, t_2)$, is defined as

$$C_{XX}(t_1, t_2) = R_{XX}(t_1, t_2) - \mu_X^*(t_1)\mu_X(t_2) \tag{3.39}$$

### 3.7.1 Stationarity

According to definitions in [49], a random process is **strict sense stationary** if all of the distribution functions describing the process are invariant under a translation in time. A random process is **wide sense stationary** if the mean and autocorrelation function are invariant under a translation in time. Hence, the mean is constant and the autocorrelation function depends only on the time difference,

$$E[X(t)] = \mu_X \tag{3.40}$$

$$E[X^*(t)X(t+\tau)] = R_{XX}(\tau) \tag{3.41}$$

### 3.7.2 Power Spectral Density

The **power spectral density** or **spectrum**, $S(f)$, describes the power distribution of a random process in the frequency domain. For a wide sense stationary signal, it can be shown[49] that the power spectral density is the Fourier transform of the autocorrelation function.

$$S(f) = F\{R_{XX}(\tau)\} = \int_{-\infty}^{\infty} R_{XX}(\tau)e^{-j2\pi f\tau}d\tau \tag{3.42}$$

Given the power spectral density, the autocorrelation function is obtained as

$$R_{XX}(\tau) = F^{-1}\{S(f)\} = \int_{-\infty}^{\infty} S(f)e^{j2\pi f\tau}df \tag{3.43}$$

### 3.7.3 Time samples of random processes

Suppose $X(t)$ is a random process with power spectral density $S(f)$. For a specific $t$, $X(t)$ is a random variable. Therefore, the result of sampling a random process at $t$ is a random variable. Suppose we take $m$ samples of $X(t)$ at uniform interval $T$ starting from $T$. The result is a $m$-dimensional random vector, $Y$, with mean vector, $\mu_Y$, given by

$$\mu_Y = \begin{bmatrix} \mu_X(T) \\ \vdots \\ \mu_X(mT) \end{bmatrix} \tag{3.44}$$

and covariance matrix given by

$$\Sigma_Y(i,j) = C_{XX}(iT - jT) \tag{3.45}$$

# Chapter 4

# Behavioral Model for Data Converters

## 4.1 Background

Data converters convert the continuous-time, continuous-amplitude **analog** signals in the outside world to discrete-time, discrete-amplitude **digital** signals in electronic systems, and vice-versa. The behavior of a converter is affected by two basic statistical effects: (1) *noise*, (2) *process variations*. Noise can be of different sources, e.g, thermal noise, flicker noise, shot noise, and noise coupled from digital circuitry. Noise can cause the same chip to behave differently even if the same inputs are applied. Process variations cause different chips of the same circuit to have different behavior. To characterize these unintended, *second order effects*, traditional user specifications include static specifications such as integral nonlinearity (INL), differential nonlinearity (DNL), gain error, offset error, and probability of error due to noise, and dynamic specifications such as signal-to-noise ratio as a function of input frequency or input amplitude. The static specifications are for moderate speed converters that can be treated as **memoryless** (e.g. output does not depend on past inputs), while the dynamic specifications are important for high speed converters, where output can depend on past inputs. In this research, we focus on modeling a memoryless converter for static performance specifications.

Previously, Ruan[46] proposed different behavioral models for different types

of A/D converter architectures. One drawback is that the architectural dependence necessitates derivation of a new model for any changes in the converter architecture. Another drawback is that the model is deterministic. For example, a deterministic model can be expressed mathematically as

$$code = A(V_{in}, \hat{v}) \tag{4.1}$$

where *code* is the output code, $A$ is a function, $V_{in}$ is the input, and $\hat{v}$ is a set of $m$ parameters. There are two problems with this behavioral modeling approach; namely,

- Worst case analysis must be used to find worst case converter performance. For example, $2^m$ simulations are needed to find the worst case. For a typical converter, $m$ can be very large, making worst case analysis infeasible.
- Noise effects are not modeled, so signal-to-noise ratio cannot be calculated.

## 4.2 New Data Converter Behavioral Model

In contrast to the traditional deterministic model, we derive a stochastic converter model that include noise and process variation effects as follows. A memoryless Nyquist A/D operating at a certain environment such as fixed sampling frequency and temperature can be described by its transfer curve, which plots the output code on the range against the input continuous value on the abscissa. Assuming an A/D has $N$ bits of resolution, is designed to be monotonic and has no missing codes, the transfer curve can be characterized by $2^N - 1$ real numbers in a vector, $t$, representing the values of the transition points $t_i, i = 1 \ldots 2^N - 1$. Due to statistical process variations, the transfer curve $t$ has a statistical distribution. Due to noise, an A/D produces a wrong output with a non-zero probability. For example, if the input is near a transition point, the probability of getting a wrong output code, the code at the other side of the transition point, is non-zero. As a result, the noise effects depend on the input, and they become more significant as the input gets closer to a transition point. Mathematically, this is represented by a probability distribution function of the output codes for any input value. Our behavioral model captures this noise effect by a joint probability density function of the input $V_{in}$ and the output code for a given realization of the transfer curve $\hat{t}$,

$$f(code, V_{in}, \hat{t}) \tag{4.2}$$

From the above equation, which describes the behavior of any memoryless A/D due to noise and process variations completely, we can extract all information about the converter.

In our model, noise and process variation effects are separated. The distribution $t$ captures process variation effects, while the joint density function $f(\cdot)$ captures noise effects. The function $f(\cdot)$ is computed using either direct techniques[32] or Monte Carlo techniques. The process variation effects can be derived by making assumption:

**Assumption 4.2.1** *Process variations are the cumulative effects of many integrated circuit fabrication steps. Thus, we assume that the distribution of the process variations, $v$, is multivariate normal.*

Next, we propose a Gaussian model, as well as a non-Gaussian model for the distribution $t$. The Gaussian model is derived from making the following assumption:

**Assumption 4.2.2** *In general, an A/D is designed to be insensitive to process variations which are relatively small. Thus, the transition points, t, change relatively little with respect to process variations. Hence, we assume that t changes linearly with respect to the process variations:*

$$t = \frac{\partial t}{\partial v} v + \mu_t \tag{4.3}$$

*where $v$ is a random vector of parameters, corresponding to process variations such as offsets and mismatches in components, and $\frac{\partial t}{\partial v}$ is a sensitivity matrix.*

From assumption 4.2.1 and (4.3), the covariance matrix of $t$ is given by

$$\Sigma_t = \frac{\partial t}{\partial v} \Sigma_v \frac{\partial t}{\partial v}^T \tag{4.4}$$

where $\Sigma_v$ is the covariance matrix of $v$. Thus, the distribution of $t$ is

$$t \sim normal(\mu_t, \Sigma_t) \tag{4.5}$$

In general, the rank $r_t$ of $\Sigma_t$ is much less than full rank since not all the transition points are independent. In other words, any realization of $t$ must fall into a space spanned by $r_t$ independent basis vectors. A set of such independent basis vectors can be found by the singular value decomposition of $\Sigma_t$. Mathematically,

$$\Sigma_t = U_t \Sigma_{ct} U_t^T \tag{4.6}$$

where $U_t$ is an $2^N - 1$ x $r_t$ transformation matrix, $\Sigma_{ct}$ is an $r_t$ x $r_t$ diagonal matrix, and $r_t$ is the rank of $\Sigma_{ct}$. The columns of $U_t$ are called the **error signatures** [53] and form a set of orthonormal vectors spanning the **error space.** For storage and computational efficiency $t$ is expressed as the following equation instead of (4.3),

$$t = U_t c_t + \mu_t \tag{4.7}$$

where $c_t \sim normal(0, \Sigma_{ct})$ is a zero mean multivariate normal distribution with $r$ statistically *independent* variates. Typically, $r \ll m$, so (4.7) uses less computational resources than (4.5). The parameters $U_t$ and $\mu_t$ are unique for each A/D architecture, while the distribution $c$ represents a mixture of the relevant process variations.

Summarizing, our entire Gaussian A/D model consists of three equations,

$$f(code, V_{in}, \hat{t}), \hat{t} = realization(t) \tag{4.8}$$

$$t \sim normal(\mu_t, \Sigma_{ct}) = U_t c_t + \mu_t \tag{4.9}$$

$$c_t \sim normal(0, \Sigma_{ct}) \tag{4.10}$$

For cases where assumption (4.2.2) does not hold, we propose a non-Gaussian model. In such cases, the Gaussian model can be extended to a non-Gaussian model by appending a nonlinear *filter* to create the non-Gaussian distributions[5, 6]. Figure 4.1 shows a block diagram of a nonlinear model where the zero mean Gaussian error, $e = U_t c_t$, is being filtered to model a non-Gaussian error. For example, to model an

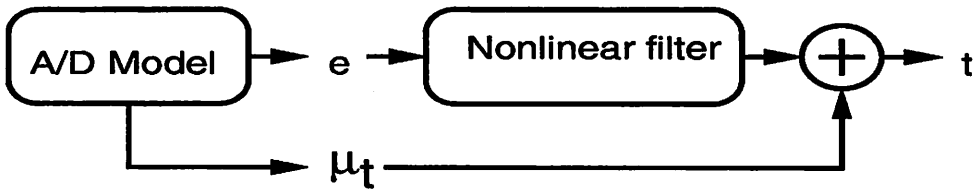


Figure 4.1: Nonlinear model for data converters

error with third order non-Gaussian statistics[57], a quadratic filter $b$ is used in the following equation for $t$ as a replacement for (4.9),

$$t = \mu_t + U_t c_t + \int_1^{2^N-1} \int_1^{2^N-1} b(t-u, t-v) e(u) e(v) du dv \tag{4.11}$$

where $e = U_t c_t$, $b$ is the two-dimensional impulse response of the quadratic filter, and $2^N - 1$ is the number of transition points. Although the non-Gaussian model

is more general, its use is limited because from our experience of modeling many converters most converters are adequately represented by a Gaussian model. Secondly, a Gaussian model is computationally more efficient than a non-Gaussian model.

After presenting the complete A/D model, we develop a similar D/A model using the same strategy. The resulting Gaussian D/A model is,

$$t = U_t c_t + \mu_t + z \tag{4.12}$$

$$c_t \sim normal(0, \Sigma_{ct}) \tag{4.13}$$

$$z \sim normal(0, \Sigma_z) \tag{4.14}$$

where $t$ is an $n$-dimensional vector and $z$ is an $n$-dimensional vector representing additive noise in D/A outputs. To extend the Gaussian D/A model to a non-Gaussian model, we replace (4.12) by the following,

$$t = \mu_t + U_t c_t + z + \int_1^{2^N} \int_1^{2^N} b(t-u, t-v) e(u) e(v) du dv \tag{4.15}$$

where $2^N$ is the number of D/A input codes.

## 4.3 Calculation of System Performance

Using the model, we propose a novel strategy to calculate system performance. The performance of a converter is computed in two steps. First, the converter model parameters are extracted from the circuit. Then, the converter performance is computed using only the model parameters since the model captures the converter behavior. The advantage of this strategy is that the model parameters are only extracted once, but used many times to calculate system performance. The following definitions illustrate how system performance parameters can be computed from the proposed D/A model parameters.

**Definition 4.3.1** *The ideal output vector, $L$, of a D/A converter is an $n$-dimensional vector, where the $i^{th}$ component of $L$, $L(i)$, represents the desired output for input code $i$, $n = 2^N$, and $N$ is the number of bits.*

**Definition 4.3.2** *The output vector, $t$, of a D/A converter is an $n$-dimensional vector, where the $i^{th}$ component of $t$, $t(i)$, is the output for input code $i$.*

The output vector, $t$, is identical to that in the model equation (4.12) with noise $z = 0$. Theoretically, $t$ is sufficient to describe the converter behavior due to process variations. Nevertheless, designers traditionally used a different set of specifications which can be derived easily from $t$. All of them are described as follows,

**Definition 4.3.3** *Offset error is* $t(1) - L(1)$.

**Definition 4.3.4** *Full scale gain error is* $t(n) - L(n) + L(1) - t(1)$.

The integral nonlinearity is the deviation of the output from the ideal value *after* compensation for the gain and offset errors.

**Definition 4.3.5** *The integral nonlinearity vector, $s$, of a $D/A$ converter is an $n$-dimensional vector*

$$s(i) = at(i) + b - L(i) \tag{4.16}$$

*where $a$, $b$ are constants such that $s(1) = s(n) = 0$.*

Intuitively, Definition 4.3.5 means that we transform $t$ using constants $a$ and $b$ such that the two end-points are ideal. In this case, we have adopted the more popular end-point method for linear error compensation[14], instead of the less popular least square method. The motivation behind the transformation is to compensate for the offset and gain errors before calculation of nonlinearity. In many applications, offset and gain errors are irrelevant as they do not contribute to distortion.

**Lemma 4.3.1** *Integral nonlinearity is given by*

$$s(i) = \left[\frac{L(n) - L(1)}{t(n) - t(1)}\right](t(i) - t(1)) + L(1) - L(i) \tag{4.17}$$

*The sensitivity matrix of the integral nonlinearity to output vector is given by*

$$\frac{\partial s}{\partial t}(i,j) = \begin{cases} \frac{(L(n)-L(1))(t(i)-t(n))}{(t(n)-t(1))^2} & i \neq 1,n \quad j = 1 \\ 0 & i \neq 1,n \quad j \neq 1,i,n \\ \frac{L(n)-L(1)}{t(n)-t(1)} & i \neq 1,n \quad j = i \\ 0 & i = 1,n \quad j = 1 \ldots n \\ \frac{(L(n)-L(1))(t(1)-t(i))}{(t(n)-t(1))^2} & i \neq 1,n \quad j = n \end{cases} \tag{4.18}$$

*where $\frac{\partial s}{\partial t}(i,j)$ is the entry at the $i^{th}$ row and $j^{th}$ column of the sensitivity matrix, $\frac{\partial s}{\partial t}$.*

**Proof.** Solving for $a, b$ using constraints in Def. 4.3.5 yield (4.17). Equation (4.18) follows from differentiation of (4.17). ■

Using a first order Taylor approximation of $s$ as a function of $t$,

$$s \approx \frac{\partial s}{\partial t}(t - \mu_t) + \mu_s = \frac{\partial s}{\partial t} U_t c_t + \mu_s \tag{4.19}$$

we find the integral nonlinearity has distribution,

$$s \sim normal(\mu_s, \Sigma_s) \tag{4.20}$$

where $\mu_s = s(t)$, at $t = \mu_t$ using (4.17) and $\Sigma_s = \frac{\partial s}{\partial t} \Sigma_t \frac{\partial s}{\partial t}^T$ at $s = \mu_s$. In general, the rank $r_s$ of $\Sigma_s$ is much less than full rank since not all the INL errors are linearly independent. In other words, any realization of $s$ must fall into a space spanned by $r_s$ independent basis vectors. A set of such independent basis vectors can be found by the singular value decomposition of $\Sigma_s$. Mathematically,

$$\Sigma_s = U_s \Sigma_{cs} U_s^T \tag{4.21}$$

where $U_s$ is a $2^N$ x $r_s$ transformation matrix, $\Sigma_{cs}$ is an $r_s$ x $r_s$ diagonal matrix, and $r_s$ is the rank of $\Sigma_{cs}$. The columns of $U_s$ are called the **INL signatures** and form a set of orthonormal vectors spanning the **INL space**. For storage and computational efficiency $s$ is expressed as the following equation,

$$s = U_s c_s + \mu_s \tag{4.22}$$

where $c_s \sim normal(0, \Sigma_{cs})$ is a small, zero mean multivariate normal distribution with $r_s$ statistically *independent* variates.

The motivation behind finding the distribution of $s$ is that INL is an important user constraint. By using the distribution of $s$, bounds on $s$ can be determined quickly and made to fit user constraints during design.

**Definition 4.3.6** *The $\pm k\sigma$ INL bounds are*

$$s_{\pm k\sigma}(i) = \mu_s(i) \pm k\sqrt{\Sigma_s(i,i)}. \tag{4.23}$$

These bounds are useful for design since any realization of $s$ will have a high probability of falling within the $k\sigma$ bounds if $k \geq 2$. Furthermore, once the design is done, circuit yield can be estimated by Monte Carlo integration over the distribution $s$.

In addition, the contribution of INL from each component can be precisely calculated. The INL contribution matrix, $p_s$, is defined as the cross-covariance between the INL vector $s$ and component variation vector $v$. An entry at $p_s(i,j)$ represents the contribution from component variation $v_j$ to the INL of code $i$, $s_i$. For example,

**Definition 4.3.7** $p_s = E[sv^T]$

**Lemma 4.3.2** $p_s = \frac{\partial s}{\partial t}\frac{\partial t}{\partial v}\Sigma_v$

**Proof.** Using (4.19), we have

$$p_s = E[sv^T] = E[(\frac{\partial s}{\partial t}(t-\mu_t)+\mu_s)v^T] \tag{4.24}$$

and using (4.3), we have

$$p_s = E[\frac{\partial s}{\partial t}\frac{\partial t}{\partial v}vv^T] + E[\mu_s v^T] \tag{4.25}$$

$$p_s = \frac{\partial s}{\partial t}\frac{\partial t}{\partial v}\Sigma_t \tag{4.26}$$

since $v$ has zero mean, and $E[vv^T] = \Sigma_v$. ■

Another useful piece of information to designers is the contribution to INL signatures from each component. Using this information, the importance of each component can be ranked, and the worst component can be pinpointed for redesign. The INL signature contribution matrix, $q_s$, is defined as the cross-covariance between the vector $c_s$ and component variation vector $v$. An entry at $q_s(i,j)$ represents the contribution from process variable $v_j$ to the magnitude of signature $i$, $c_i$. For example,

**Definition 4.3.8** $q_s = E[c_s v^T]$

**Lemma 4.3.3** $q_s = U_s^{-1}\frac{\partial s}{\partial t}\frac{\partial t}{\partial v}\Sigma_v$

**Proof.** Using (4.19), (4.22) and (4.3), we have

$$c_s = U_s^{-1}\frac{\partial s}{\partial t}(t-\mu_t) = U_s^{-1}\frac{\partial s}{\partial t}\frac{\partial t}{\partial v}v \tag{4.27}$$

$$q_s = E[c_s v^T] = E[U_s^{-1}\frac{\partial s}{\partial t}\frac{\partial t}{\partial v}vv^T] = U_s^{-1}\frac{\partial s}{\partial t}\frac{\partial t}{\partial v}\Sigma_v \tag{4.28}$$

using the definition. ■

Besides INL, designers are also interested in differential nonlinearity, DNL, defined as follows,

**Definition 4.3.9** *The differential nonlinearity vector, d, of a D/A converter is an $n-1$-dimensional vector obtained by a first order difference of the integral nonlinearity s.*

$$d = D_+ s \tag{4.29}$$

where $D_+$ is an $n-1$ by $n$ matrix called the first order difference operator,

$$D_+(i,j) = \begin{cases} -1 & i = j \\ 1 & i = j-1 \\ 0 & else \end{cases} \tag{4.30}$$

As DNL is a linear transformation of INL, most results related to INL carry over to DNL immediately. For instance, the differential nonlinearity has distribution,

$$d \sim normal(\mu_d, \Sigma_d) \tag{4.31}$$

where $\mu_d = D_+ s$, $\Sigma_d = D_+ \Sigma_s D_+^T$. In general, the rank $r_d$ of $\Sigma_d$ is much less than full rank since not all the DNL errors are independent. In other words, any realization of $d$ must fall into a space spanned by $r_d$ independent basis vectors. A set of such independent basis vectors can be found by the singular value decomposition of $\Sigma_d$. Mathematically,

$$\Sigma_d = U_d \Sigma_{cd} U_d^T \tag{4.32}$$

where $U_d$ is an $2^N - 1$ x $r_d$ transformation matrix, $\Sigma_{cd}$ is an $r_d$ x $r_d$ diagonal matrix, and $r_d$ is the rank of $\Sigma_{cd}$. The columns of $U_s$ are called the **DNL signatures** and form a set of orthonormal vectors spanning the **DNL space**. For storage and computational efficiency $d$ is expressed as the following equation,

$$d = U_d c_d + \mu_d \tag{4.33}$$

where $c_d \sim normal(0, \Sigma_{cd})$ is a small, zero mean multivariate normal distribution with $r_d$ statistically *independent* variates.

**Definition 4.3.10** *The $\pm k\sigma$ DNL bounds are:*

$$d_{\pm k\sigma}(i) = \mu_d(i) \pm k\sqrt{\Sigma_d(i,i)}. \tag{4.34}$$

These bounds are useful for design since any realization of $d$ will have a high probability of falling within the $k\sigma$ bounds if $k \geq 2$. In addition, the contribution of DNL from each component is defined similarly as the DNL contribution matrix, $p_d$.

**Definition 4.3.11** $p_d = E[dv^T]$

**Lemma 4.3.4** $p_d = D_+ \frac{\partial s}{\partial t} \frac{\partial t}{\partial v} \Sigma_v$

**Proof.**

$$p_d = E[dv^T] = E[D_+ sv^T] = D_+ E[sv^T] \tag{4.35}$$

from (4.29) and (4.26). ■

The DNL signature contribution matrix, $q_d$, is defined as

**Definition 4.3.12** $q_d = E[c_d v^T]$

**Lemma 4.3.5** $q_d = U_d^{-1} D_+ \frac{\partial s}{\partial t} \frac{\partial t}{\partial v} \Sigma_v$

**Proof.** Using (4.33), (4.29), (4.19) and (4.3), we have

$$c_d = U_d^{-1} D_+ \frac{\partial s}{\partial t}(t - \mu_t) = U_d^{-1} D_+ \frac{\partial s}{\partial t} \frac{\partial t}{\partial v} v \tag{4.36}$$

$$q_d = E[c_d v^T] = E[U_d^{-1} D_+ \frac{\partial s}{\partial t} \frac{\partial t}{\partial v} v v^T] = U_d^{-1} D_+ \frac{\partial s}{\partial t} \frac{\partial t}{\partial v} \Sigma_v \tag{4.37}$$

using the definition. ■

Traditionally, more complex converter specifications such as yield, harmonic distortion and signal-to-noise ratio are obtained from chip data since simulations are computationally infeasible for these system specifications. However, using the proposed model these specifications can be estimated efficiently. For example, yield is typically defined as the fraction of circuits that have INL and DNL within some bounds, $s_{max}$ and $d_{max}$, respectively. Therefore, yield is obtained by Monte Carlo integration over a region in distributions, $s$ and $d$. Using (4.19) and (4.29), the regions are

$$|\frac{\partial s}{\partial t} U_t c_t + \mu_s| < s_{max} \tag{4.38}$$

$$|D_+(\frac{\partial s}{\partial t} U_t c_t + \mu_s)| < d_{max} \tag{4.39}$$

In the Monte Carlo simulations, many realizations of $c_t$ are generated from a random number generator, some of which are labeled success if they fall inside both regions. Yield is estimated from the fraction of success over a finite number of realizations.

Two other useful system performance parameters are harmonic distortion and signal-to-noise and distortion ratio, S/(N+D). These parameters are defined in the

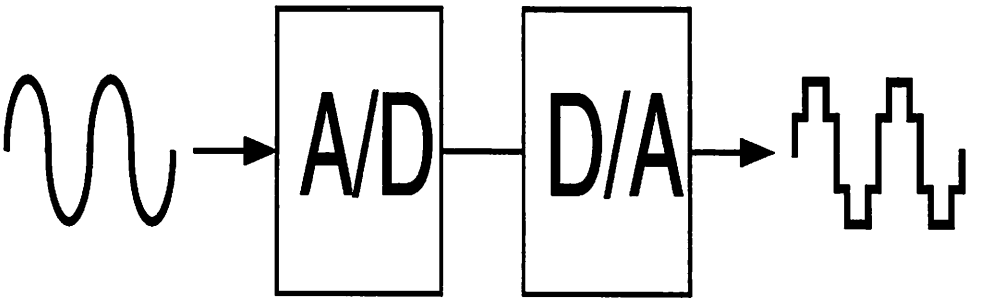


Figure 4.2: System setup for distortion and noise measurements

context of a simple A/D-D/A test system shown in Figure 4.2 in which one component is ideal and the other non-ideal component is under evaluation. Harmonic distortion describes the harmonic content of the near sinusoidal output generated from a sinusoidal input. Traditional time domain simulations and spectral estimation techniques employed for this calculation are inefficient because a long time domain simulation is required for a confident estimate of the spectrum. As a result, we propose a direct approach. First, the composite static system transfer function, $T(\cdot)$, relating the instantaneous input and output is obtained from the static transfer functions of the D/A and A/D. Assuming $x(i)$ corresponds to $2^N - 1$ A/D transition points and $y(i)$ corresponds to $2^N$ D/A output values, the transfer function $T(\cdot)$ has the shape shown in Figure 4.3. Then, the transfer function is expressed as a Taylor expansion of the input,

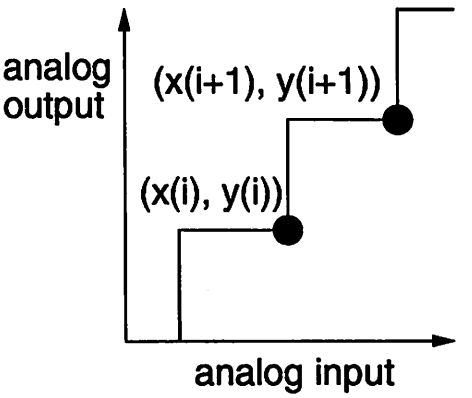


Figure 4.3: Composite transfer function of A/D-D/A system

$$T(x) = \sum_{j=1}^{N_a} a(j)x^{j-1} + \epsilon(x) \tag{4.40}$$

where $a$ is an $N_a$ dimensional vector of power series coefficients and $\epsilon(\cdot)$ is the residual corresponding to higher order terms. A least square estimate of $a$ is obtained by linear regression over the points

$$(x(i), \frac{y(i)+y(i+1)}{2}), i = 1 \ldots 2^N - 1 \tag{4.41}$$

From linear regression theory[5], the optimal least square fit corresponds to

$$a = (X^T X)^{-1} X^T D_1 y \tag{4.42}$$

where $X(i,j) = x(i)^{j-1}$ is an $2^N - 1$ by $N_a$ matrix and $D_1$ is an $2^N - 1$ by $2^N$ transformation matrix given by

$$D_1(i,j) = \begin{cases} 0.5, & i = j \\ 0.5, & j = i+1 \\ 0, & else \end{cases} \tag{4.43}$$

The amplitudes, $V$, of the first seven output sinusoids, including DC, are

$$V(0) = \frac{a(2)}{2} + \frac{3a(4)}{8} + \frac{37a(6)}{32} \tag{4.44}$$

$$V(1) = a(1) + \frac{3a(3)}{4} + \frac{5a(5)}{8} \tag{4.45}$$

$$V(2) = \frac{a(2)}{2} + \frac{a(4)}{2} + \frac{21a(6)}{16} \tag{4.46}$$

$$V(3) = \frac{a(3)}{4} + \frac{5a(5)}{16} \tag{4.47}$$

$$V(4) = \frac{a(4)}{8} + \frac{3a(6)}{16} \tag{4.48}$$

$$V(5) = \frac{a(5)}{16} \tag{4.49}$$

$$V(6) = \frac{a(6)}{32} \tag{4.50}$$

According to the definition of harmonic distortion, the $i^{th}$ harmonic distortion parameter is

$$HD_i = \frac{V(i)}{V(1)}, for\ i > 1 \tag{4.51}$$

Ignoring higher order harmonics, the total harmonic distortion[14] in decibels is estimated to be

$$THD = 20log\left[\frac{\sqrt{V(2)^2 + V(3)^2 + V(4)^2 + V(5)^2 + V(6)^2}}{V(1)}\right] \tag{4.52}$$

The mean and standard deviation of all harmonic distortion parameters are computed using a small number of Monte Carlo simulations from which the $3\sigma$ worst case values are estimated.

The final performance specification is signal-to-noise and distortion ratio, S/(N+D). The full scale sinusoidal input, $u$, has probability distribution function,

$$f_U(u) = \begin{cases} \frac{1}{\pi\sqrt{(\frac{V_{ref}}{2})^2-(u-\frac{V_{ref}}{2})^2}}, & 0 < u < V_{ref} \\ 0, & else \end{cases} \tag{4.53}$$

The output noise and distortion error energy is given by the D/A output plus noise minus the linear components

$$(DA(c) + z(c) - a(0) - a(1)u)^2 \tag{4.54}$$

where $DA(\cdot)$ is the D/A transfer function, $c$ is the output code from A/D, and $z(\cdot)$ is the D/A noise. When A/D noise is considered also, then we have the sum over all possible A/D output codes

$$\sum_{c=1}^{2^N}(DA(c) + z(c) - a(0) - a(1)u)^2 f_c(u, c, \hat{t}) \tag{4.55}$$

where $f_c$ is the joint probability function given by (4.8). The total error is the integral of the above over the entire input range,

$$\int_0^{V_{ref}} \sum_c (DA(c) + z(c) - a(0) - a(1)u)^2 f_c(u, c, \hat{t}) f_U(u) du \tag{4.56}$$

Although the above expression describes completely the errors from noise and distortion, the special case of noiseless converters is also useful because it provides a performance upper bound. The total error energy without noise can be expressed as

$$\int_0^{V_{ref}} (T(u) - a(0) - a(1)u)^2 f_U(u) du \tag{4.57}$$

The signal energy is given by

$$\int_0^{V_{ref}} (a(1)u)^2 f_U(u) du \tag{4.58}$$

Therefore, S/(N+D) is the ratio of the quantities defined above in decibels. A standard measure of effective converter resolution is the effective number of bits[14],

$$\frac{S/N + D - 1.76}{6.02} \tag{4.59}$$

The mean and standard deviation of the above parameters are estimated using a small number of Monte Carlo simulations from which the $3\sigma$ worst case values are estimated.

## 4.4 Model Validation

A model must be verified against measurements to test its validity. Our model,

$$f(code, V_{in}, \hat{t}), \hat{t} = realization(t) \tag{4.60}$$

describes the behavior of any memoryless A/D due to noise and process variations completely. However, in our more specific Gaussian model, the Gaussian assumption of $t$ must be verified. Therefore, we need to verify process variation effects in (4.9) by verifying $t$ has the multivariate normal distribution for a variety of converter types. The verifications are done using Monte Carlo simulations with SPICE, as well as real chip data. Assuming typical process variations such as standard deviation of transistor length and width being $0.05\mu m$ and standard deviation of resistor value being one percent, Monte Carlo simulations of a 10 bit current-switched D/A converter consisting of 1068 transistors and 345 parasitic resistors have been performed for randomly selected input codes. The random codes are obtained using a uniformly distributed random number generator on an Hewlett Packard 42S calculator.

The output current distributions for all such simulations match well with the normal distribution, as verified by the quantile-quantile plots in Figure 4.4 and Figure 4.5 for randomly selected input codes 100 and 900, respectively. Each point on the graph represents a comparison between the cumulative distribution function of the output current and that of the normal distribution. Perfect matches fall in a straight line, validating components of the output current vector, $t$, have normal distributions.

Another implication of the A/D model is that the errors of a converter can be decomposed into its errors signatures. To verify that errors signatures are sufficient to characterize an actual A/D, we obtained the measured error, $\hat{t} - \mu_t$, of a fabricated cyclic A/D[38], and try to fit the measured data, $\hat{t}$, with a linear combination of error signatures using linear regression techniques

$$\hat{t} = \mu_t + U_t\hat{c} + \delta \tag{4.61}$$

where we solve for $\hat{c}$, an unknown realization of $c$ that minimize the measurement noise $||\delta||^2$, where $||\cdot||^2$ denotes the inner product of a vector with itself. Figure 4.6 shows

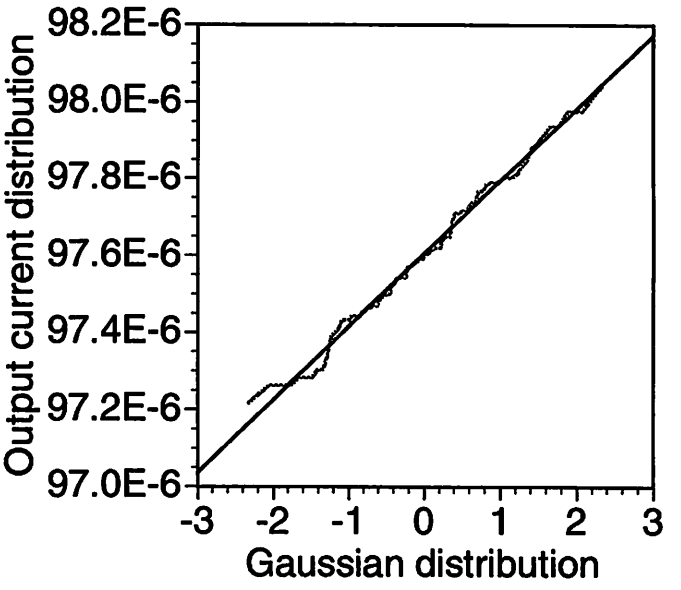


Figure 4.4: Quantile-quantile plot of output distribution vs. normal distribution for code=100

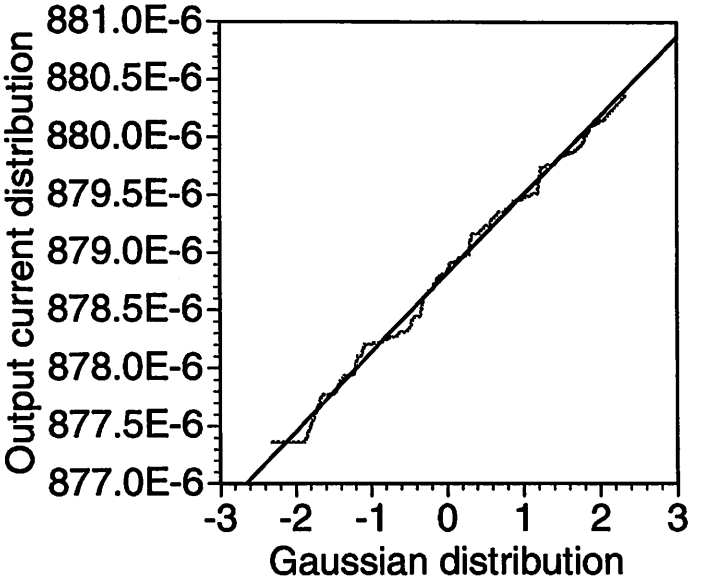


Figure 4.5: Quantile-quantile plot of output distribution vs. normal distribution for code=900

the curve $\hat{t} - \mu_t$ matches well with the curve $U\hat{c}$, proving that the model sufficiently captures the A/D behavior.

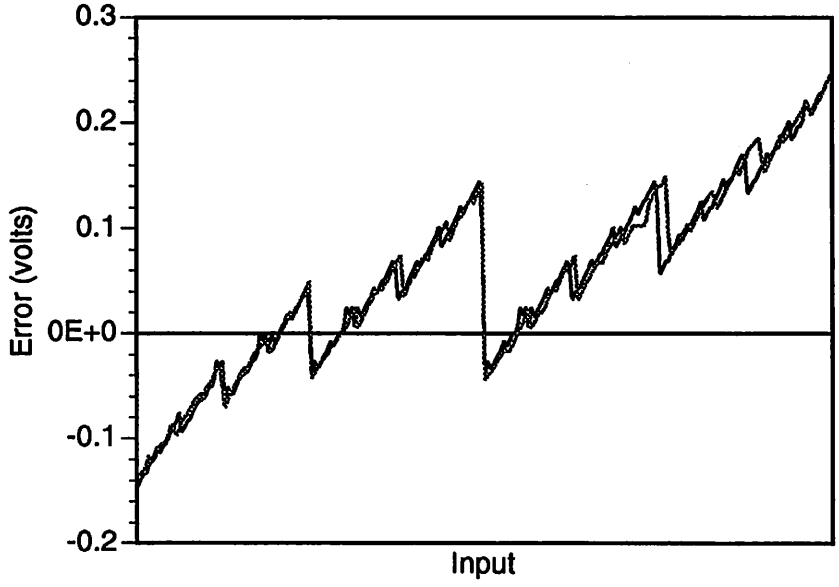


Figure 4.6: Comparison of measured and model generated errors

## 4.5 Conclusion

We have presented a behavioral representation of Nyquist data converters. The representation captures the behavior of a memoryless Nyquist data converter, including statistical variations. The variations are classified into noise and process variations according to how these non-idealities affect the converter behavior. To describe noise effects, a joint probability density function is used. To describe process variations effects on the converter transfer function, a Gaussian model is used.

Using the behavioral representation, a new strategy to estimate system performance is developed. The performance specifications of a converter, including offset error, full scale gain error, INL, DNL, harmonic distortion and signal-to-noise ratio, are estimated in two steps. First, the converter model parameters are extracted from the circuit. Then, the converter performance is computed using only the model parameters since the model captures the converter behavior. Given that the behavioral model is valid, offset error and gain error are computed exactly, while INL and DNL are

approximated using first order Taylor approximation. Worst case harmonic distortion and signal-to-noise ratio are approximated using Monte Carlo simulations.

From SPICE simulations, the distribution of D/A errors for a 10-bit converter were shown to agree well with the Gaussian distribution; thus, validating our Gaussian error assumption. Furthermore, behavioral simulation results agree well with measurements for an 8-bit cyclic A/D converter[38]; thus, validating the behavioral model.

# Chapter 5

# Analog System Verification using Behavioral Simulation

## 5.1 Background

We have proposed a constraint-driven, top-down design methodology for mixed-mode systems[10] that is supported by analog design tools. An integral part of such methodology is the complete verification of the synthesized circuit in the presence of layout parasitic resistances and capacitances due to routing, supply variations, and coupling. Because parasitics degrade analog system performance, it is crucial to verify by simulation as completely as possible the circuit functionality in the presence of these second order effects.

Existing design methodologies, manual or automatic [26, 7, 1], use SPICE simulations to verify the final designs at the transistor level. Unfortunately, SPICE simulations often fail for larger systems due to non-convergence or unreasonable computational cost. For example, to simulate process variations of a circuit, designers traditionally use worst case analysis. In this approach, designers simulate all, $2^M$, process corners where $M$ is the number of process variables. In a large system such as a 10 bit D/A with 600 matched transistors, the number of process corners reaches an astronomical $2^{600}$, rendering exact worst case analysis impossible. Consequently, designers simulate a subset of the process corners, but no general algorithm has been proposed for choosing the subset.

Due to the difficulties, several behavioral simulation approaches have been proposed earlier[36, 46]. Unfortunately, they do not take into account loading parasitics and must also use worst case analysis to compute process variation effects.

In some design methodologies such as standard cell analog designs, designers verify each cell extensively by SPICE simulation. Yet, the final system of interconnected cells may fail due to interconnect parasitics or cell interactions.

## 5.2 Verification Tasks in Top-Down Design

In contrast, our top-down design methodology[10] decomposes a system into its components, then the components into subcomponents using the same methodology until constraint-driven physical layout synthesis. In such a design environment, the key verification tasks are the verification of component functionality and the verification of the system *after* the components are routed in a system. We use the former type of verification right after component synthesis to check whether or not an individual component works *without loading effects*. If not, we redesign the component immediately, thus avoiding expensive design iterations. This type of verification is easier because the isolated component can often be verified with SPICE. After we verify each component against its specifications, we check the system performance in the presence of layout parasitics. This last step has been carried out in the past at the SPICE simulation level. In this chapter, we present two methodologies that use behavioral models of the components and behavioral simulation to verify analog systems. The first methodology verifies analog systems in the absence of parasitics (useful for system level block diagram verification in which parasitics are not considered), while the second methodology verifies analog systems in the presence of parasitics (useful for final verification from layout). The benefits of the new approach are overwhelming when compared to SPICE level simulation.

## 5.3 Verification in the Absence of Parasitics

### 5.3.1 General Strategy

Behavioral simulation is effective in verifying analog systems without parasitic loading effects because without parasitic effects, each component can be modeled and characterized *independently*. For example, we follow the following steps in verification:

- Development of behavioral models of components.
- Component layout extraction.
- Component identification using SPICE simulation of the components at ideal bias conditions.
- Parameter fitting for behavioral models.
- System simulation at the *behavioral* level using behavioral models only.

### 5.3.2 Verification of D/A Converters

We illustrate the behavioral modeling and verification results of a five-bit, current-switched, interpolative D/A converter, compared with traditional SPICE simulations. We assume the presence of process variations such as standard deviation of transistor length and width being $0.05\mu m$. Using simulation, we seek the complete static performance such as offset error, full scale gain error, INL, and DNL. To achieve our goal, we use the Nyquist converter model proposed in Chapter 4, in conjunction with the procedure to calculate the static performance presented in Section 4.3.

Illustrated in Figure 5.1, the D/A architecture contains a first stage, a second stage, and a mirror connecting the first to the second stages. $I_{ref}$ supplies the reference current, $I_{out}$ is the output, and the five input bits drive the switches through some combinational logic block. The components are all current mirrors described by their statistics, such as the nominal mismatch $\mu_v$ and the covariance matrix $\Sigma_v$, where $\mu_v$ is

$$\mu_v(i) = k_i, i = 1 \ldots 8, \tag{5.1}$$

and each $k_i$ is the mismatch, e.g. the output to input current ratio, of a component. The covariance matrix is given by $\Sigma_v(i,j) = Cov(k_i, k_j)$. For this example, we ignore

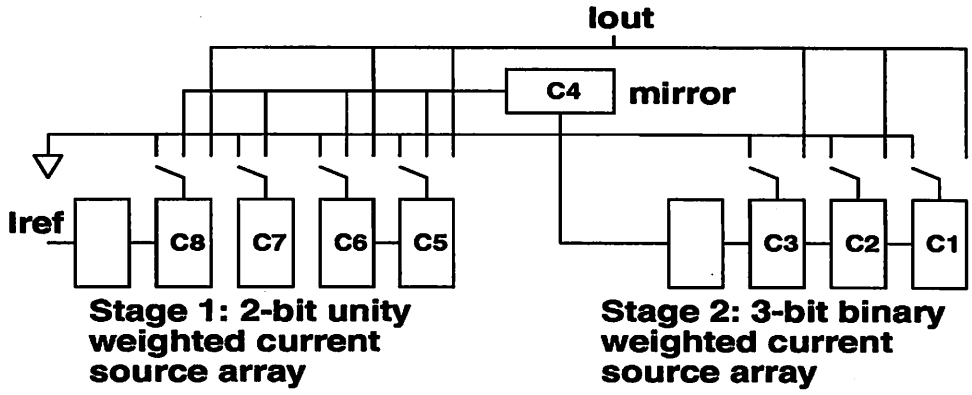


Figure 5.1: 5-bit interpolative D/A architecture

correlations between components in our architecture, so the covariance matrix is diagonal. However, correlations can be handled by using a full matrix if so desired by the user.

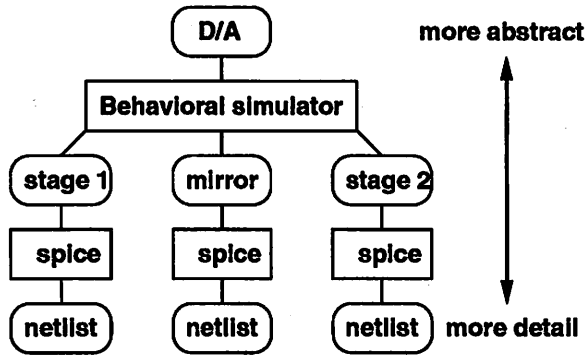


Figure 5.2: Model Hierarchy for 5-bit D/A

Using the model hierarchy shown in Figure 5.2, we compute the converter performance in two steps. First, component parameters $k_i$ are extracted using SPICE simulations on each component separately. *It should be emphasized that the component parameters are extracted independently, thus avoiding costly simulation of an entire chip.* In this experiment, the ideal bias point is chosen to be very close to the actual bias point. In cases where a bias point is not known in advance, we use a procedure based on sensitivity analysis[29] (Section 5.4). Variances of $k_i$ are extracted using either sensitivity analysis (exact) or Monte Carlo SPICE simulations (approximate) of each component. Once the component parameters are extracted, higher level Nyquist

converter parameters are expressed as,

$$\mu_t = f(\mu_v) \tag{5.2}$$

$$\Sigma_t = \left[\frac{\partial t}{\partial v}\right] \Sigma_v \left[\frac{\partial t}{\partial v}\right]^T \tag{5.3}$$

where $f(\cdot)$ is computed using a behavioral simulator and $\frac{\partial t}{\partial v}$ is estimated using finite differences. The behavioral simulator is written in the C++ language (Section 5.3.3). Once the converter model parameters $\mu_t$ and $\Sigma_t$ are computed, all static specifications are computed using the formulae given in Section 4.3.

### 5.3.3 Software Implementation

The basic behavioral description of a converter is the function, $g$, relating the output, $t_i$, to the input code, $i$, and component parameters, $v$.

$$t_i = g(i, v) \tag{5.4}$$

This function can be implemented using either a custom input language for a custom simulator or standard computer languages and standard compilers. In this example, the function coded in the C++ language is shown below, where *code* is the input code and the component parameters are *k1, k2, .....*

```
double g(int code)
{
  double iref = 31.25e-6;
  double i=0.0, inext;
  switch(code>>3) {
    case 0: inext = k5 * iref; break;
    case 1: i += k5 * iref; inext = k6 * iref; break;
    case 2: i += (k5+k6) * iref; inext = k7 * iref; break;
    case 3: i += (k5+k6+k7) * iref; inext = k8 * iref; break;
  }
  inext *= k4;
  if (code & 1) i += k1 * inext;
  if (code & 2) i += k2 * inext;
```

```
  if (code & 4) i += k3 * inext;
  return i;
}
```

The function, $f$, returns the output current vector, $t$, in an array after evaluating (5.4) for all input codes,

$$t = f(v) \tag{5.5}$$

Using (5.5) with $v = \mu_v$, we compute (5.2). Next, we compute $\frac{\partial t}{\partial v}$ using finite differences on (5.5). Finally, we evaluate (5.3) and performance parameters using the formulae given in Section 4.3. All of the above operations, computing output current vectors and taking finite differences, are implemented in a prototype simulator written in C++. The behavioral description of the converter is linked-in during compile time.

### 5.3.4 Experimental Results

From theory, it was predicted that most fabricated chips will have INL curve falling within the predicted $\pm 2\sigma$ INL bounds. The simulated $\pm 2\sigma$ INL bounds are shown in Figure 5.3 along with 100 INL curves from Monte Carlo SPICE simulations. Almost all INL curves from SPICE simulations fall within the predicted bounds, thus demonstrating that the behavioral model is plausible. Similarly, all DNL curves from SPICE simulation fall within the predicted bounds as shown in Figure 5.4.

The INL errors are also predicted to be in an error space spanned by independent INL signatures. Shown in Figure 5.5 are the four most important INL signatures, along with a weight indicating their relative importance. The most important error (first error in Figure 5.5) has a bow shape, which confirms with designer experience that converters of the type exhibit INL errors resembling a bow. However, designers neglected the other errors also found using our mathematical approach.

In order to give feedback to designers, it is useful to find the contribution of the bow shape INL error from each component; thereby, pinpointing the worst component for redesign. Figure 5.6 shows the INL signature contribution matrix which shows that the bow shape error is contributed mostly by the first stage components $C_5$ to $C_8$. As confirmed by design experience, INL is contributed mainly by the unity weighted current sources in the first stage of this type of D/A. Notice that the contributions from components $C_5$ and $C_8$ are equal and opposite, indicating that they should be matched.

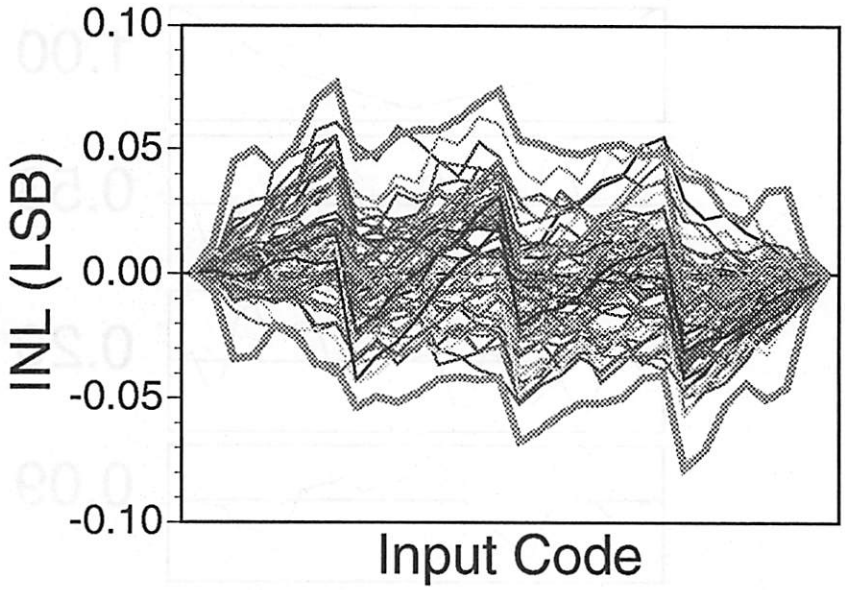


Figure 5.3: 100 Monte Carlo INL simulations within predicted INL bounds

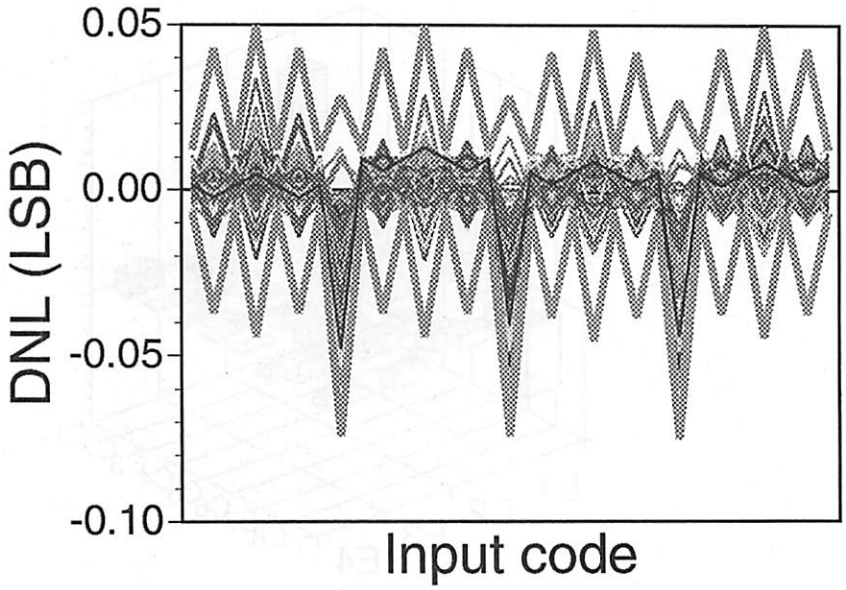


Figure 5.4: 100 Monte Carlo DNL simulations within predicted DNL bounds

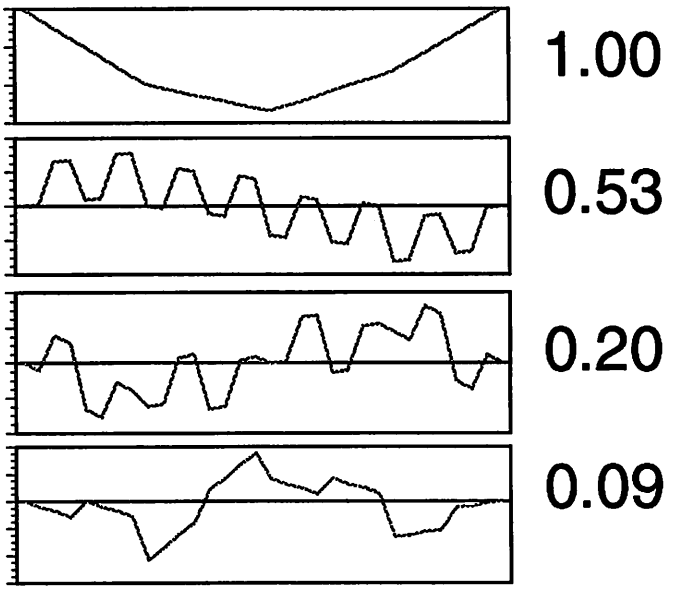


Figure 5.5: Four most important INL signatures with relative weights

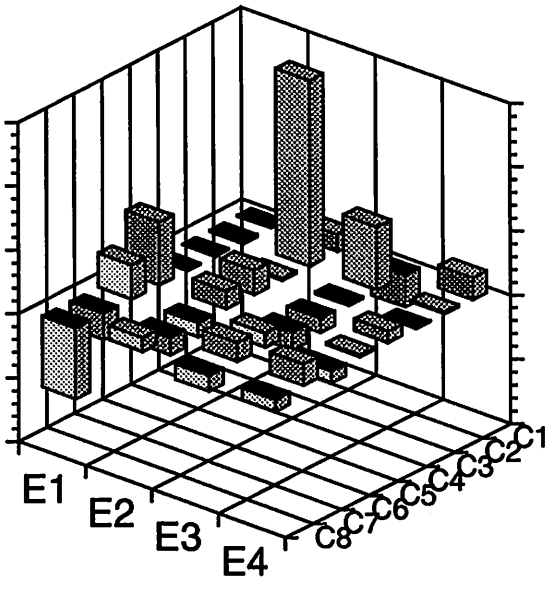


Figure 5.6: INL signature contribution matrix

| Name | -3$\sigma$ | Typical | +3$\sigma$ | Units |
|---|---|---|---|---|
| offset | 0.0 | 0.0 | 0.0 | LSB |
| gain error | -0.166 | 0.015 | 0.196 | LSB |
| INL | | 0.020 | 0.107 | LSB |
| DNL | | 0.024 | 0.101 | LSB |

Table 5.1: Performance summary

If calibration or dynamic matching techniques are used, then they should be calibrated or matched to have the same value. The simulated performance summary is given in Table 5.1.

Finally, the most time consuming step in behavioral simulation is the extraction of component parameters using SPICE. Together with the high level behavioral simulation, the total DEC 5000/125 CPU time needed for behavioral simulation is 534.5 seconds in this example. In contrast, worst case analysis of the entire circuit using SPICE will be infeasible because there are 158 transistors in the circuit with 316 process variables. Worst case analysis necessitates $2^{316}$ SPICE simulations of the entire circuit each using 134 CPU seconds. To circumvent worst case analysis, we validated our behavioral simulation results instead with Monte Carlo simulations. The 100 Monte Carlo simulations of the entire chip used 3.7 CPU hours. Unfortunately, Monte Carlo simulations will become infeasible as the chip size increases. For example, 100 Monte Carlo simulations of a 10 bit D/A converter would take 6.8 CPU days, while behavioral simulation would take about the same amount of time as a 5 bit converter because the larger converter is composed of more of the same components whose characteristics are extracted once and for all.

## 5.4 Verification in the Presence of Parasitics

### 5.4.1 General Strategy

The problem with verification in the absence of parasitics in Section 5.3 is that the components are characterized at ideal bias conditions. Because of parasitics, e.g. parasitic resistance, the components bias will shift causing system performance degradation. To remedy the problem, we propose the following approach.

In general, suppose the system performance, $S$, is a function of the behavioral model parameters, $W$, which in turn are functions of the bias conditions, $V$. Due to parasitics, $p$, $V$ becomes

$$V = V_0 + \frac{\partial V}{\partial p}p + \frac{\partial^2 V}{\partial p^2}p^2 + \ldots \approx V_0 + \frac{\partial V}{\partial p}p \tag{5.6}$$

where $V_0$ is the ideal bias. The first order Taylor approximation is justified by the fact that most components have linear impedance characteristics for the typical range of parasitics. Therefore,

$$W(V) \approx W(V_0) + \frac{\partial W}{\partial V}\frac{\partial V}{\partial p}p \tag{5.7}$$

$$S(W(V)) \approx S(W(V_0)) + \frac{\partial S}{\partial W}\frac{\partial W}{\partial V}\frac{\partial V}{\partial p}p \tag{5.8}$$

using first order Taylor approximation. For (5.7) and (5.8) to be valid, the partial derivatives $\frac{\partial S}{\partial W}$ and $\frac{\partial W}{\partial V}$ must exist and errors small.

To use (5.8) to compute effects due to parasitic loading, we change the verification methodology to the following:

- Layout extraction,
- Component identification using SPICE simulation of the components at ideal bias conditions $V_0$,
- Parameter fitting for behavioral models.
- Component linearization at ideal bias conditions,
- Linearized component insertion into interconnection network,
- Network solving for new bias conditions $V$,
- Optional step: Sensitivity computation of $V$ with respect to all elements in the interconnect using adjoint techniques[12].
- Bias change (from ideal bias conditions) computation, $V - V_o$, on linearized components.
- Changes (sensitivity) of $V$ propagation to changes (sensitivity) of behavioral model parameters $W$, and then to changes (sensitivity) of system performance $S$.

The advantage of the new approach is that we can compute system level performance sensitivity due to routing parasitics. In our performance driven layout synthesis approach, the layout synthesis tools use the sensitivity information intelligently to assemble a global layout from a collection of component layouts. Then, a behavioral simulator computes the system performance degradation due to the extracted routing parasitics.

### 5.4.2 Verification of D/A Converters

In this section we apply the new verification approach to a synthesized current-switched, interpolative 10 bit D/A (architecture in Figure 5.7 and layout in Figure 5.8) that consists of a 5 bit linear current source array, a 5 bit binary current source array, and a mirror. The 10 bit converter is synthesized using our proposed constraint-driven, top-down design methodology[10] for maximum INL less than 2.0 LSB. Due to high converter resolution, parasitic effects are no longer negligible.

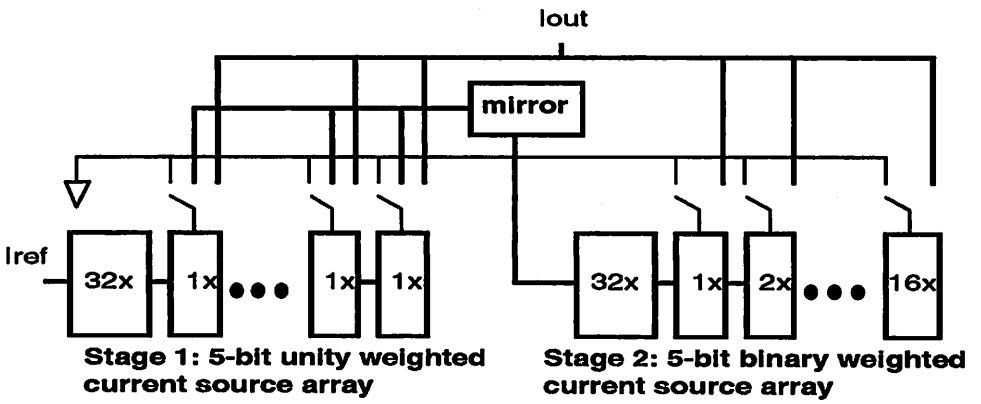


Figure 5.7: 10 bit interpolative D/A architecture

The key non-idealities in the design are mismatches in the transistors, parasitic resistances, parasitic capacitances, and output resistances in the transistors. Capacitances cause output glitches during fast switching. Because the converter is assumed to operate at a moderate speed, parasitic capacitances are less significant and ignored in this design. Parasitic resistances, transistor mismatches, and output resistances degrade system performance such as integral nonlinearity. We compute the system performance in three steps. First, we extract SPICE netlists for each component from the layout, then behavioral models parameters from the netlists, and finally system per-

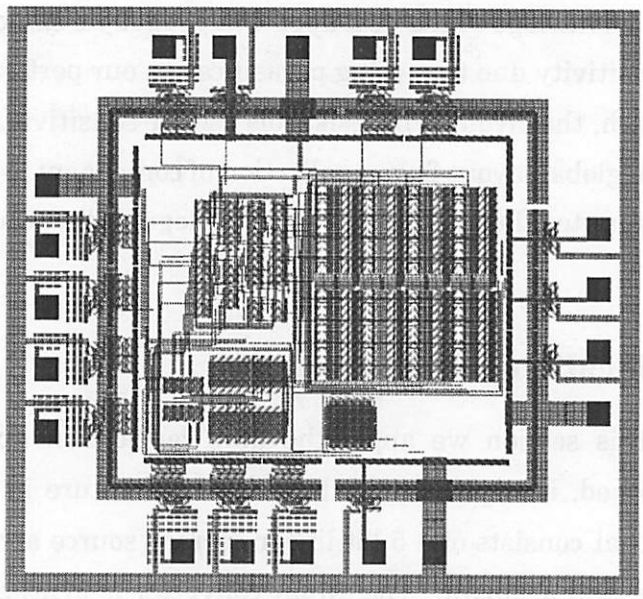

Figure 5.8: 10 bit interpolative D/A layout

formance from behavioral models. Because we only consider effects due to parasitic resistance and resistive loading in our D/A, we only implemented software to compute system degradation due to these effects.

### 5.4.3 From Layout to SPICE Netlist

An extractor capable of extracting resistances is used to convert the layout to schematic. The process variation random vector, $v$, is assumed to be Gaussian distributed,

$$v \sim normal(0, \Sigma_v) \tag{5.9}$$

where $\Sigma_v$ is obtained by combining information about the process and the *geometry of the layout*. Suppose transistor $i$ has parameter, $v(i)$, that depends on its location on the wafer,

$$\begin{aligned} v(i) &= f(x_i, y_i, r_1, \ldots, r_n) \\ &\approx \frac{\partial f}{\partial r_1}(x_i, y_i, r_1, \ldots, r_n) r_1 + \ldots + \frac{\partial f}{\partial r_n}(x_i, y_i, r_1, \ldots, r_n) r_n \end{aligned} \tag{5.10}$$

where $x_i$ is the x-coordinate, $y_i$ is the y-coordinate of the transistor, $r_1 \ldots r_n$ are zero mean random coefficients, and the approximation is justified by a Taylor approximation. The random coefficients correspond to random process variations such as random

slope and orientation of process gradients. The joint distribution of the parameter variation of $k$ different transistors has covariance matrix

$$\Sigma_v = \frac{\partial v}{\partial r}\Sigma_r \frac{\partial v}{\partial r}^T \tag{5.11}$$

$$\frac{\partial v}{\partial r}(i,j) = \frac{\partial v(i)}{\partial r_j} \tag{5.12}$$

where $\Sigma_r$ is the covariance of the coefficients.

We applied the above theory to the analysis of process gradient effects. Random coefficients, $r_1$ and $r_2$, corresponds to random slopes on the x-axis and the y-axis, respectively, such that

$$v(i) = f(x_i, y_i, r_1, r_2) = r_1 x_i + r_2 y_i \tag{5.13}$$

where $(x_i, y_i)$ is the location of transistor $i$. Suppose the covariance of $[r_1\ r_2]^T$ is

$$\Sigma_r = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_r^2 \end{bmatrix} \tag{5.14}$$

we compute the covariance of the parameters of two transistor located at $(x_1, y_1)$ and $(x_2, y_2)$ from

$$\Sigma_v = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix} \Sigma_r \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}^T \tag{5.15}$$

Next, we express the mismatch of the two parameters as $v(1) - v(2)$, a linear transformation of $v$,

$$v(1) - v(2) = \begin{bmatrix} 1 \\ -1 \end{bmatrix} v$$

Consequently, the variance of mismatch is given by $\sigma^2 = \sigma_r^2((x_1 - x_2)^2 + (y_1 - y_2)^2))$, implying that the mismatch $\sigma$ is proportional to the Euclidean distance between two components. Such prediction is consistent with data in [41][42]. Although theories for mismatches in MOS transistors were developed in [42] based on **spatial frequency**, we propose here a new theory that can estimate covariance of transistor parameters in large transistor arrays efficiently by using sensitivity analysis and matrix algebra.

### 5.4.4 From SPICE Netlist to Behavioral Models

We extract behavioral model parameters from netlist for each component using SPICE. We compute parameters by applying *ideal* voltage and current sources, $V_s$, to the pins of the components. For instance, the mirror model shown in Figure 5.9 has four parameters; namely, the current mismatch, $K_M = \frac{I_{out}}{I_{in}}$, the output resistance of the mirror, $R_M$, the standard deviation of $K_M$, $S_M$, and the input voltage of the mirror, $V_{in}$. We compute current mismatch $K_M$ and input voltage $V_{in}$ by a DC analysis,

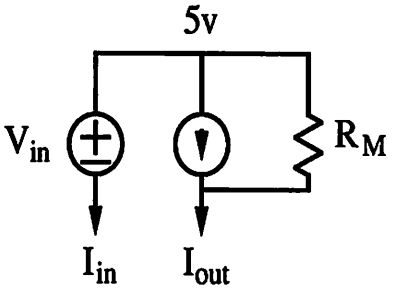


Figure 5.9: Mirror behavioral model

the output resistance $R_M$ by an AC analysis, and the standard deviation $S_M$ by

$$S_M = \left[\frac{\partial K_M}{\partial v}\right] \Sigma_v \left[\frac{\partial K_M}{\partial v}\right]^T \tag{5.16}$$

where $v$ is a random vector of process variations (5.9) such as the width and length variations of all the devices. For simplicity, we compute $\frac{\partial K_M}{\partial v}$ using finite differences, and $\Sigma_v$ from layout and process variation information using (5.15).

To characterize the effects due to interconnect and loading parasitics, we first compute the sensitivity of the bias due to changes in ideal sources $V_s$ using AC analysis in SPICE. We obtain a *linearized* mirror model for later insertion into the linear interconnect network.

We characterize other components in the D/A the same way. But, due to the mixed-mode nature of some components in the D/A, the parameters for some linearized components change as a function of the input bits. For example, the output impedance of the 5 bit linear array decreases with the input code because more current sources turning on in parallel decreases output resistance. To characterize completely the linear array, we take 32 sets of linearized parameters corresponding to every

combination of the 5 bit input code. The same characterization is done for the binary array.

### 5.4.5 From Behavioral Models to INL

In Chapter 4, we approximate the distribution of $t$ as a multivariate Gaussian distribution,

$$t \sim normal(\mu_t, \Sigma_t) \tag{5.17}$$

We compute the nominal values, $\mu_t$, by calculating the output current for every input code using a behavioral simulator (Section 5.4.6) which takes as input the deterministic mismatch, the output resistance, and the input voltage of all the components. Next, we compute the covariance $\Sigma_t$ using

$$\Sigma_t = \left[\frac{\partial t}{\partial K_M}\right] S_M^2 \left[\frac{\partial t}{\partial K_M}\right]^T + \left[\frac{\partial t}{\partial K_1}\right] \Sigma_{K_1} \left[\frac{\partial t}{\partial K_1}\right]^T + \left[\frac{\partial t}{\partial K_2}\right] \Sigma_{K_2} \left[\frac{\partial t}{\partial K_2}\right]^T \tag{5.18}$$

where vector parameters $K_1$ and $K_2$ correspond to the mismatches of the array of outputs in the linear and binary stages, respectively. Matrix parameters $\Sigma_{K1}$ and $\Sigma_{K2}$ correspond to the covariance of the mismatches of the array of outputs in the linear and binary stages, respectively.

From parameters $\mu_t$ and $\Sigma_t$, we calculate integral nonlinearity using equations given in Section 4.3.

### 5.4.6 Software Implementation

For this example, we extend the software implementation of the D/A described in Section 5.3.3 from 5 bits to 10 bits. Due to the increase in resolution, parasitic resistance effects are no longer negligible. We describe global parasitic resistances by a *linear* network of resistors, $R = \{R_1, R_2, \ldots\}$, the linearized components, and ideal voltage and current sources. Then, we compute the new DC bias using modified node analysis[44] implemented with a sparse matrix solver[27]. We repeat the bias calculations for all 1024 input codes to get a new nominal output current vector, $\mu_t'$. Assuming that parasitic effects do not affect process variations, the new output current

vector is distributed as,

$$t' \sim normal(\mu_t', \Sigma_t) \tag{5.19}$$

from which we estimate the new system performance using formulae given in Section 4.3.

Furthermore, we create an adjoint network by building a network using the adjoint representation of elements[12]. Then, we find the sensitivity matrix of the output current with respect to the resistances, $\frac{\partial t}{\partial R}$, by solving the adjoint network with the appropriate ideal voltage and current sources.

### 5.4.7 Experimental Results

We verified an "industrial strength" 10 bit D/A[10] bottom-up from layout to system level performance on a DEC 5000/125. By using the adjoint technique for sensitivity analysis on the linearized chip-level interconnect network, we computed efficiently the INL performance sensitivity with respect to all chip level routing resistances. According to our goal of constraint-driven layout synthesis, the layout synthesis tools use the sensitivity information intelligently to assemble a global layout from a collection of component layouts. Figure 5.10 shows the sensitivity of INL with respect to the resistance from the ground pad to the linear array.

To confirm the accuracy of the new approach, we compare our results with SPICE simulation of the final extracted circuit. Due to the long simulation times for SPICE, we consider only one process variable, the main transistor width of the mirror, when computing the $\pm 3\sigma$ worst case INL (4.23). The behavioral simulation results are verified against SPICE results for 18 random input codes in Figure 5.11. Each data point on the graph is a comparison, and a perfect match will fall on the 45 degree line. Behavioral simulation achieved an accuracy of 0.005LSB. Furthermore, results from behavioral simulation for all 1024 input codes plotted along with results from SPICE on Figure 5.12 show good agreements.

In our example, SPICE takes 118 DEC 5000/125 CPU seconds per input code. For behavioral simulation, the characterization of the components takes 1000 CPU seconds, the linear network solver takes 100 CPU seconds, and the behavioral simulation takes 100 CPU seconds. On average for 1024 input codes, behavioral simulation takes 1.2 CPU seconds per input code. Therefore, the speed advantage is

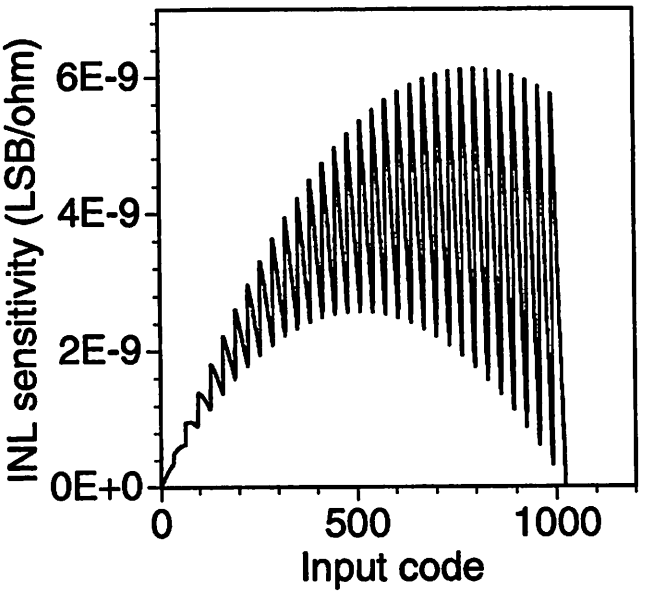


Figure 5.10: INL sensitivity to R from Vss pad to linear array

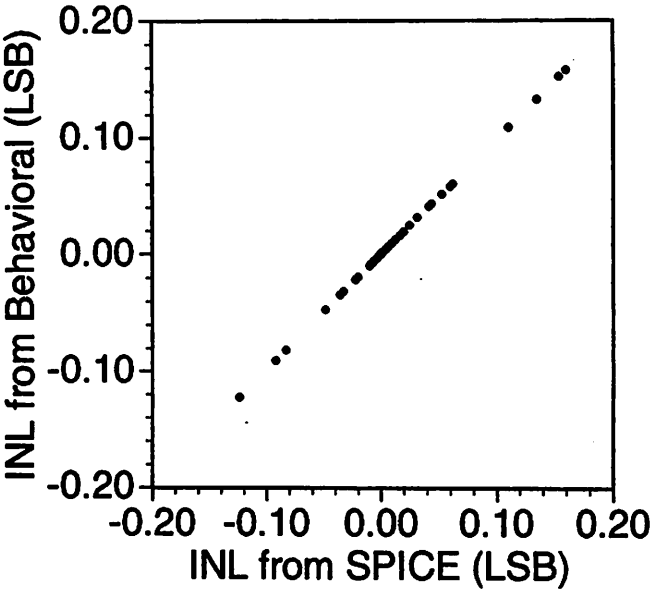


Figure 5.11: Accuracy comparison between SPICE and behavioral simulation

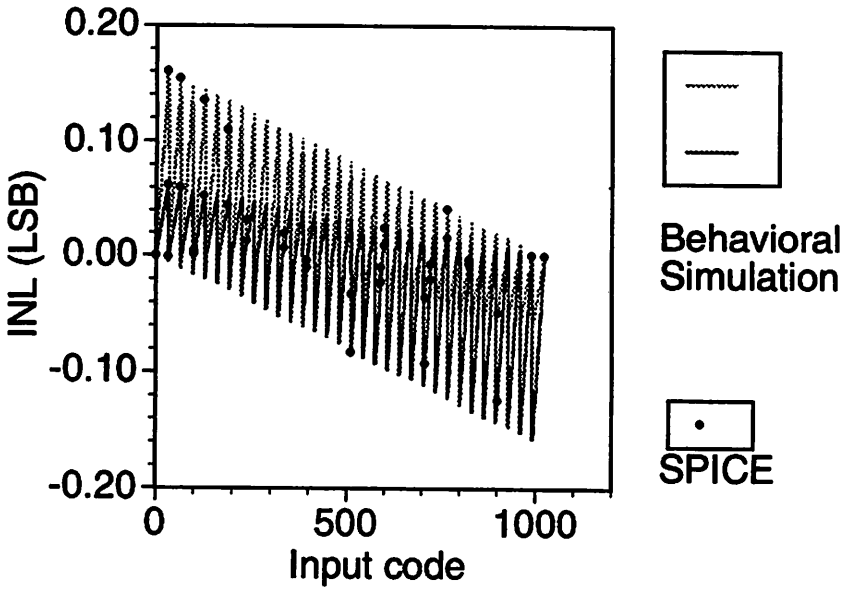


Figure 5.12: $\pm 3\sigma$ INL bounds compared with SPICE

100 *for simulation of one process variable.* For $M$ process variable, SPICE would take $59 \times 2^M$ CPU seconds per input code, while behavioral simulation would take about the same time per input code because the CPU time is dominated by the characterization of the components. For our circuit with 600 transistors, $M$ is at least 600, so the speed advantage of behavioral simulation is proportional to $2^{600}$. Notice that traditionally, designers simulate a subset of the process corners using SPICE because of the large number of process corners. But no general algorithm has been proposed for choosing the subset.

As a result of the speed advantage, we verified our D/A performance under many types of process variations. Assuming typical process data such as transistor width and length standard deviation of $0.05\mu m$ and threshold variation of $\frac{0.0623Vm}{\sqrt{WL}}$, we computed the $\pm 3\sigma$ INL bounds (Figure 5.13) for the converter. From our theory, the INL of any fabricated chip will likely fall within the bounds. The maximum of the INL bounds is 1.5LSB which satisfy the constraints of 2.0LSB.

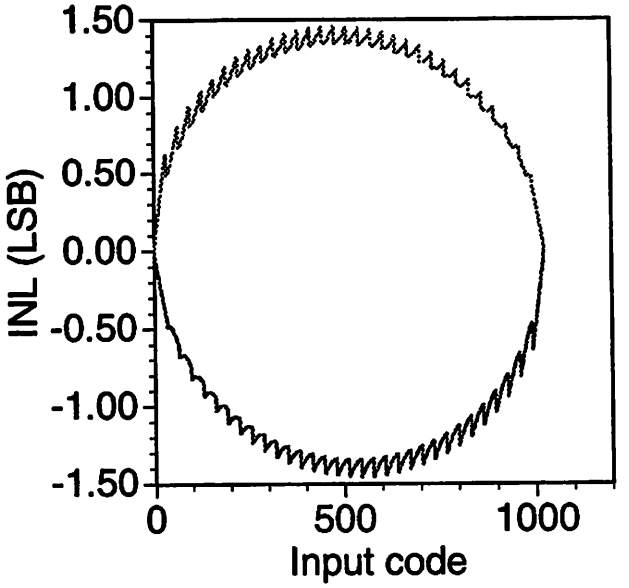


Figure 5.13: $\pm 3\sigma$ INL bounds

## 5.5 Conclusion

In analog system design, final verification is crucial to guarantee functionality of the entire circuit. The traditional system verification strategy based on SPICE is too slow for large systems, especially when worst case analysis is used to simulate process variations. In our approaches, we take advantage of the hierarchical decomposition of the system into components. In the first approach, we extract each component individually, verify it under ideal bias conditions, and fit parameters for its behavioral model. Then, we simulate the system at the *behavioral* level using behavioral models only. Behavioral simulation results compare well with Monte Carlo SPICE simulations. The approach is exact and works well for architectures with negligible parasitic loading effects.

To verify the system in the presence of parasitic loading between components, we propose an approximate strategy. We extract each component individually, verify it under ideal bias conditions, fit parameters for its behavioral model, linearize the components at the operating point, substitute the linearized component in the interconnect network, find the changes in the bias conditions, and estimate the performance deviation due to bias changes using a first order Taylor approximation. From experimental

results, we have validated the verification approach for a 10 bit interpolative D/A to be within 0.005 LSB compared with SPICE. From CPU time analysis, this verification approach can be many orders of magnitude faster than SPICE.

# Chapter 6

# Data Converter Testing using Behavioral Simulation

## 6.1 Background

Data converters are commodity products, yet their testing is very expensive. Test engineers use expensive equipment to take accurate measurements against background noise to characterize converters over a variety of operating conditions such as temperature and supply ranges. Furthermore, engineering time needed to develop unique test software for each product adds substantially to the overall testing cost.

A behavioral model that captures the converter behavior can provide critical information for design engineers to evaluate the testability of the design at an early design stage and for test engineers to choose the optimum testing strategy after design. From the information contained in the behavioral model, engineers can evaluate the tradeoffs between *test set size*, *test coverage*, *detection thresholds*, *measurement noise*, *chip performance*, and *estimated yield*. To achieve this goal, we propose a new converter testing strategy for data converters from a behavioral model[33]. We present previous work in Section 6.2, a new testing strategy in Section 6.3, a yield analysis algorithm in Section 6.4, and experimental results in Section 6.5.

## 6.2 Previous Work

In [51, 53] a *linear model* for data converters along with a test selection strategy was presented. The model for an $N$ bit A/D converter represents the $2^N - 1$ transition points of the converter as a linear function of the component errors,

$$t = S_v v \tag{6.1}$$

where $t$ is an $2^N - 1$-dimensional vector that represents the transition points, $v$ is an $m$-dimensional vector that represents the component errors, and $S_v$ is the $2^N - 1$ by $m$ sensitivity matrix with full column rank.

Assuming no measurement noise, only $m$ linearly independent test points are required to estimate the $m$ model coefficients. The objective is to find the optimal subset of the full set of $2^N - 1$ test points that minimizes the prediction variance of $m$. This problem falls in the category of **Optimal Design of Experiments**. It has been shown under the **D-Optimality criterion**[3] that in the limit where the number of test points is large, an optimal selection is $S'_v$, an $m$ by $m$ matrix formed by selected rows of $S_v$. The $m$ model coefficients, $v$, are then computed as

$$v = [S'_v]^{-1} t' \tag{6.2}$$

where $t'$ are the selected measurements. Under this criterion, the prediction variance of $v$ is minimized by maximizing $|S'^T_v S'_v|$ where $|\cdot|$ is the determinant.

While the D-Optimality criterion gives the optimal solution, the computational complexity prevents its use for large models. Stenbakken[54] introduced an algorithm for a near optimal solution based on QR factorization with pivoting. The algorithm first chooses the row of $S_v$ with the largest norm, orthogonalizing all remaining rows to it using a modified Gram-Schmidt orthogonalization procedure, then choosing the row of largest norm of those remaining, and repeating the process. The algorithm records the row indices during pivoting, and chooses the $m$ pivot rows of the initial $S_v$ for $S'_v$.

The main drawback of the previous testing strategy for data converters lies in the difficulty of making accurate measurements. For example, an A/D produces a discrete output code from a continuous input. Under the previous testing strategy, we measure the transitions between adjacent output codes using a bisection search

algorithm[43]. In this algorithm, we guess initial lower bound, $t_l(i)$, and upper bound, $t_u(i)$, for the transition $t(i)$. If the outputs of the converter for $t_l(i)$ and $t_u(i)$ are $o_1 \leq i$ and $o_2 \geq i+1$, respectively, then $t(i)$ must lie within the bounds. By moving the bounds closer iteratively until convergence, we can determine $t(i)$ up to an accuracy limited by the measurement noise. Because the algorithm has complexity of $O(log_2(\frac{1}{\alpha}))$, where $\alpha$ is the required accuracy in fraction of the smallest step size (LSB), we need many iterations requiring long test time for accurate measurements.

Moreover, measurement noise poses a significant problem because it corrupts the estimated parameters. To reduce measurement noise problems, Hemink[23] presented algorithms for testability analysis and optimal test selection in the presence of measurement noise. On the other hand, Souders[52] proposed adding more test points to reduce noise effects by creating an overdetermined system of equations to be solved by least square techniques. Although additional test points reduce noise effects, no algorithm has been presented so far for determining the number of extra tests needed or selecting the additional tests based on the amount of measurement noise.

Another drawback of the previous approach is the large amount of computation required for solving the system of equations for each chip under worst case operating conditions such as temperature and supply extremes. Also, the lack of consideration for tradeoffs between test set size, chip performance, test coverage, and yield reduces the usefulness of the previous strategies.

## 6.3 New Testing Strategy

Data converters are binned with respect to their performance and sold at prices accordingly; therefore, we need a strategy to tradeoff tested chip performance and test costs, as well as other parameters such as test coverage, measurement noise, and estimated yield. Decreasing test coverage, smaller measurement noise, stricter detection thresholds, and lower chip performance would require smaller test set and less test time. Stricter detection thresholds, on the other hand, would decrease estimated yield. An optimal choice of these parameters will lead to the most cost effective testing solution. In this section, we present a testing strategy that allows tradeoffs between these parameters.

Our strategy focuses on testing all DC performance of Nyquist data convert-

ers including offset error, full scale gain error, integral nonlinearity, and differential nonlinearity. The behavioral model for data converters presented in Chapter 4 is useful for testing DC performance. In contrast, the model proposed previously in [53] is deterministic and hence does not represent statistical effects. We represent electronic noise as well as process variation effects. Moreover, our model has distributions for INL and DNL. We will show that these are essential to developing a testing strategy that tradeoffs between test set size, test coverage, detection thresholds, measurement noise, chip performance, and estimated yield.

### 6.3.1 Measurement and Detection Threshold

As mentioned in Section 6.2, previous testing strategies[53, 23] have difficulty in accurate measurements of circuit performance such as A/D converter transition points in the presence of measurement noise. To circumvent the problem, we propose a simpler measurement that is robust against noise. In contrast to the traditional approach where the exact value of a performance parameter such as a transition point is measured, we verify that *a circuit performance parameter falls within certain detection thresholds in the presence of measurement noise.* For example, the test for A/D converter transition point, $t(i)$, consists of only *two* A/D conversions for input thresholds $t_l(i)$ and $t_u(i)$. A successful test corresponds to outputs of the converter for $t_l(i)$ and $t_u(i)$ being $o_l \leq i$ and $o_u \geq i+1$, respectively. In this case, the transition point plus uncertainty due to noise, $t(i) + \delta$, will lie within the range given by $(t_l(i), t_u(i))$, where $\delta$ is the uncertainty due to noise. As a result, the following inequalities hold,

$$t_l(i) \leq t(i) + \delta \leq t_u(i), \tag{6.3}$$

$$t_l(i) - \delta \leq t(i) \leq t_u(i) - \delta, \tag{6.4}$$

$$t_l(i) - max(\delta) \leq t(i) \leq t_u(i) - min(\delta). \tag{6.5}$$

Assuming noise $\delta$ falls in the range of $(-\Delta, \Delta)$, $t(i)$ satisfies the following inequality in the presence of measurement noise (Figure 6.1),

$$t_l(i) - \Delta \leq t(i) \leq t_u(i) + \Delta. \tag{6.6}$$

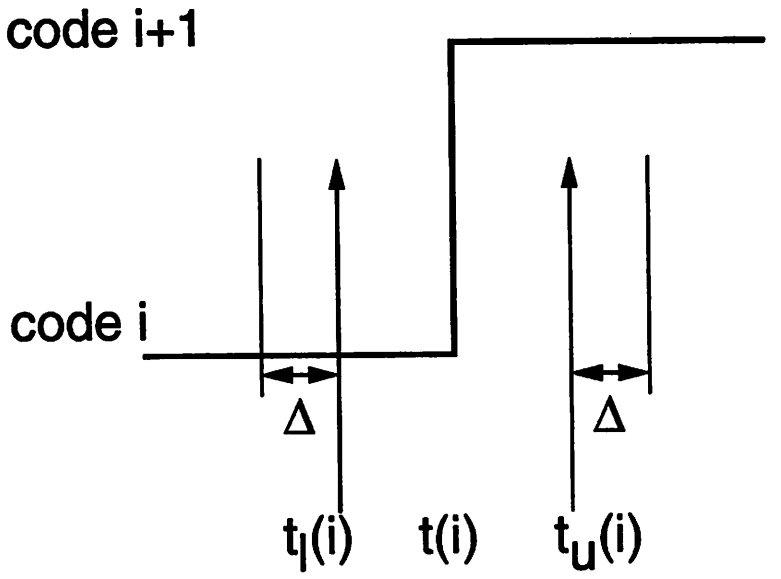


Figure 6.1: Detection thresholds

### 6.3.2 Testing Gain and Offset Errors

Successful tests for gain and offset errors establish bounds on these parameters. Gain and offset errors (Definitions 4.3.3 and 4.3.4) are linear functions of the first and last transition points, $t(1)$ and $t(n)$, respectively. We can establish bounds on these errors by choosing tight bounds such as $t_l(1)$, $t_u(1)$, $t_l(n)$, and $t_u(n)$. For example, bounds on offset error, $V_{os}$, and full scale gain error, $V_{ge}$, are

$$t_l(1) - \Delta - L(1) \leq V_{os} \leq t_u(i) + \Delta - L(1), \tag{6.7}$$

$$t_l(n) - L(n) + L(1) - t_u(1) - 2\Delta \leq V_{ge} \leq t_u(n) - L(n) + L(1) - t_l(1) + 2\Delta. \tag{6.8}$$

### 6.3.3 Testing Nonlinearity Errors

Successful tests for integral and differential nonlinearity establish bounds on these parameters for *all* inputs. For a 10 bit converter, there are a total of 1024 possible inputs. Fortunately, it suffices to test a subset, $T$, of the set of all inputs, $S$, due to correlations between outputs of data converters. That is, we can establish bounds on all nonlinearity errors by performing tests in $T$.

We can bound integral and differential nonlinearities for individual inputs using the proposed measurement scheme. The user specifies the **INL test thresh-**

**old,** $s_b$**, from which we compute the needed test input thresholds using the following equations.**

From (4.17) and (6.6), we derive inequalities for the integral nonlinearity, $s(i)$,

$$s_l(i) \leq s(i) \leq s_u(i), \tag{6.9}$$

where

$$s_l(i) = \left[\frac{L(n) - L(1)}{t_u(n) - t_l(1) + 2\Delta}\right](t_l(i) - t_u(1) - 2\Delta) + L(1) - L(i) = -s_b, \tag{6.10}$$

$$s_u(i) = \left[\frac{L(n) - L(1)}{t_l(n) - t_u(1) - 2\Delta}\right](t_u(i) - t_l(1) + 2\Delta) + L(1) - L(i) = s_b, \tag{6.11}$$

and $s_b$ is the INL threshold in the presence of noise. We use (6.10) and (6.11) to compute the test inputs $t_l(i)$ and $t_u(i)$ where $i \in 2 \ldots 2^N - 2$. As a rule, the threshold, $s_b$, should be set as follows,

$$s_b = s_b' + 2\Delta, \tag{6.12}$$

where $s_b'$ is the INL threshold for ideal noiseless measurements, due to the following approximations,

$$s_l(i) \approx (t_l(i) - t_u(1) - 2\Delta) + L(1) - L(i) \approx -s_b' - 2\Delta \tag{6.13}$$

$$s_u(i) \approx (t_u(i) - t_l(1) + 2\Delta) + L(1) - L(i) \approx s_b' + 2\Delta \tag{6.14}$$

The approximations show a linear dependence of the threshold, $s_b$, on noise and help designers set the INL threshold based on noise estimates. Notice that (6.13) and (6.14) are only used as a rule for designers. In testing, we use the exact inequalities (6.10) and (6.11) to compute the test inputs $t_l(i)$ and $t_u(i)$ where $i \in 2 \ldots 2^N - 2$. Similar exact inequalities (6.10) and (6.11), and approximate inequalities (6.13) and (6.14) can be derived for differential nonlinearity.

### 6.3.4 Test Selection for Nonlinearity Errors

Each successful test $i$ establishes upper and lower bounds on $s(i)$. Due to correlations between the transition points (Section 4.2), INL at other points may also be bounded. In test selection, we choose the best subset, $T$, of test points, $S$, such that

all untested INL's are bounded by a **maximum INL performance**, $s_m$. Let $U$ be the set of unbounded INL. **Test coverage** is defined as

$$1 - \frac{|U|}{|S|} \tag{6.15}$$

where $|\cdot|$ denotes the cardinality of a set. **Full coverage** refers to the case where $U$ is empty and hence test coverage is one.

We formulate the test selection problem as follows. Let $\Delta$ be the magnitude of the measurement noise in (6.6), $s_m$ be the specified maximum INL performance, $s_b'$ be the INL threshold in the absence of noise, $s_b = s_b' + 2\Delta$ be the INL threshold in the presence of noise, $T$ be the test set, choose the minimum $T$ such that

$$-s_m \leq -s_b \leq s(i) \leq s_b \leq s_m, i \in T \tag{6.16}$$

$$-s_m \leq s(i) \leq s_m, i \notin T \tag{6.17}$$

where $s$ is given by a linear transformation of some independent variables in (4.22),

$$s = U_s c_s + \mu_s \tag{6.18}$$

where $c_s \sim normal(0, \Sigma_{cs})$ is a zero mean multivariate normal distribution with $r_s$ statistically *independent* variates.

Due to linear dependence, we use **linear programming**[43] to check the bounds on untested INL, $s(i)$, in (6.17). We formulate the problem as follows. We check that

$$\max_{c_s} s(i) \leq s_m, i \notin T \tag{6.19}$$

$$\min_{c_s} s(i) \geq -s_m, i \notin T \tag{6.20}$$

such that

$$-s_b \leq s(j) \leq s_b, j \in T \tag{6.21}$$

$$-k\sqrt{\Sigma_{cs}(i,i)} \leq c_s(i) \leq k\sqrt{\Sigma_{cs}(i,i)}, i = 1 \ldots r_s, \tag{6.22}$$

where $k$ is a user specified parameter to limit the distribution, $c_s$. The limit prevents signatures with small magnitudes from being scaled unrealistically with unbounded $c_s$ in the linear program. Typically, we choose $k = 5$ for a $10\sigma$ statistical limit. If $c_s$ is unbounded, then the statistical information provided by the distribution of $c_s$ is lost.

In a different context of delay fault testing[28], a similar problem formulation has been applied to check bounds on delays of digital circuits. In both formulations, the problems seem to be analytically intractable. For example, the simpler problem of checking whether $|T|$ tests provide sufficient bounds on $s$ would require an exponential complexity of

$$O(|S-T|2^i). \tag{6.23}$$

where the exponential dependence is due to the linear programming step and $i = \begin{pmatrix} |S| \\ |T| \end{pmatrix}$, where $\begin{pmatrix} x \\ y \end{pmatrix}$ is the number of ways $y$ objects can be chosen from $x$ different objects.

As a result, the optimum test selection problem is likely intractable. Thus, we propose a heuristic solution. First, we rank the tests according to the algorithm proposed in [54] based on QR factorization with pivoting.

- (a) Choose the row of $U_s$ with the largest norm,
- (b) Orthogonalize all remaining rows to it using a modified Gram-Schmidt orthogonalization procedure,
- (c) Choose the row of largest $L_2$ norm of those remaining,
- (d) Repeat (b) and (c) until all rows are chosen,
- (e) Record the row indices of all chosen rows, and list them in the order they were chosen.

As a result, we produce a list of INL's with decreasing order of importance where the $j^{th}$ item on the list is $s(i)$.

Next, we use the following algorithm to find $T$.

- (a) Let $j = 1$, and $T$ be empty.
- (b) If $j$ is larger than the number of test points, done.
- (c) Pick the $j^{th}$ item on the list, $s(i)$. Check $s(i)$ for bounds specified in (6.17) using linear programming.
- (d) If the linear programming is infeasible, quit due to performance specifications too tight.

- (e) If the bounds are not satisfied, add $i$ to $T$.
- (f) $j = j + 1$ and go to (b).

Unless the specified performance is infeasible, the algorithm guarantees full coverage since every INL is either bounded or constrained by a test. To tradeoff test size against coverage, the algorithm should stop after a certain coverage is achieved.

## 6.4 Yield Analysis

**Yield**, $Y$, is defined as the ratio of the number of chips working correctly to the total number of chips. As a lower bound on $Y$, the **estimated yield**, $\hat{Y}$, is the ratio of the number of chips passing all tests under our proposed strategy to the total number of chips. Assuming the chip passes gain and offset error tests, estimated yield due to INL tests is given by

$$\hat{Y} = \int_R f(s)ds \tag{6.24}$$

where $f(s)$ is the probability density function of $s$ and

$$R : -s_b' \leq s(i) \leq s_b', i \in T \tag{6.25}$$

As a result, stricter test thresholds would decrease estimated yield, while increasing test coverage. Thus, yield analysis for a particular test set and thresholds provides critical information for engineers to improve design and to choose the optimal testing strategy.

We estimate yield using Monte Carlo integration according to the following algorithm.

- (a) $i = j = 0$, and $k$ is the number of samples.
- (b) Generate a realization of $c_s$, $\hat{c}_s$, from the distribution of $c_s$ using a random number generator.
- (c) Compute a realization of $s$, $\hat{s}$, using (4.22) and $\hat{c}_s$.
- (d) Check if $\hat{s}$ is in $R$ using (6.25). If so, $j = j + 1$.
- (e) $i = i + 1$. If $i < k$, go to (b), else done.

Estimated yield, $\hat{Y}$, is then computed as $\hat{Y} = \frac{i}{k}$. Because Monte Carlo integration is used, our estimate $\hat{Y}$ improves with the number of trials, $k$. The variance of our estimate is inversely proportional to $k^2$.

## 6.5 Experimental Results

We have applied the new strategy to the automatic test generation for a 10 bit current-switched, interpolative D/A[10] shown in Figure 6.2. We used a bottom-up verification strategy[29] to compute the D/A behavioral model parameters from SPICE netlist and expected process variation parameters. Behavioral simulation of the D/A and automatic test generation have been integrated into a single software package to facilitate **design-for-testability**.

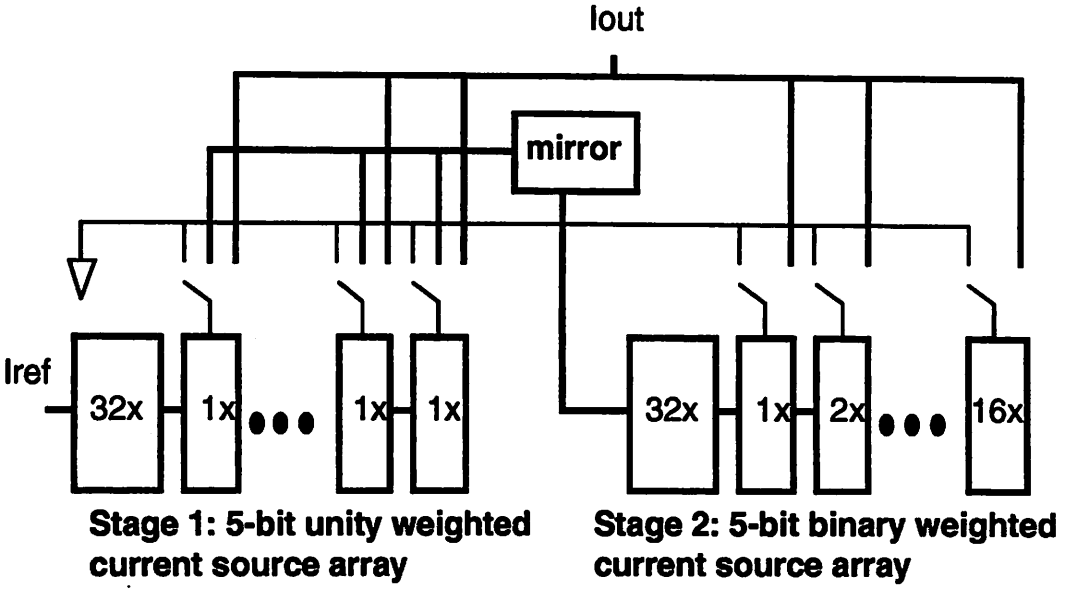


Figure 6.2: 10 bit current-switched, interpolative D/A architecture

Using the proposed testing strategy, we first determined the tradeoff between maximum INL performance, $s_m$, and test set size (Figure 6.3) for $s_b' = 1.5$ LSB, $\Delta = 0.1$ LSB, and full coverage. Notice that test set size varies inversely with INL as expected. Also, the size drops below the rank of 38 for INL larger than 3.2 LSB, where the **rank** is the number of INL signatures. This occurs because some INL signatures with small magnitudes do not contribute significantly to large INL errors. In contrast, previous

testing strategy[53] uses the same number of tests as the rank because such strategy does not take into consideration the magnitudes of the signatures. In Figure 6.4, we plotted the DEC ALPHA CPU times for test generation against INL. Notice that test generation times drops significantly for lower performance chips.

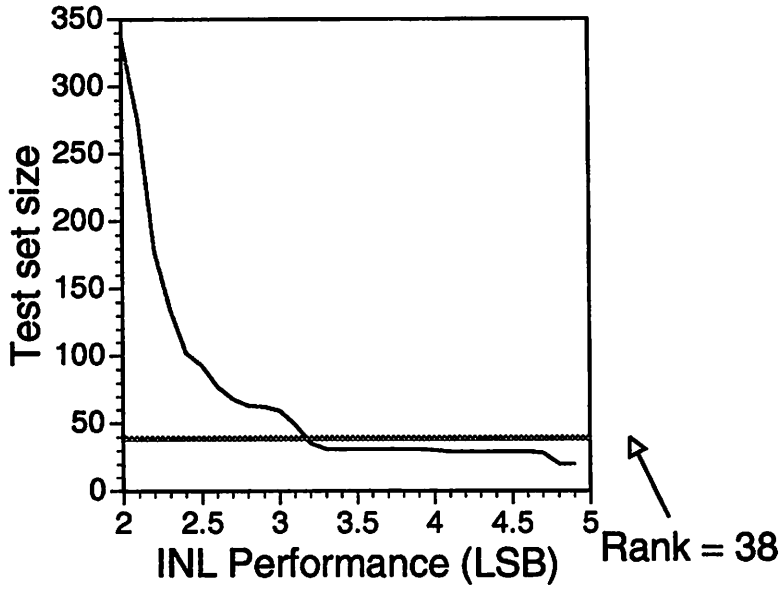


Figure 6.3: Tradeoff between test set and INL performance

Next, we determined the tradeoff between measurement noise and test set size (Figure 6.5) for $s_b'$ = 1.5 LSB, $s_m$ = 3.0 LSB, and full coverage. Notice that the size increases somewhat linearly with noise. The average DEC ALPHA CPU times for each data point is 600 seconds.

Also, we determined the tradeoff between INL threshold in the absence of noise, $s_b'$, test set size, and estimated yield for $\Delta$ = 0.1 LSB, $s_m$ = 3.0 LSB, and full coverage. Figure 6.6 plots test set size against $s_b'$, while Figure 6.7 plots estimated yield against $s_b'$. Together, they showed that stricter thresholds decrease estimated yield and test size, yet do not reduce the test size under 31. The reason is that for this level of INL performance, 31 signatures contribute to the INL, while the remaining 7 signatures are insignificant. Thus, 31 or more independent tests are needed to guarantee INL performance regardless of INL thresholds. As a result, we should choose the highest threshold, $s_b$ = 1.3 LSB, such that test size is 31 for highest estimated yield and better noise tolerance. The average DEC ALPHA CPU times for each data point is 582

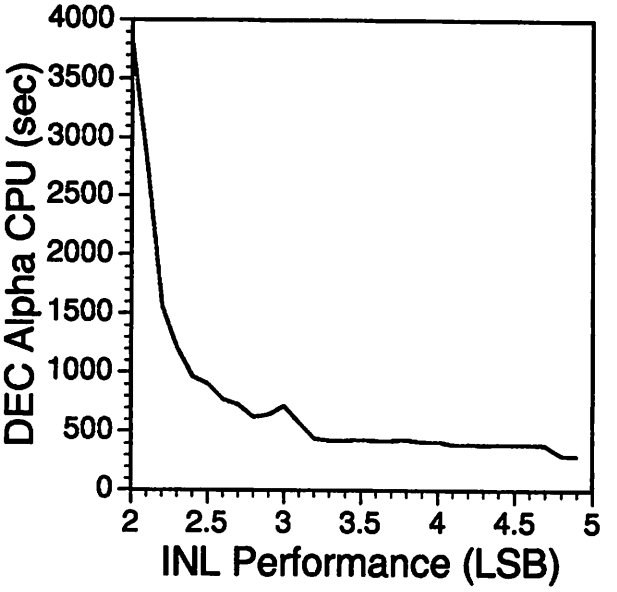


Figure 6.4: Tradeoff between test generation time and INL performance

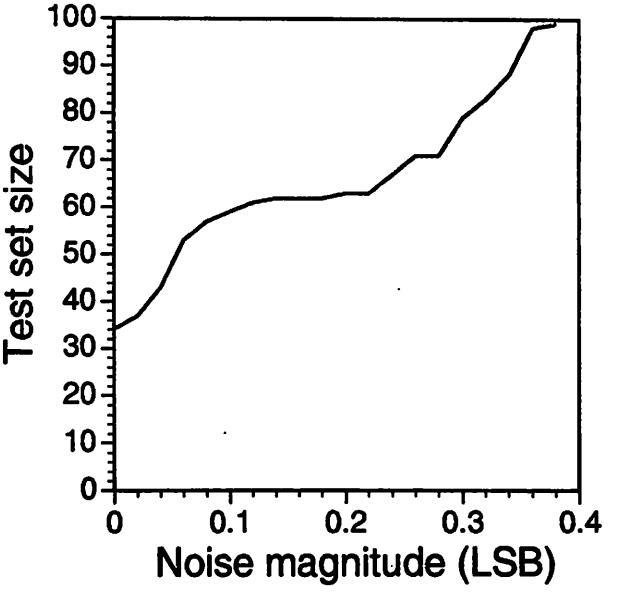


Figure 6.5: Tradeoff between test set and measurement noise

seconds.

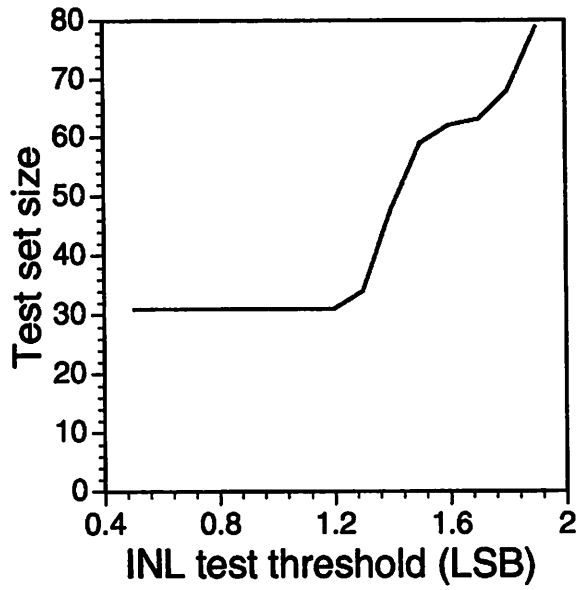


Figure 6.6: Tradeoff between test set and INL test threshold

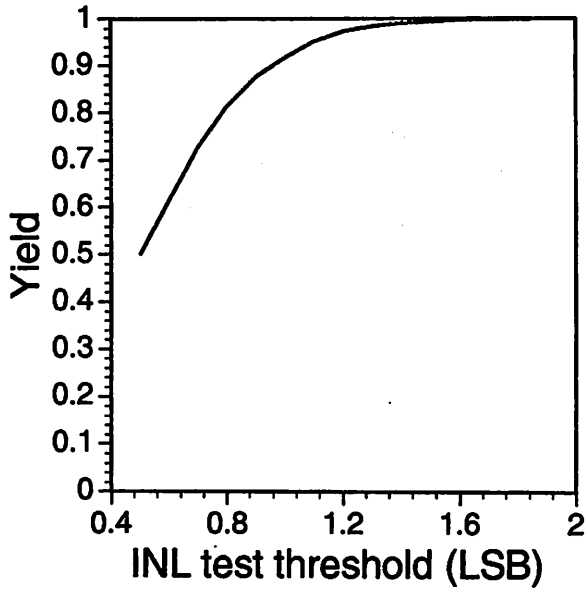


Figure 6.7: Tradeoff between estimated yield and INL test threshold

Finally, we determined the tradeoff between maximum DNL performance, $d_m$, and test set size (Figure 6.8) for DNL test bound in the absence of noise $d'_b = 1.5$ LSB, $\Delta = 0.1$ LSB, and full coverage. Notice that test set size varies inversely with DNL as

expected. The average DEC ALPHA CPU times for each data point is 303 seconds.

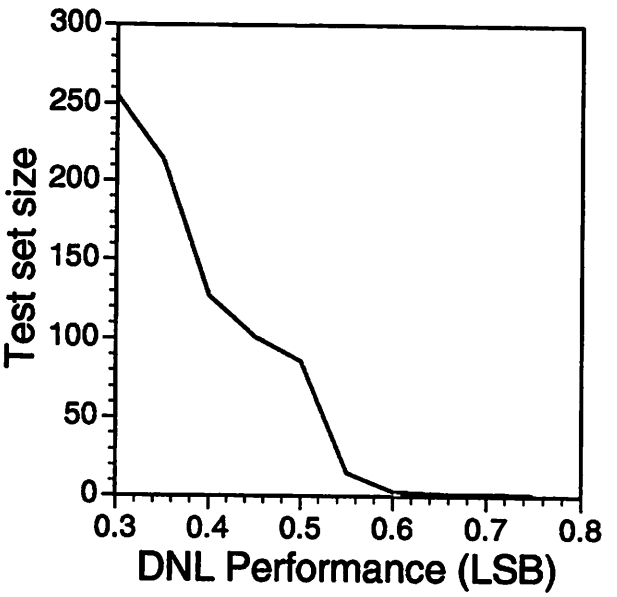


Figure 6.8: Tradeoff between test set and DNL performance

Next, we applied our strategy to the automatic test generation for an 8 bit cyclic A/D converter shown in Figure 6.9. We computed behavioral model parameters from a high level architectural description and expected process variation parameters. In Figure 6.10, we showed the tradeoff between maximum INL and test set size for $s_b' = 0.8$ LSB, $\Delta = 0.05$ LSB, and full coverage. Notice that, in general, the test size for this type of A/D converter is very small because the outputs are highly correlated. The average DEC ALPHA CPU times for each data point is 2 seconds.

## 6.6 Conclusion

We have presented a strategy for testing all DC performance of Nyquist data converters including offset error, full scale gain error, integral nonlinearity, and differential nonlinearity. Assuming that the behavioral model is correct, the testing strategy is exact, but the yield estimate is approximate.

In contrast to previous testing strategies based on linear models that require accurate measurements of circuit performance in the presence of measurement noise, our strategy uses a simpler measurement to verify that a circuit performance param-

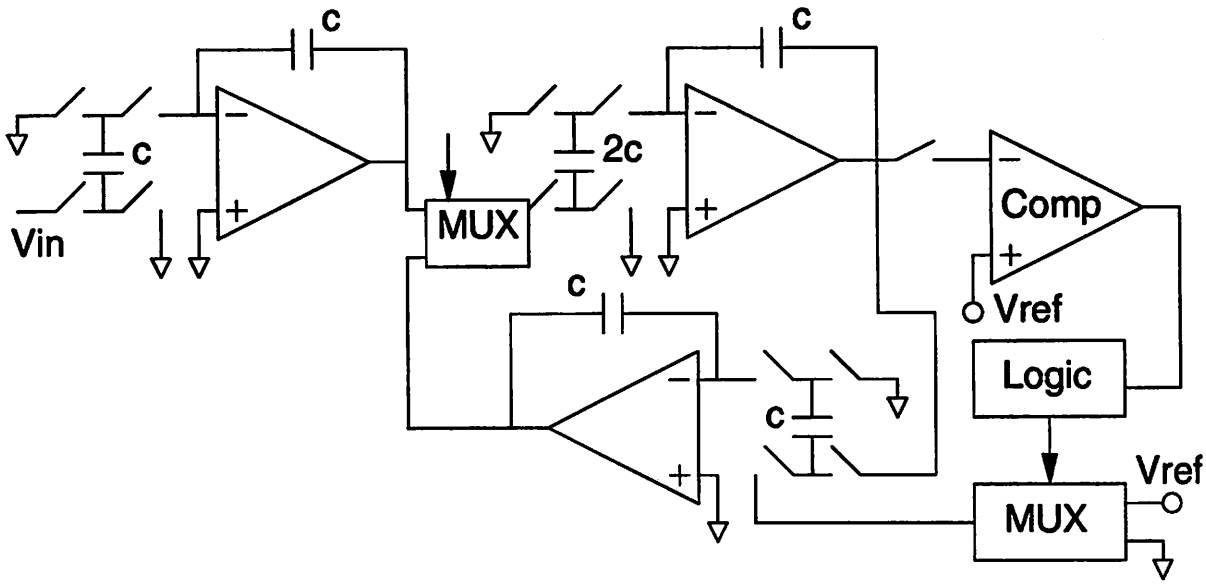


Figure 6.9: 8 bit cyclic A/D architecture

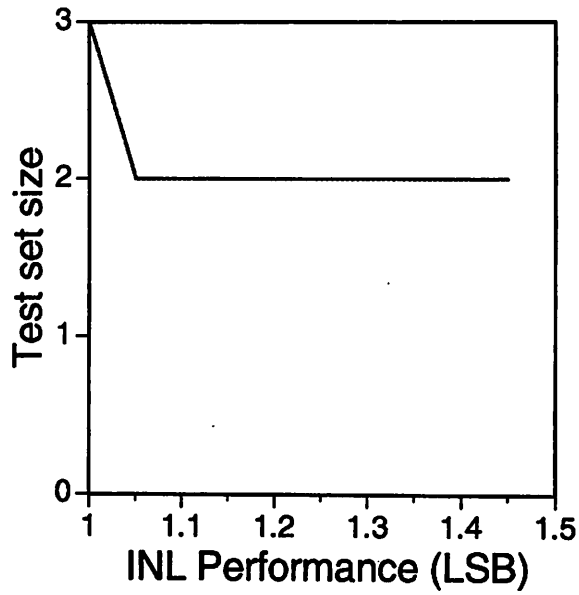


Figure 6.10: Tradeoff between test set and INL performance

eter falls within a certain range in the presence of measurement noise. Using the proposed strategy and behavioral modeling of the device under test, we evaluated tradeoffs between test set size, detection thresholds, measurement noise, chip performance, and estimated yield. We showed that smaller measurement noise, stricter detection thresholds, and lower chip performance would require smaller test set and reduce test time. Stricter detection thresholds, on the other hand, would decrease estimated yield.

# Chapter 7

# Behavioral Simulation for Noise in Mixed-Mode Sampled Data Systems

## 7.1 Background

The simulation of mixed analog and digital systems is crucial in system verification as more analog and digital circuits are integrated on the same integrated circuit in data acquisition, automotive, or disk drive electronics applications. For these systems, there is a trend to reduce system power consumption by reducing the supply voltage or minimizing parasitic capacitances. One of the fundamental limit to these power reduction techniques is the electronic noise inherent in the mixed analog/digital subsystems. While reducing the supply voltage reduces the allowable signal power, the noise power due to physical effects remain the same. Due to the potential decrease in signal-to-noise ratio, it is desirable to simulate the mixed-mode system for noise performances. Unfortunately, there are no noise simulators that can analyze noise effects for mixed-mode systems.

To develop such a simulator, we follow the **behavioral simulation** paradigm in which circuits are modeled mathematically as in [33, 19, 30, 31]. In this chapter, we present a new **noise behavioral model** and a **direct noise analysis** approach for mixed-mode systems. Using the appropriate model, there is no need for circuit

or macromodel simulation, but a **direct algebraic** approach where the objects being manipulated are noise characterizations.

In Section 7.2, the problem is defined. In Section 7.3, the traditional Monte Carlo method for noise simulation is described. Then, new behavioral models (Sections 7.4 and 7.5) and new approach for noise analysis (Section 7.4) are presented with experimental results (Section 7.7).

## 7.2 Problem definition

The class of circuits under consideration includes sampled-data systems that have mixed analog/digital inputs and outputs. This includes, but is not limited to, A/D converters, D/A converters, and receivers of digital signals. The objective of the simulation is to determine the noise effects due to electronic noise. Thermal, flicker, and shot noise in the transistors and thermal noise in resistors contribute to the noise of the circuit. Traditionally, electronic noise is modeled with a Gaussian random process described by its power spectral density[20]. From *the noise sources*, *the system architecture*, and *the deterministic input*, we seek the distribution of the output. For example, we seek the continuous output distribution of a D/A converter, the bit error rate of a receiver, or the output code distribution of an A/D converter. To illustrate the problem, Figure 7.1 shows a switched capacitor implementation of a cyclic A/D converter. Due to noise effects, the output of the comparator is wrong with a non-zero probability. Since the $N$-bit digital output code is a logic function of $N$ successive outputs of the comparator, we seek the probabilities for each of the $2^N$ possible output codes given a deterministic input.

## 7.3 Previous work

Traditionally, the SPICE-like simulators[44] analyze analog circuit noise in the frequency domain. SPICE linearizes the circuit at the operating point, adds sinusoidal sources in parallel to the noisy elements, and analyze the resulting AC equivalent circuit. The approach has two problems. Firstly, many components in mixed-mode systems, such as the comparator, have discontinuous transfer functions and cannot be linearized. Secondly, mixed-mode systems are usually non-steady state (e.g. an

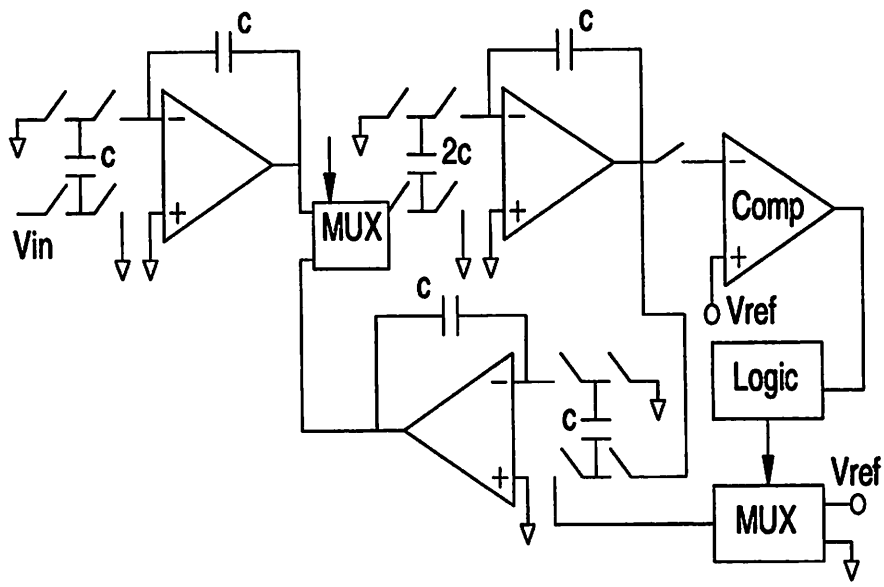


Figure 7.1: Cyclic A/D

A/D produces an output code after a *finite* time), so an operating point does not exist for the circuit. Simulators such as [59] analyze noise in switched-capacitor circuits using linear systems theory, but fail to handle noise in mixed-mode switched-capacitor circuits as mixed-mode circuits are in general nonlinear.

Without simulators for noise effects, designers use the Monte Carlo noise simulation technique. In this technique, they model the circuit as a deterministic network of components, and inject random numbers into the signal path. They run the system many times, and hope to get a wrong output code by chance.

Figure 7.2(a) illustrates a sample-and-hold, a major component in a sampled-data system. The electronic noise source, $v$, has **power spectral density**, $S_v(f)$, where $f$ is frequency. The component is modeled by a delay with an additive noise, $n$, as shown in Figure 7.2(b). At each simulation time point, the simulator takes samples of $n$ and add them into the signal before the delay.

We compute the distribution of $n$ from the noise power spectral density, $S_v(f)$. Due to the lowpass filter formed by R and C, the voltage on the capacitor when the switch, S1, is closed has power spectral density given by

$$\frac{S_v(f)}{1+(2\pi fRC)^2} \tag{7.1}$$

As S1 switches, thus sampling the voltage onto the capacitor, we compute the equiva-

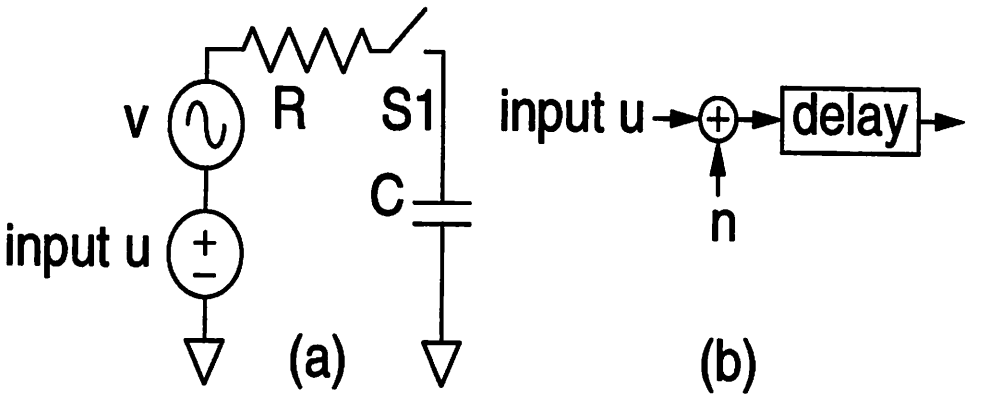


Figure 7.2: Sample-and-hold

lent **sampled noise power spectral density** on the capacitor by shifting and adding the original spectral density. From the sampling theorem, the sampled noise power spectral density, $S_d(f)$, is

$$S_d(f) = \sum_{k=-\infty}^{\infty} \frac{S_v(f - kf_s)}{1 + (2\pi(f - kf_s)RC)^2} \tag{7.2}$$

where $f_s$ is the switching frequency. The inverse Fourier transform of $S_d(f)$ gives the time domain discrete autocovariance function $c_d(t)$ from which the sampled noise $n$ is defined. The sampled noise, $n$, is a random vector with a multivariate normal distribution,

$$n \sim normal(0, \Sigma_n) \tag{7.3}$$

where $\Sigma_n(i,j) = c_d(iT - jT)$, $n$ is a random vector with $m$ components, $mT$ is the total duration of the simulation, and $T$ is the sampling period. Entry $n(i)$ in random vector $n$ represents a noise sample at time $iT$. We generate realizations of noise sample $n(i)$ from a random number generator and inject them into the system at time $iT$. Correlated samples specified in $\Sigma_n$ can be generated using filtering of uncorrelated random samples.

Despite the simplicity of the Monte Carlo approach, it has problems with computing low error probabilities and machine dependency. In previous chapters Monte Carlo integration suffices to estimate roughly parameters of interest because the probabilities of the events are not low; therefore, large variance of estimation is tolerated. However, for a system with low error probability $p_e$, the number of system runs required for a confident statistical error estimate is proportional to $\frac{1}{p_e^2}$. For

example, to simulate a receiver with bit error rate of $10^{-8}$, we need approximately $10^{16}$ samples. Furthermore, pseudo-random number generators on computers often do not generate such a large sequence of independent random numbers, but will re-use old random numbers instead. In such instances, the extra simulations do not yield a more accurate statistical estimate.

To compute the probability of rare events, special variance reduction techniques for Monte Carlo simulations such as importance sampling can be used. In general, Monte Carlo simulations can be expressed as the following integral,

$$\int_{\Omega} g(x)f(x)dx \tag{7.4}$$

where $g(x)$ is the parameter function, $f(x)$ is the probability density function, and $\Omega$ is the sample space. In simple Monte Carlo, the integral is computed by generating samples of $x$ from density function $f(\cdot)$ and summing $g(x)$. An **important sample**, $x$, provides useful information such as non-zero $g(x)$. Importance sampling[21] forces generation of important samples by using an artificial density function instead of $f(x)$. We can rewrite (7.4) as

$$\int_{\Omega} g(x)\left[\frac{f(x)}{h(x)}\right]h(x)dx \tag{7.5}$$

where $h(x)$ is an artificial density function designed to increase the number of important samples, and $\frac{f(x)}{h(x)}$ is the weight of a sample.

Importance sampling is viable provided $h(x)$ is given, but no general algorithm has been proposed for choosing $h(x)$. There are typically many different noise sources, so the number of possible $h(x)$ is large. As a result, automatic selection of a suitable $h(x)$ out of many possibilities is difficult.

To circumvent the difficulty of choosing $h(x)$, Kahn introduced "splitting" techniques[24] as another form of importance sampling. In this technique, he splits an important sample into many neighboring samples with appropriate weights, thus increasing the number of important samples.

## 7.4 Behavioral model of noise

The key disadvantage of the Monte Carlo technique is the representation of noise by computer generated samples of noise because many noise samples are

required for good accuracy. In contrast, we represent noise by a **random signal**. We sample the random signal $n$ at time $iT$ to obtain a noise sample $n(i)$ in (7.3). Each noise sample, $X$, is a **random variable** described by its distribution function $f_X$, or alternatively by all of its **moments**, where the $k^{th}$ moment $x_k$ is defined as

$$x_k = E[X^k] = \int_{-\infty}^{\infty} u^k f_X(u) du \tag{7.6}$$

Although not all random variables can be described by their moments, we assume that the random variables we consider can be completely described by their moments.

In our noise model, we will approximate a random variable by its first $k+1$ moments from $x_0$ to $x_k$, where $k$ is an integer selected by the user based on accuracy requirements. For example, we represent a Gaussian random variable $X$ with zero mean ($E[X] = 0$) and variance $\sigma^2 = (E[X^2] - (E[X])^2)$ using the first three moments as

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ \sigma^2 \\ 0 \end{bmatrix}$$

In mixed-mode sampled-data systems, signals such as reference, supply voltage, and ground are deterministic. We represent a deterministic signal $X = a$ by its first three moments also as

$$X = \begin{bmatrix} a \\ a^2 \\ a^3 \end{bmatrix}$$

since $E[X] = a, E[X^2] = a^2, E[X^3] = a^3$.

The moments are parameters to characterize a distribution function. If all moments are used, then the distribution function is completely characterized. The advantage of using a finite number of lower order moments is computational efficiency. Instead of manipulating a distribution function, we manipulate a set of numbers representing the moments. Furthermore, from the Central Limit Theorem, random variables resulting from many noise sources should converge to a Gaussian distribution which can be characterized by the first two moments, $x_1$ and $x_2$. As a result, low order moments are often the principal components of a random variable in practical circuits. The disadvantage of using a finite number of moments is that effects of higher order moments are not considered.

Although the moments describe the distribution of a random variable, they do not provide information about the correlation between each random variable. In general, noise in a system are correlated because the noise sources themselves may be correlated or the system may contain architectures with **reconvergent fan-out**. For example, when the signal path diverges as shown in Figure 7.3, the resulting signals A and B are correlated since they originate from a single noise source. In the presense

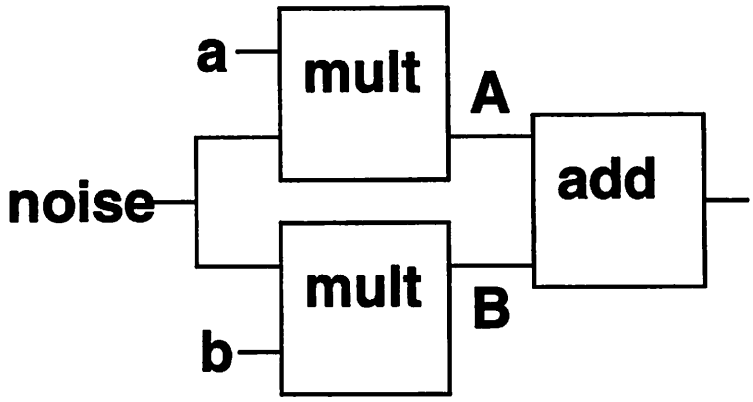


Figure 7.3: Reconvergent fan-out causes correlated noise

of correlated noise, **joint moments** as well as moments are necessary to describe the noise statistics completely. As a first step, we will use only moments and focus on systems with noise that are *independent* for the following reasons.

- Noise sources from physically different components should be independent of each other.
- In most sampled-data systems, white noise dominates non-white noise. Thus, successive samples in time of a noise source are approximately independent.
- We focus on systems such as converters and finite impulse response filters that lack reconvergent fan-out.

## 7.5 Behavioral models of components

As we generalize analog signals from a deterministic representation to a stochastic representation in Section 7.4, we must generalize the system components

to handle such random signals as well. Components in a mixed-mode sampled-data system fall into three classes; *linear*, *mildly nonlinear*, and *strongly nonlinear*.

### 7.5.1 Linear components

The output of a **linear component** is a linear combination of the inputs. The output moments can be computed using binomial expansion and the independence assumption. For example, an adder sums two random variables $X$ and $Y$ to produce an output $Z$. We find the moments of $Z$ from the moments of $X$ and $Y$ using

$$z_k = E[Z^k] = E[(X+Y)^k] = \sum_{j=0}^{k} \binom{k}{j} E\left[X^{k-j}Y^j\right] \tag{7.7}$$

using the binomial expansion of $(X+Y)^k$. From the independence of $X$ and $Y$ assumed in Section 7.4, we then have

$$z_k = \sum_{j=0}^{k} \binom{k}{j} EX^{k-j}EY^j = \sum_{j=0}^{k} \binom{k}{j} x_{k-j}y_j \tag{7.8}$$

### 7.5.2 Mildly nonlinear components

The output of a **mildly nonlinear** component is a polynomial function of the inputs. The output moments can be computed from input moments using binomial expansion and the independence assumption. For example, a multiplier multiplies two inputs $X$ and $Y$ to produce $Z$.

$$Z = XY \tag{7.9}$$

We find the moments of $Z$ from the moments of $X$ and $Y$ using

$$z_k = E[(XY)^k] = E[X^kY^k] = E[X^k]E[Y^k] = x_k y_k \tag{7.10}$$

### 7.5.3 Strongly nonlinear components

For our purposes, the output of a **strongly nonlinear** component is a piecewise polynomial function (with $m$ pieces) of the inputs. The domain is partitioned into $m$ disjoint pieces, where $r_i$ is the domain for the $i^{th}$ polynomial function. The **scenario**

for $X \in r_i$ corresponds to the input, $X$, falling in the domain of the $i^{th}$ polynomial function. For this scenario, the output, $Z$, is given by the conditional distribution

$$Z = Z|X \in r_i \tag{7.11}$$

and $X \in r_i$ implies that $X$ should be replaced by

$$X = X|X \in r_i \tag{7.12}$$

The expected value of the output due to all possible scenario is the sum of (7.11) over all possible scenarios,

$$Z = \sum_{i=1}^{m} (Z|X \in r_i) Pr(X \in r_i) \tag{7.13}$$

In Section 7.6, we propose to compute the sum using a new simulation algorithm.

In mixed-mode sampled-data systems, the comparator is a strongly nonlinear component which has a piecewise constant transfer function as shown in Figure 7.4. The comparator has as inputs a random variable $X$ and a threshold value $t$, and

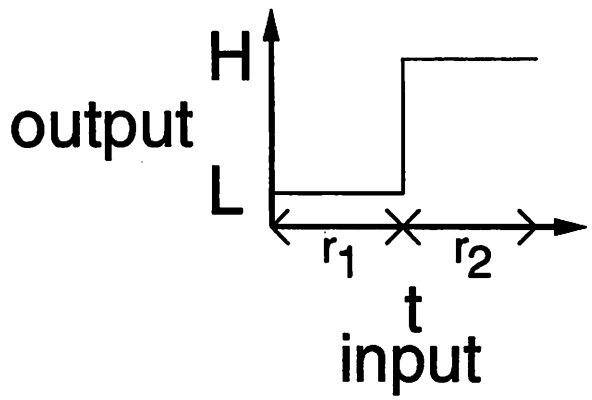


Figure 7.4: Comparator transfer function

produces the probability $p_H$ for the scenario of output high (H), the probability $p_L$ for the scenario of output low (L), the conditional distribution of $X$ given the discrete output $Y$ being low ($f_{X|Y}(x|L)$ in (7.12)), and the conditional distribution of $X$ given $Y$ being high ($f_{X|Y}(x|H)$ in (7.12)). Mathematically, the probability for output above $t$ is defined as

$$p_H = \int_t^{\infty} f_X(u)du \tag{7.14}$$

where $f_X$ is constructed from the given moments $x_n$ (Section 7.5.4). Conversely, the probability for output low is defined as

$$p_L = \int_{-\infty}^{t} f_X(u)du = 1 - p_H \tag{7.15}$$

In (3.23), it is shown that the conditional distribution of $X$ given that $Y = y$ is

$$f_{X|Y}(x|y) = \frac{P\{Y = y|X = x\}}{P\{Y = y\}} f_X(x) \tag{7.16}$$

Also, we have $P\{Y = H\} = p_H$ and

$$P\{Y = H|X = x\} = \begin{cases} 1 & x \geq t \\ 0 & x < t \end{cases}$$

Applying (7.16) in our comparator, we find the conditional distribution of $X$ given $Y = H$ is

$$f_{X|Y}(x|H) = \begin{cases} \frac{f_X(x)}{p_H} & x \geq t \\ 0 & x < t \end{cases} \tag{7.17}$$

Conversely, the conditional distribution of $X$ given $Y = L$ is

$$f_{X|Y}(x|L) = \begin{cases} 0 & x \geq t \\ \frac{f_X(x)}{p_L} & x < t \end{cases} \tag{7.18}$$

Once the conditional distributions are determined, they are converted to a set of $k + 1$ moments (Section 7.5.4).

### 7.5.4 Algorithms for comparators

In this section, we will show the algorithms to compute from input moments of $X$ and the comparator threshold $t$ the probabilities of the comparator output Y=H and Y=L, as well as the moments for the conditional distributions $f_{X|Y}(x|H)$ and $f_{X|Y}(x|L)$. We will show the cases for $k = 2, 3$, where $k$ is the highest moment of interest.

#### Highest moment k=2

When $k = 2$, the random variable $X$ is approximated by a Gaussian random variable. A Gaussian distribution is chosen because random variables resulting from

many noise sources should converge to a Gaussian distribution which can be characterized by the first two moments, $x_1$ and $x_2$. The mean, $\mu$, is given by $x_1$, and the variance $\sigma^2$ is given by $x_2 - x_1^2$. With these parameters, the distribution of $X$ is

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad -\infty < x < \infty \tag{7.19}$$

The probability for $Y = H$ is given by integrating the distribution from the trigger point $t$ to infinity,

$$p_H = \int_t^\infty f_X(x)dx = 1 - Q\left(\frac{t-\mu}{\sigma}\right) \tag{7.20}$$

where $Q(\cdot)$ is a standard mathematical function defined as

$$Q(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du \tag{7.21}$$

Letting $A \sim f_{X|Y}(x|H)$, and using (7.17), we have

$$f_A(x) = \begin{cases} \frac{1}{\sqrt{2\pi}\sigma p_H} e^{-\frac{(x-\mu)^2}{2\sigma^2}} & x \geq t \\ 0 & x < t \end{cases} \tag{7.22}$$

The moments of $A$ are then given by

$$a_1 = \int_{-\infty}^{\infty} x f_A(x)dx = \frac{\sigma}{\sqrt{2\pi}p_H} e^{-\frac{(t-\mu)^2}{2\sigma^2}} + \mu \tag{7.23}$$

$$a_2 = \int_{-\infty}^{\infty} x^2 f_A(x)dx = \frac{\sigma}{\sqrt{2\pi}p_H}(t-\mu)e^{-\frac{(t-\mu)^2}{2\sigma^2}} + \sigma^2 + 2\mu a_1 + \mu^2 \tag{7.24}$$

The probability for $Y = L$ is then given by $p_L = 1 - p_H$. Letting $B \sim f_{X|Y}(x|L)$, the moments of $B$ are given by

$$b_1 = (x_1 - a_1 p_H)/p_L \tag{7.25}$$

$$b_2 = (x_2 - a_2 p_H)/p_L \tag{7.26}$$

**Highest moment k=3**

When $k = 3$, the random variable, $X$, is approximated by distribution[55] created by joining together two halves of two different Gaussian random variables defined as

$$f_X(x) = \begin{cases} \frac{2}{\sqrt{2\pi}(\sigma_1+\sigma_2)} e^{-\frac{(x-a)^2}{2\sigma_1^2}} & x \geq a \\ \frac{2}{\sqrt{2\pi}(\sigma_1+\sigma_2)} e^{-\frac{(x-a)^2}{2\sigma_2^2}} & x < a \end{cases} \tag{7.27}$$

where its moments are given by

$$x_1 = (\sqrt{\frac{2}{\pi}}\sigma_1 + a)\frac{\sigma_1}{\sigma_1+\sigma_2} + (a - \sqrt{\frac{2}{\pi}}\sigma_2)\frac{\sigma_2}{\sigma_1+\sigma_2} \tag{7.28}$$

$$x_2 = (\sigma_1^2 + 2\sqrt{\frac{2}{\pi}}\sigma_1 a + a^2)\frac{\sigma_1}{\sigma_1+\sigma_2} + (\sigma_2^2 - 2\sqrt{\frac{2}{\pi}}\sigma_2 a + a^2)\frac{\sigma_2}{\sigma_1+\sigma_2} \tag{7.29}$$

$$x_3 = (2\sqrt{\frac{2}{\pi}}\sigma_1^3 + 3\sigma_1^2 a + 3\sqrt{\frac{2}{\pi}}\sigma_1 a^2 + a^3)\frac{\sigma_1}{\sigma_1+\sigma_2} +$$
$$(-2\sqrt{\frac{2}{\pi}}\sigma_2^3 + 3\sigma_2^2 a - 3\sqrt{\frac{2}{\pi}}\sigma_2 a^2 + a^3)\frac{\sigma_2}{\sigma_1+\sigma_2} \tag{7.30}$$

This distribution is chosen because it degenerates into a Gaussian distribution when $\sigma_1 = \sigma_2$. A closed form solution for $\sigma_1, \sigma_2, a$ in terms of $x_1, x_2, x_3$ is intractable. A numerical solver is difficult to implement due to the need for good initial guesses. As a result, parameters $\sigma_1, \sigma_2, a$ are computed from $x_1, x_2, x_3$ using the downhill simplex optimization method in multidimensions[43] to best match the moments.

The cost function being used is $c = \sum_{n=1}^{k}(x_k - x_k')^2$, where $x_k$ are the moments given, and $x_k'$ are the resulting moments given by (7.28) to (7.30).

Let $A \sim f_{X|Y}(x|H)$, then using (7.17) we have

$$a_1 = \sqrt{\frac{2}{\pi}}\frac{\sigma_1}{\sigma_1+\sigma_2}\frac{\sigma_1}{p_H}e^{-\frac{(t-a)^2}{2\sigma_1^2}} + a \tag{7.31}$$

$$a_2 = \sqrt{\frac{2}{\pi}}\frac{\sigma_1}{\sigma_1+\sigma_2}\frac{\sigma_1}{p_H}(t-a)e^{-\frac{(t-a)^2}{2\sigma_1^2}} + \sigma_1^2 + 2a_1 a + a^2 \tag{7.32}$$

$$a_3 = 2\sqrt{\frac{2}{\pi}}\frac{\sigma_1}{\sigma_1+\sigma_2}\frac{\sigma_1^3}{p_H}(1 + \frac{(t-a)^2}{2\sigma_1^2})e^{-\frac{(t-a)^2}{2\sigma_1^2}} + 3a_2 a + 3a_1 a^2 + a^3 \tag{7.33}$$

## 7.6 System simulation algorithm

After describing the random signal representation, we focus in this section on the algorithm used to simulate the noise in sampled-data systems. The algorithm is event-driven as described in detail below, yet it must handle noise represented as random signals in mixed-mode, sampled-data systems. In such systems, the strongly nonlinear components are special. For example, the comparator is a special interface since its input is analog while its output is digital (e.g. H or L). Due to noise, the

analog signals are assumed random; therefore, both comparator output states are possible. The technique used in our algorithm is to compute the probability of each possible outcome, and continue the simulation recursively for each possible **scenarios**. Schematically, the simulation follows a tree of scenarios based on comparator outputs as shown in Figure 7.5. To compute the probability of a particular scenario occurring, we take the product of the probabilities on the unique path from the tree root to the scenario.

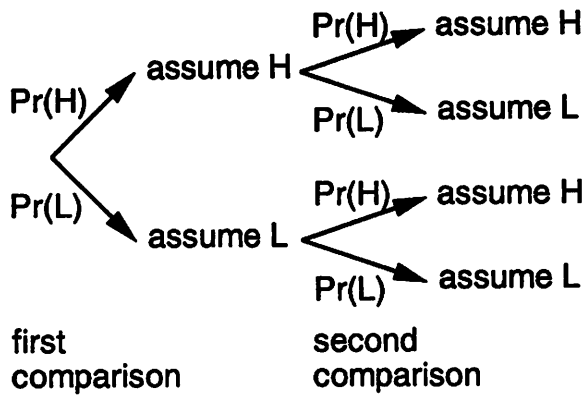


Figure 7.5: Scenario tree

The simulation algorithm is as follows.

- (a) Extract noise parameters from circuits using (1) and (2). Construct moments of distributions for noise sources.
- (b) Topologically sort network to find the evaluation order of components. The procedure always succeeds because all feedback paths are broken by at least one delay in sampled-data systems.
- (c) Evaluate each component in order, return if end of list.
- (d) When evaluating a comparator, check distribution at comparator input X, find probabilities for output Y = H and L, execute recursively (e) and (f)
- (e) Assume the scenario with output Y=H, find conditional distribution of X | Y=H, replace X with X | Y=H, continue simulation in (c)
- (f) Assume the scenario with output Y=L, find conditional distribution of X | Y=L, replace X with X | Y=L, continue simulation in (c)

## 7.7 Experimental results

### 7.7.1 Cyclic A/D

A 12 bit cyclic A/D (schematic in Figure 7.1 and model for noise calculations in Figure 7.6) has been modeled in the C++ language for both Monte Carlo simulations and direct noise calculations. The gain parameter of a sample-and-hold can be computed from a circuit level implementation using SPICE transient analysis given the fixed system clock period. In this experiment, we assumed the gain to be one for $1x$ S/H and two for $2x$ S/H. The comparator threshold can be computed using SPICE also, but we assumed it to be $1V$. Input voltage is half of the full range input. The effective noise of a sample-and-hold can be computed using (7.2) and (7.3). However, for this experiment we assume that the effective noise is given and is white with standard deviation of $0.365mV$, which may be considered high. However, a high noise value is used to prevent excessive CPU time in Monte Carlo simulations, while a lower noise value would not affect our method.

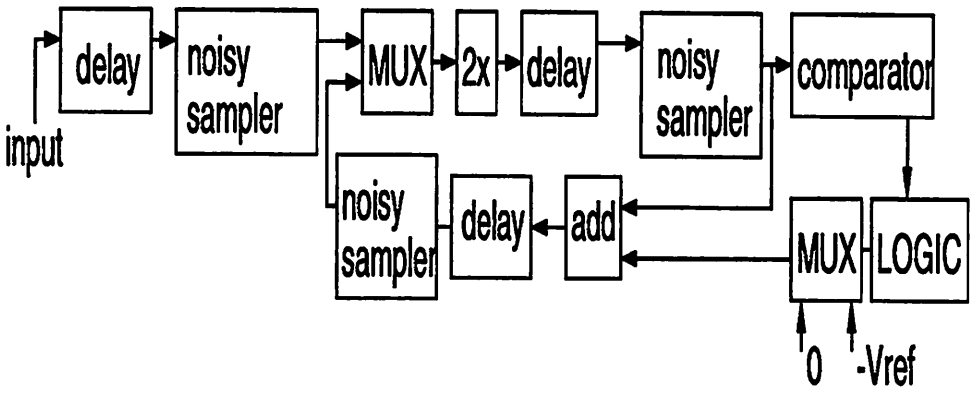


Figure 7.6: Noise model for cyclic converter

Figure 7.7 compares the predicted output code distribution from a reference Monte Carlo simulation ($10^7$ trials) with the output code distributions from direct noise calculations ($k = 2$) and ($k = 3$), where $k$ is the highest moment of interest. Notice that the results are comparable, but the CPU times are much less (9620s vs. 0.1s and 3.7s). Next, in Figure 7.8 we compare the Monte Carlo method with the direct method for accuracy as a function of CPU time. The number of trials in Monte Carlo simulations is varied from one hundred to seven hundred thousand. Assuming the

reference simulation gives the true solution, the absolute errors in the probabilities are computed for all output codes. The average of the estimated absolute error is then plotted against CPU time. The error curve for the Monte Carlo method suggests that its accuracy is inversely proportional to CPU time. The error curve for the direct method shows that error decreases with higher order approximation. Besides, the direct method has smaller error than the Monte Carlo method for comparable CPU times.

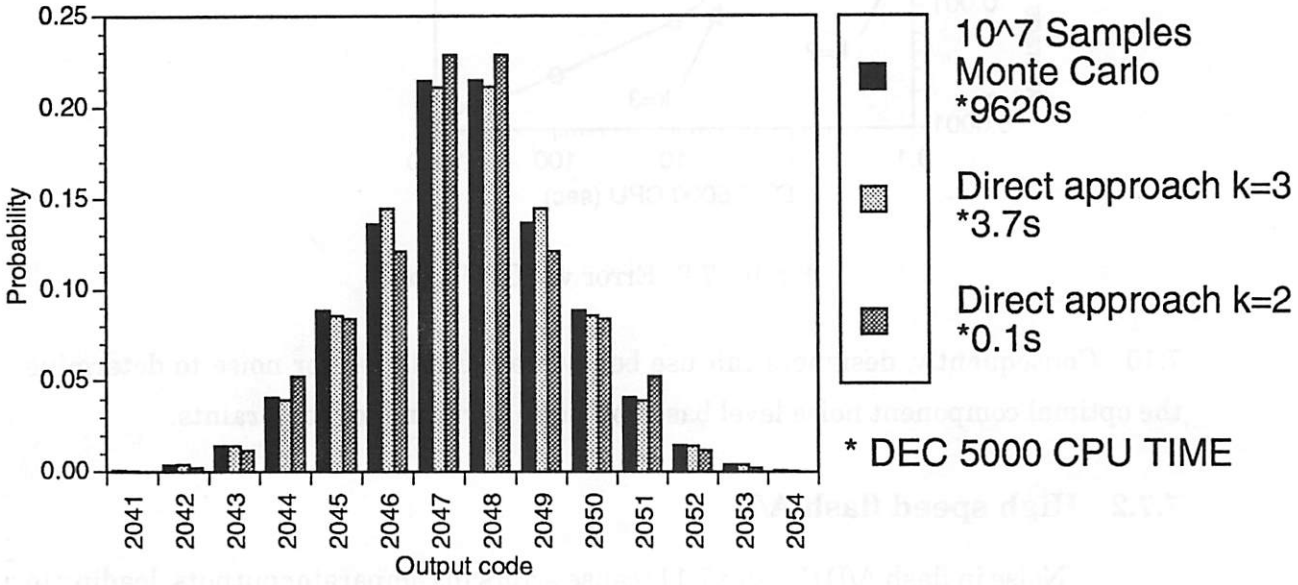


Figure 7.7: Twelve-bit cyclic A/D code distribution

To investigate noise effects for different inputs, we computed noise effects for a voltage range of several LSB's of input voltage centered around half the reference. First, we divide the input range into 30 uniformly spaced inputs. Then, we compute the output code distribution for each input. The result, obtained using 1.3 DEC 5000 CPU seconds, is the joint probability density function of the input and output shown in Figure 7.9.

Notice that the probabilities for wrong codes are as high as that of the correct code, rendering the A/D unusable. In top-down design methodologies, designers can try various noise parameters to determine the proper component noise level from system level constraints. For example, if the noise level in the sample-and-hold is reduce by ten times, then the A/D noise performance is much better as shown in Figure

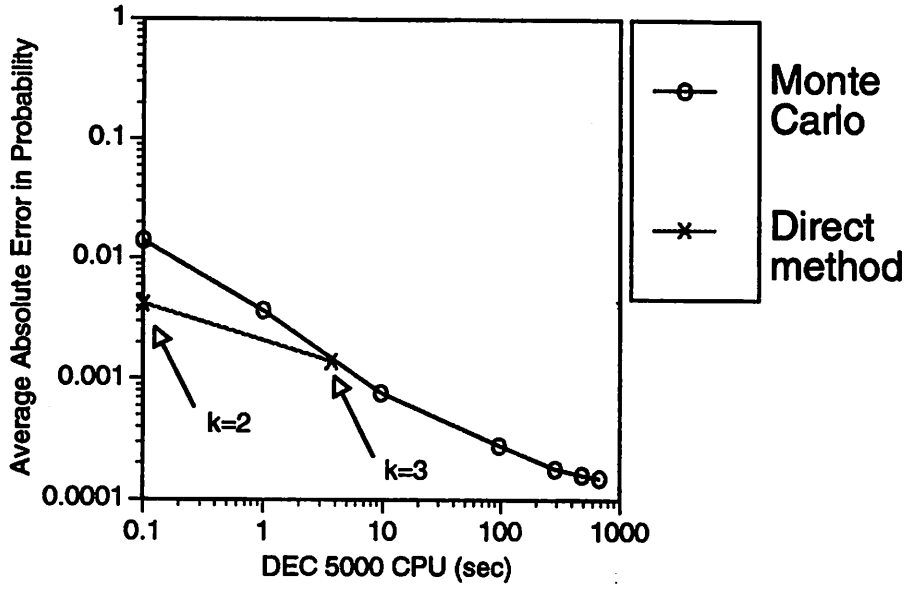


Figure 7.8: Error vs. CPU time

7.10. Consequently, designers can use behavioral simulation for noise to determine the optimal component noise level based on noise performance constraints.

### 7.7.2 High speed flash A/D

Noise in flash A/D (Figure 7.11) cause errors in comparator outputs, leading to errors in digital decoding. If a simple digital decoder (Figure 7.12) is used, the output code can be very different from the correct code, in which case the error is called a **sparkle**[16]. Because the output error probabilities are usually very small, ($6.4 \cdot 10^{-4}$ in Figure 7.13), many simulations are needed in the Monte Carlo method to predict sparkles.

In contrast, using the direct noise computation method, we have modeled two flash A/D's, one with a simple thermometer-to-binary ROM decoder and one with a more advanced decoder(Figure 7.12). Effective comparator noise can be estimated from [37]. In this experiment, for convenience we set the standard deviation of the comparator noise to a quarter of the smallest input voltage step size. Input voltage is half of the full range input. The output code distribution, computed using direct method (k=2), is shown in Figure 7.13.

Because the probability of error for sparkles is small, computing sparkles

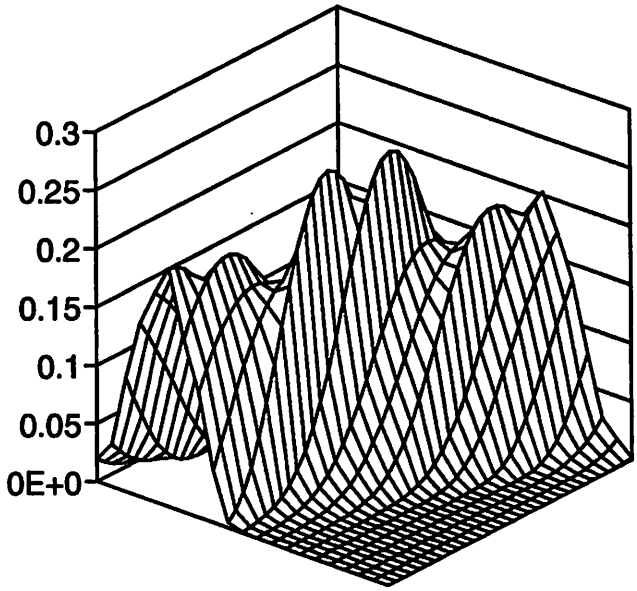


Figure 7.9: Joint probability density function for high component noise

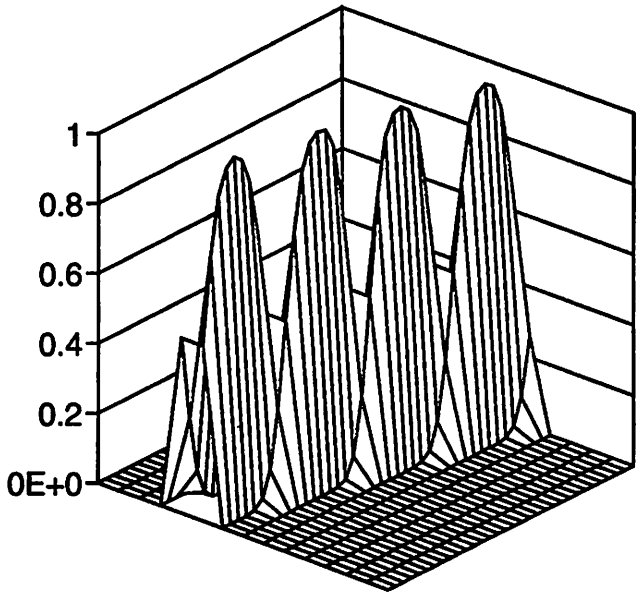


Figure 7.10: Joint probability density function for low component noise

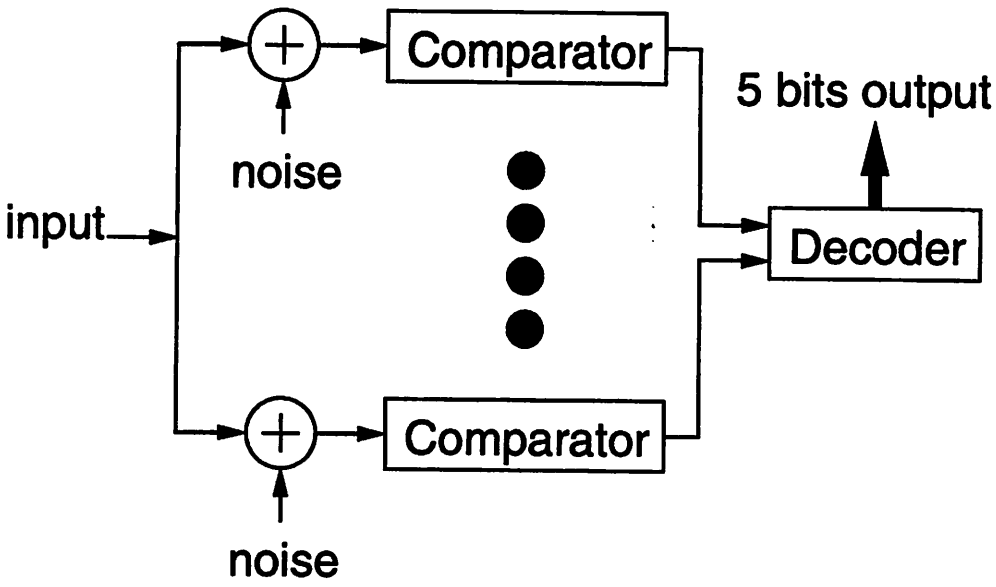


Figure 7.11: Flash converter architecture

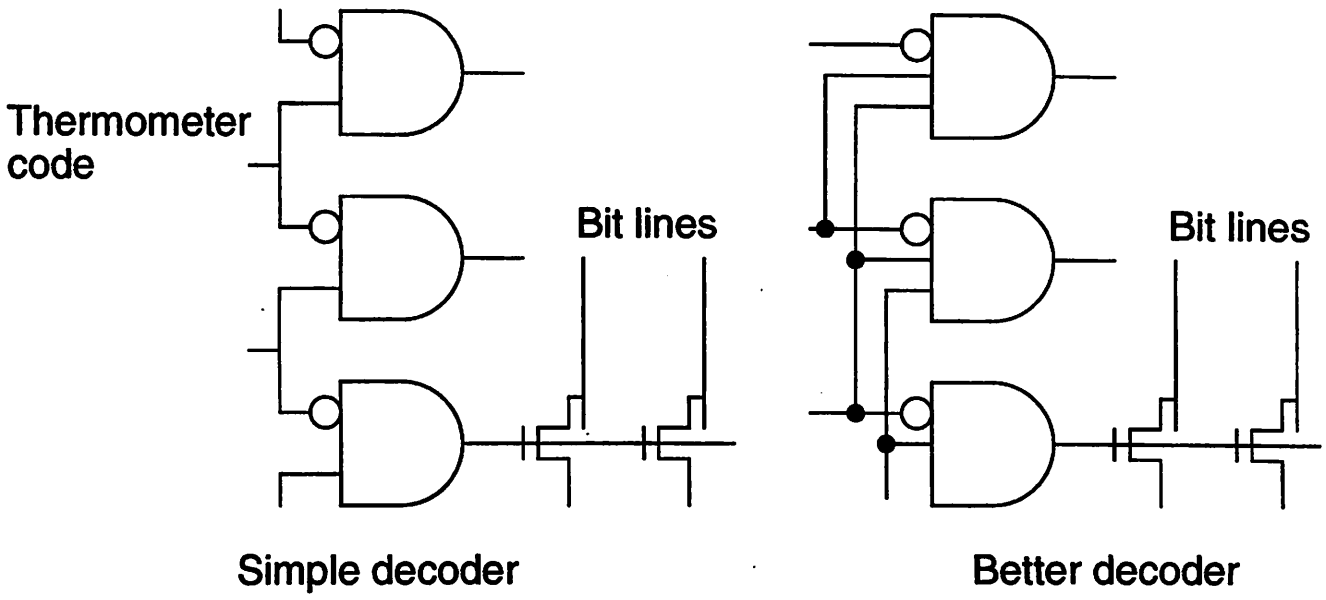


Figure 7.12: Decoders for flash converter

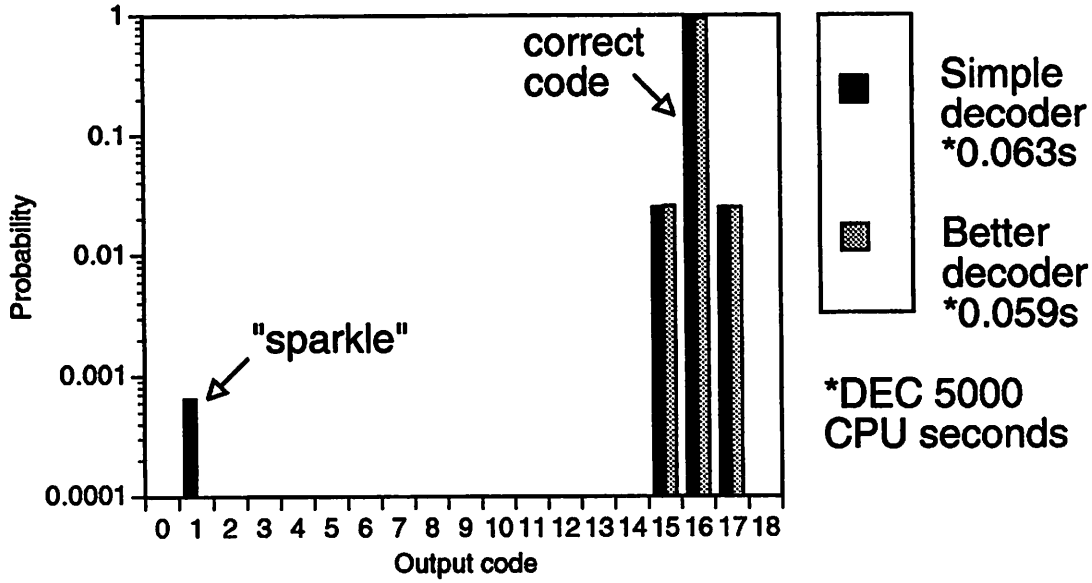


Figure 7.13: Five-bit flash A/D code distribution

with Monte Carlo simulations would take a large amount of CPU time. To illustrate, we show in Figure 7.14 the percent error in computing sparkles as a function of CPU time. Notice that the accuracy of the Monte Carlo simulation technique is inversely proportional to the CPU time as expected. To get an accuracy of 1%, 10000 DEC 5000 CPU seconds are needed. On the other hand, to get the same accuracy using the proposed strategy with $k = 2$, where $k$ is the user selected highest moment of interest, 0.1 DEC 5000 CPU seconds are needed. Therefore, behavioral simulation is about 5 orders of magnitude faster.

Furthermore, using higher order moments increases the CPU time needed as shown in Figure 7.14 for $k = 3$, yet do not increase the accuracy much. Therefore, in this case, lower order moments are sufficient for a good estimate of noise effects.

To investigate noise effects for different inputs, we computed noise effects for the full input range. The result, obtained using 1.9 DEC 5000 CPU seconds, is the joint probability density function of the input and output shown in Figure 7.15. Notice that the wall in the center corresponds to the high probability of getting the correct codes, while the individual small columns on the right corresponds to sparkles.

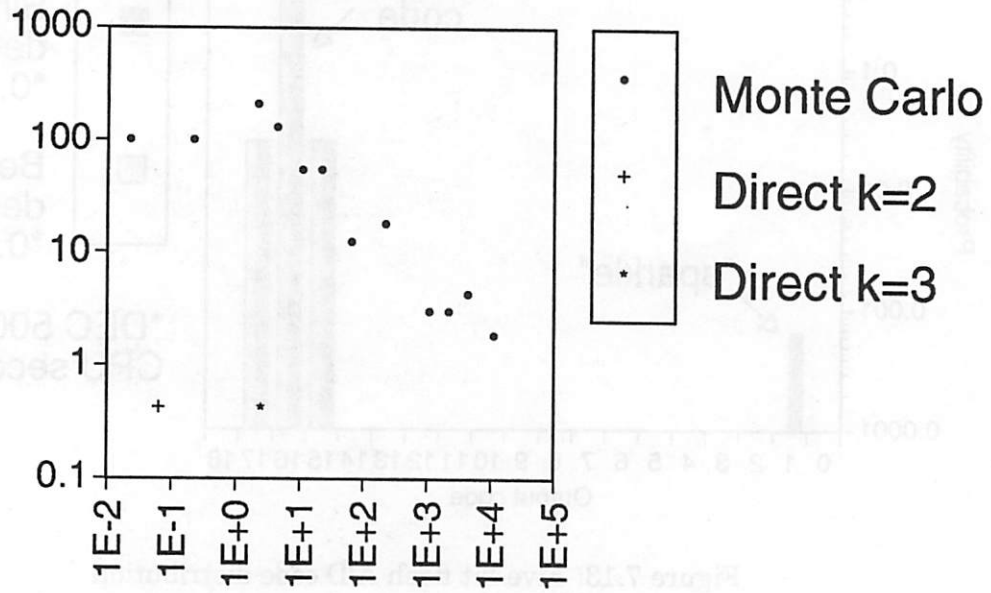


Figure 7.14: Accuracy vs. DEC 5000 CPU time for Monte Carlo and direct methods

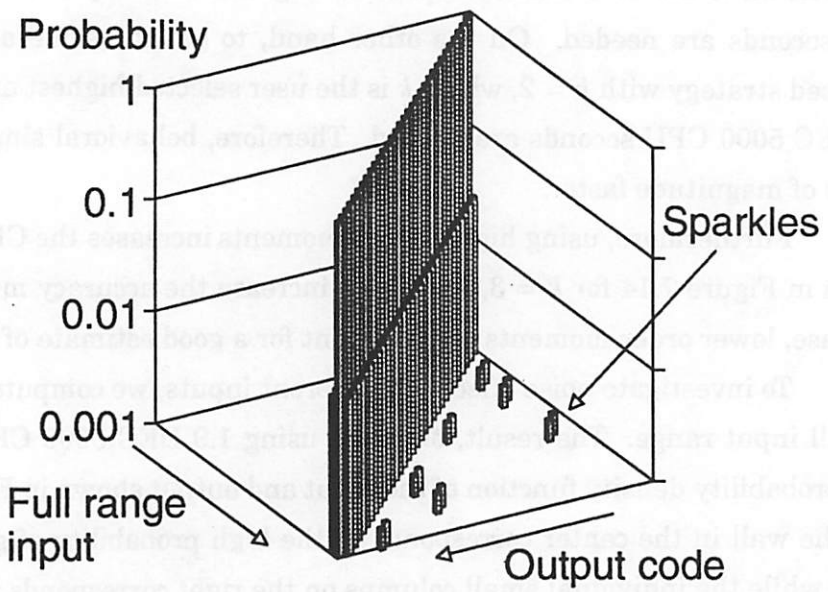


Figure 7.15: Joint probability density function for flash converter

## 7.8 Conclusion

We have presented a "direct" noise analysis approach for mixed mode systems, and compared our approach with the traditional Monte Carlo approach. The approach is approximate and computes noise effects by performing arithmetic on a finite number of moments of distribution functions that characterize electronic noise. One key advantage of this approach is its ability to compute low error probabilities.

From experimental results, we have shown that our approach is several orders of magnitude more efficient than the Monte Carlo approach in computing sparkles (low probability code errors) in data converters. Furthermore, the estimation errors due to low order moment approximation seem to be small, since the simulation results for a flash converter using a second order approximation do not differ significantly from results obtained using a third order approximation. As a result, low order moments, such as second order, are sufficient for a good estimate of noise effects for flash converters.

# Chapter 8

# Conclusion and Future Directions

## 8.1 Conclusion

System design and verification using traditional simulators such as SPICE is often impossible due to long simulation times. To circumvent the design and verification problems associated with traditional simulators, we proposed a **top-down, constraint-driven** approach[10, 11] to designing complex mixed-signal circuits, where abstraction, successive design refinement, and constraint propagations are key. To support the proposed design methodology, we developed system simulation algorithms and behavioral models for different types of analog systems and components.

In analog systems, the nominal circuit functions are usually very simple, and system malfunctions are most often due to second order effects caused by noise and process variations. As a result, system constraints are usually specified in terms of the maximum amount of second order effects allowed such as signal-to-noise ratio and total harmonic distortion. In turn, component constraints are usually specified in terms of basic statistical effects such as random offsets and mismatches. Therefore, *behavioral models at all levels must capture second order effects for constraint translation in top-down design.*

Traditional circuit simulators developed previously are inadequate for large analog circuit design due to the long simulation time. Using the circuit simulation and

macromodeling approaches, it is very difficult to simulate frequency domain effects, noise effects, or effects due to process variations because all models are deterministic. As a result, we proposed a new strategy for behavioral simulation and modeling for the design and verification of systems in the presence of noise effects and effects due to process variations.

A major result of this thesis is the development of a behavioral representation for Nyquist data converters. The representation captures the behavior of a memoryless Nyquist data converter, including statistical variations. The variations are classified into noise and process variations according to how these non-idealities affect the converter behavior. To describe noise effects, a joint probability density function is used. To describe process variations effects on the converter transfer function, a Gaussian model is used.

We applied our behavioral simulation strategy to verification of analog systems. In our approaches, we take advantage of the hierarchical decomposition of the system into components. In the first approach, we extract each component individually, verify it under ideal bias conditions, and fit parameters for its behavioral model. Then, we simulate the system at the *behavioral* level using behavioral models only. Behavioral simulation results compare well with Monte Carlo SPICE simulations. The approach is exact and works well for architectures with negligible parasitic loading effects.

To verify the system in the presence of parasitic loading between components, we proposed an approximate strategy. In this strategy, we extract each component individually, verify it under ideal bias conditions, fit parameters for its behavioral model, linearize the components at the operating point, substitute the linearized component in the interconnect network, find the changes in the bias conditions, and estimate the performance deviation due to bias changes using a first order Taylor approximation. From experimental results, we validated the verification approach for a 10 bit interpolative D/A using the proposed converter behavioral model.

Data converters are commodity products, yet their testing is very expensive. We proposed a strategy for testing all DC performance of Nyquist data converters including offset error, full scale gain error, integral nonlinearity, and differential nonlinearity. In contrast to previous testing strategies based on linear models that require accurate measurements of circuit performance in the presence of measurement noise,

our strategy uses a simpler measurement to verify that a circuit performance parameter falls within a certain range in the presence of measurement noise. We proposed an optimal test selection strategy for nonlinearity errors in data converters that uses a simple heuristic ordering based on QR factorization and linear programming. The test selection strategy is exact, but the yield estimate is approximate. Using the proposed strategy and behavioral modeling of the device under test, we evaluated tradeoffs between test set size, detection thresholds, measurement noise, chip performance, and estimated yield.

Finally, we focused on noise modeling and simulation for mixed-mode sampled-data systems. We presented a "direct" noise analysis approach for mixed-mode systems, and compared our approach with the traditional Monte Carlo approach. The approach is approximate and computes noise effects by performing arithmetic on a finite number of moments of distribution functions that characterize electronic noise. One key advantage of this approach is its ability to compute low error probabilities. From experimental results, we have shown that our approach is several orders of magnitude more efficient than the Monte Carlo approach in estimating the probabilities of sparkles (low probability code errors) in data converters. Furthermore, the estimation errors due to lower order moment approximation is small, since the simulation results for a flash converter using a second order approximation do not differ significantly from results obtained using a third order approximation. We showed from experimental results that low order moments, such as second order, are sufficient for a good estimate of noise effects for a flash converter.

## 8.2 Future Directions

Although the user can choose between a Gaussian or non-Gaussian data converter model, formulae are available to estimate data converter performance only for the Gaussian model. For completeness, formulae for the non-Gaussian model should be developed in the future.

Also, our analog system verification strategy works for DC bias changes due to parasitic and loading resistances because only DC sensitivity calculations for linear networks have been implemented using adjoint techniques. As a result, transient effects due to parasitic loading capacitance cannot be handled. In the future, the

verification technique can be extended to handle loading effects in transient simulation when transient sensitivity analysis is available.

Currently, our optimal test selection strategy for nonlinearity errors in data converters uses a simple heuristic ordering based on QR factorization and bound checking using linear programming. In the future, we need to determine the complexity of an exact method, and propose better heuristic algorithms if the exact method is analytically intractable.

A useful contribution is to extend our proposed noise simulation method to handle higher order moments for better accuracy. In particular, we need to implement comparator models that handle fourth and fifth order moments. Also, to handle more general sampled-data systems with correlated noise, we need to investigate extension of the technique to handle joint moments.

With the experience gained from behavioral modeling presented in this thesis, we can derive requirements for an Analog Hardware Description Language (AHDL) and a comprehensive software environment for the behavioral modeling and simulation of mixed-mode systems. For example, from the work in converter modeling, we understand that statistical models are essential to describe data converters. Parasitic loading effects are essential in system verification, so models should satisfy some form of KCL and KVL. From the work in noise simulation, noise is better represented analytically using distributions.

We believe it is essential to build a comprehensive library of components for top-down design and verification of mixed-mode systems. In the future, we plan to derive more requirements by investigating behavioral models for different types of analog components. For example, we need models for components in phase-locked loops, filters, bandgaps, sample-and-holds, etc.

We also plan to implement a simulator with a hardware description language for the simulation of data converters in the presence of process variations. The user will be able to simulate converter performance such as $\pm 3\sigma$ offset error, gain error, integral nonlinearity, and differential nonlinearity. When the design is completed, tests vectors can be generated automatically using the algorithm proposed in Chapter 6.

Instead of using the C++ language to describe converter behavior (Section 5.3.3), the user will use an *analog high level hardware description language* to specify

the converter architecture with a network of library components such as sample-and-holds, resistor strings, comparators, etc. Using the language, the user will also be able to specify nominal, process variation, and loading parameters for each of the components. The language description will be compiled into an internal representation for input to a simulator.

After the system is entered with the language, the simulation phase begins. The results presented in this thesis were obtained by a customized simulator where algorithms and models were hard-coded. In the next generation, a general purpose analog behavioral simulator will be built. The architecture of the simulator is shown in Figure 8.1. For different domains of simulation there are different simulation engines. Given the particular problem to be solved, a supervisor will select the right computational engine to be used. The engines we plan to include are difference equation solvers, KCL and KVL solvers (bias point computation), noise algebra solvers, sensitivity analyzer, differential equation solvers, data analyzer, etc. The supervisor applies the different engines to simulate the circuit or analyze data at the appropriate sequence determined by the type of analyses specified by the user.

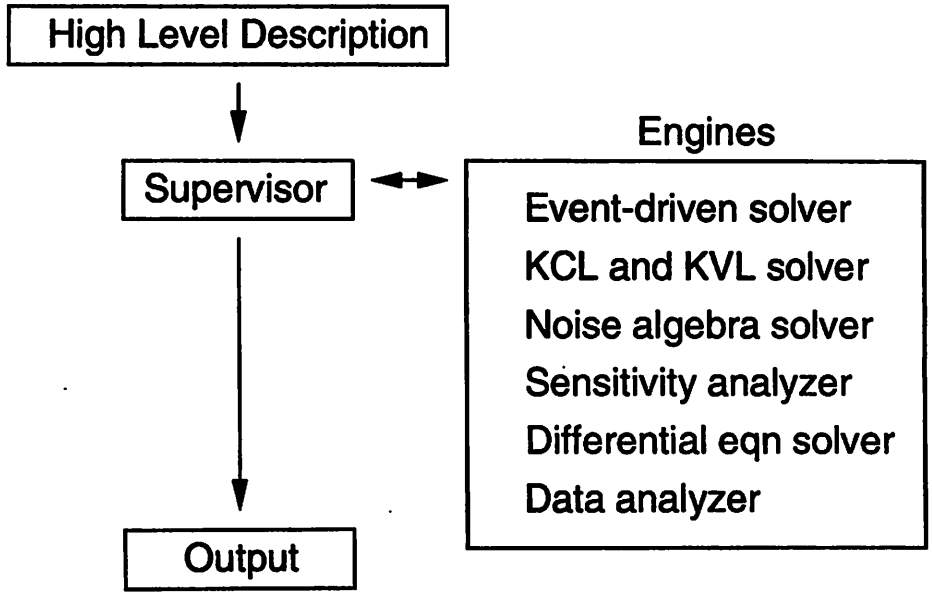


Figure 8.1: General Analog Behavioral Simulator Architecture

For the converter simulation case, the internal representation of a converter architecture is a graph. Each node of the graph is an instantiation of a component

model. Each object-oriented component model encapsulates the component behavior such as nominal behavior, statistical variations, and impedance characteristics of interface pins. Before simulation begins, the KVL and KCL solver solves the graph based on the impedance characteristics of the components to establish the proper "bias" point for each of the components and adjust the component behavioral parameters accordingly. Then, an event-driven simulation engine computes the nominal converter behavior by evaluating the components in an event-driven simulation algorithm. Next, a sensitivity computation engine computes the sensitivity of the nominal behavior with respect to component variations using perturbation and finite differences. For example, the engine perturbs a random variable in a component to the "nominal-plus-$\sigma$" value, then the engine computes the perturbed output. The ratio of the output difference and $\sigma$ gives the sensitivity of the output to the random variable. Finally, a converter data analysis engine takes the nominal and sensitivity information to compute the system parameters of interest such as $\pm 3\sigma$ offset error, gain error, integral nonlinearity, and differential nonlinearity.

# Bibliography

[1] P. E. Allen and P. R. Barton. A silicon compiler for successive approximation A/D and D/A converters. In *Proc. IEEE Custom Integrated Circuits Conference*, pages 552–555, 1986.

[2] B. Antao and F. El-Turky. Automatic analog model generation for behavioral simulation. In *Proc. IEEE Custom Integrated Circuits Conference*, pages 12.2.1–12.2.4, May 1992.

[3] M. Box and N. Draper. Factorial designs, the $|x'x|$ criterion, and some related matters. *Technometrics*, 13, 1971.

[4] G. Boyle, B. Cohn, D. Pederson, and J. Solomon. Macromodeling of integrated circuit operational amplifiers. *IEEE Journal of Solid State Circuits*, SC-9:353–363, 1974.

[5] D. Brillinger. *Time Series Data Analysis and Theory, Expanded Edition*. McGraw-Hill, New York, 1981.

[6] D. R. Brillinger. The identification of polynomial systems by means of higher order spectra. *Journal Sound Vibration*, 12:301–31, 1970.

[7] L. R. Carley and et. al. Acacia: The CMU analog design system. In *Proc. IEEE Custom Integrated Circuits Conference*, 1989.

[8] G. Casinovi and A. Sangiovanni-Vincentelli. A macromodeling algorithm for analog circuits. *IEEE Trans. on CAD*, Vol. 10, No. 2:150–160, 1991.

[9] R. Chadha, C. Visweswariah, and C. F. Chen. $m^3$ - a multi-level mixed-mode mixed D/A simulator. *Proc. IEEE ICCAD*, pages 258–261, November 1988.

[10] H. Chang, A. Sangiovanni-Vincentelli, F. Balarin, E. Charbon, U. Choudhury, G. Jusuf, E. Liu, E. Malavasi, R. Neff, and P. Gray. A top-down, constraint-driven design methodology for analog integrated circuits. In *Proc. IEEE Custom Integrated Circuits Conference*, pages 841–846, May 1992.

[11] H. Chang, A. Sangiovanni-Vincentelli, E. Charbon, U. Choudhury, E. Felt, G. Jusuf, E. Liu, E. Malavasi, and R. Neff. A top-down, constraint-driven design methodology for analog integrated circuits. In *Proc. Workshop on "Advances in Analog Circuit Design", Scheveningen, NL*, pages 301–325, April 1992.

[12] L. O. Chua and Pen-Min Lin. *Computer aided analysis of electronic circuits: algorithms & computational techniques*. Prentice-hall, 1975.

[13] MicroSim Corporation. *Circuit Analysis User's Guide Version 5.0*. MicroSim Corporation, July 1991.

[14] Analog Devices. *Data converter reference manual volume II*. Norwood, MA 02062-9106, 1992.

[15] S. C. Fang, Y. P. Tsividis, and O. Wing. SWITCAP: a switched capacitor network analysis program. *IEEE Circuits Syst. Mag.*, 5(3):4–10, September 1983.

[16] Y. Gendai, Y. Komatsu, S. Hirase, and M. Kawata. An 8b 500MHz ADC. In *Proc. IEEE International Solid-State Circuits Conference*, pages 172–174, February 1991.

[17] I. Getreu. Behavioral modeling of analog blocks using the SABER simulator. *Proc. MWCAS*, pages 977–980, August 1989.

[18] I. Getreu, A. Hadiwidjaja, and J. Brinch. An integrated-circuit comparator macromodel. *IEEE Journal of Solid State Circuits*, SC-11:826–833, 1976.

[19] G. Gielen, E. Liu, A. Sangiovanni-Vincentelli, and P. Gray. Analog behavioral models for simulation and synthesis of mixed-signal systems. In *Proc. EDAC*, March 1992.

[20] P. R. Gray and R. G. Meyer. *Analysis and design of analog integrated circuits*. J. Wiley & Sons, 3rd Edition, 1993.

[21] J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods*. London, Methuen & Co Ltd., 1964.

[22] R. Harjani, R. A. Rutenbar, and L. R. Carley. OASYS: A framework for analog circuit synthesis. *IEEE Trans. on CAD*, pages 1247–1266, 1989.

[23] G. Hemink, B. Meijer, and H. Kerkhoff. Testability analysis of analog systems. *IEEE Trans. on CAD*, June 1990.

[24] H. Kahn. Use of different monte carlo sampling techniques. In *Symposium on Monte Carlo Methods*, pages 146–190. New York, Wiley, 1956.

[25] D. W. Knapp and A. C. Parker. The ADAM design planning engine. *IEEE Trans. on CAD*, pages 829–846, 1991.

[26] H. Y. Koh, C. H. Séquin, and P. R. Gray. Automatic synthesis of operational amplifiers based on analytic circuit models. In *Proc. IEEE ICCAD*, pages 502–505, 1987.

[27] K. S. Kundert. Sparse matrix techniques and their application to circuit simulation. In A. E. Ruehli, editor, *Circuit Analysis, Simulation and Design*, pages 281–324. North-Holland, New York, 1986.

[28] W. Lam, A. Saldanha, R. Brayton, and A. Sangiovanni-Vincentelli. Delay fault coverage and performance tradeoffs. *Proc. Design Automation Conference*, June 1993.

[29] E. Liu, H. Chang, and A. Sangiovanni-Vincentelli. Analog system verification in the presence of parasitics using behavioral simulation. In *Proc. Design Automation Conference*, June 1993.

[30] E. Liu, G. Gielen, H. Chang, and A. Sangiovanni-Vincentelli. Behavioral modeling and simulation of data converters. In *Proc. IEEE Int. Symposium on Circuits and Systems*, pages 2144–2147, May 1992.

[31] E. Liu and A. Sangiovanni-Vincentelli. Behavioral representation for vco and detectors in phase-lock systems. In *Proc. IEEE Custom Integrated Circuits Conference*, pages 1231–1234, May 1992.

[32] E. Liu and A. Sangiovanni-Vincentelli. Behavioral simulation for noise in mixed-mode sampled-data systems. In *Proc. IEEE ICCAD*, pages 322–326, Nov 1992.

[33] E. Liu, A. Sangiovanni-Vincentelli, G. Gielen, and P. Gray. A behavioral representation for nyquist rate A/D converters. In *Proc. IEEE ICCAD*, pages 386–389, November 1991.

[34] V. Ma, J. Singh, and R. Saleh. Modeling, Simulation, and Optimization of Analog Macromodels. In *Proc. IEEE Custom Integrated Circuits Conference*, pages 12.1.1–12.1.4, May 1992.

[35] H. J. De Man, J. Rabaey, G. Arnout, and J. Vandewalle. Practical implementation of a general computer aided design technique for switched capacitor circuits. *IEEE Journal of Solid State Circuits*, SC-15:190–200, April 1980.

[36] H. A. Mantooth and P. E. Allen. Behavioral simulation of a 3-bit flash ADC. In *Proc. IEEE Int. Symposium on Circuits and Systems*, pages 1356–1359, 1990.

[37] A. Matsuzawa, S. Nakashima, I. Hidaka, S. Sawada, H. Kodaka, and S. Shimada. A 6b 1GHz dual-parallel A/D converter. In *Proc. IEEE International Solid-State Circuits Conference*, pages 174–175, February 1991.

[38] R. McCharles. Charge circuits for analog LSI. Ph.d. thesis, University of California at Berkeley, 1980.

[39] J. McCreary and P. R. Gray. All-mos charge-redistribution analog-to-digital conversion techniques, part i. *IEEE Journal of Solid State Circuits*, SC-10:371–379, December 1975.

[40] Meta-Software. *HSPICE User's Manual H9001.* Meta-Software, 1990.

[41] C. Michael and M. Ismail. Statistcial modeling of device mismatch for analog mos integrated circuits. *IEEE Journal of Solid State Circuits*, SC-27, No. 2, November 1992.

[42] M. Pelgrom, A. Duinmaijer, and A. Welbers. Matching properties of mos transistors. *IEEE Journal of Solid State Circuits*, SC-24, No. 5, October 1989.

[43] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes in C. The Art of Scientific Computing*. Cambridge University Press, 1989.

[44] T. Quarles. SPICE3 version 3c1 user's guide. Memorandum UCB/ERL M89/46, U. C. Berkeley, 1989.

[45] S. Ross. *A First Course in Probability*, volume page 211. Macmillan Publishing Company, New York, second edition, 1984.

[46] G. Ruan. A behavioral model of A/D converters using a mixed-mode simulator. *IEEE Journal of Solid State Circuits*, SC-26, No. 3, March 1991.

[47] R. Saleh, J. E. Kleckner, and A. R. Newton. Iterated timing analysis and SPLICE1. In *Proc. IEEE ICCAD*, November 1983.

[48] R. A. Saleh and A. R. Newton. *Mixed-Mode Simulation*. Kluwer Academic Publishers, 1990.

[49] K. S. Shanmugan and A. M. Breipohl. *Random Signals Detection, Estimation and Data Analysis*. New York, J. Wiley & Sons, 1988.

[50] J. Singh and R. Saleh. iMACSIM: A program for multi-level analog circuit simulation. In *Proc. IEEE ICCAD*, pages 16–19, November 1991.

[51] T. Souders and G. Stenbakken. Modeling and test point selection for data converter testing. *IEEE International Test Conference*, 1985.

[52] T. Souders and G. Stenbakken. A comprehensive approach for modeling and testing analog and mixed-signal devices. *IEEE International Test Conference*, 1990.

[53] T. Souders and G. Stenbakken. Cutting the high cost of testing. *IEEE Spectrum*, March 1991.

[54] G. Stenbakken and T. Souders. Test-point selection and testability measures via QR factorization of linear models. *IEEE Transactions on Instrumentation and Measurement*, June 1987.

[55] S. M. Sze. *VLSI Technology*, volume page 226. McGraw-Hill Book Company, 1983.

[56] E. Tan. Phase-locked loop macromodels. Master's thesis, U. C. Berkeley, August 1990.

[57] L. J. Tick. The estimation of the transfer functions of quadratic systems. *Technometrics*, 3:563–567, 1961.

[58] Y. P. Tsividis, P. R. Gray, D. A. Hodges, and Jr. J. Chacko. A segmented $\mu$-225 law pcm voice encoder utilizing nmos technology. *IEEE Journal of Solid State Circuits*, SC-11:740–747, December 1976.

[59] J. Vandewalle, H. De Man, and J. Rabaey. The adjoint switched capacitor network and its application to frequency, noise and sensitivity analysis. In *Circuit Theory and Applications*, pages Vol. 9, pages 77–88, 1981.

[60] R. A. Walker and D. E. Thomas. Design representation and transformation in the system architect's workbench. *Proc. IEEE ICCAD*, pages 166–169, 1987.

[61] J. White and A. Sangiovanni-Vincentelli. RELAX2.1 - a waveform relaxation based circuit simulation program. In *Proc. IEEE Custom Integrated Circuits Conference*, June 1984.

[62] L. A. Williams, B. E. Boser, E. W. Y. Liu, and B. A. Wooley. MIDAS user manual. *Center for Integrated Systems, Stanford University*, 1989.