DORIC: DESIGN OF OPTIMIZED AND ROBUST INTEGRATED CIRCUITS

by

Zeina Daoud

Memorandum No. UCB/ERL M93/90

15 December 1993

DORIC: DESIGN OF OPTIMIZED AND ROBUST INTEGRATED CIRCUITS

by

Zeina Daoud

Memorandum No. UCB/ERL M93/90

15 December 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

DORIC: DESIGN OF OPTIMIZED AND ROBUST INTEGRATED CIRCUITS

by

Zeina Daoud

Memorandum No. UCB/ERL M93/90

15 December 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

To my parents and family,

for their love and their faith in me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

My sincere gratitude goes to my advisor Professor Costas Spanos for his guidance and support, which have extended beyond the scope of this research project, to all aspects of my graduate school experience. I also wish to thank Professor Bernhard Boser for his review of this work and insightful comments.

Heartfelt thanks to all who have helped me along this path of learning: to my parents and family for their love and their faith in me, to those who have encouraged me to express myself in pride, to all my friends who have shared my joys and my sorrows. It is impossible to list their names here but it would not have been possible for me to achieve without their support.

Many people have helped me with this project. In particular, I wish to acknowledge Sherry Lee and Eric Boskin for their help and advice, Professor John Ousterhout for his helpful discussions of Tcl/tk, Enrico Malavasi for his help with the circuit examples, and Robert Neff for sharing his circuits design expertise.

I also wish to thank the members of the BCAM group for the camaraderie and the continuously warm welcome I have received: R. Chen, S. Cunningham, H. Huang, S. Leang, S.Y. Ma, T. Miranda, D. Rodriguez, P. Tsai, C. Yu, and past members: B. Bombay, J. Thomson, E. Wen. Special thanks to Sean Cunningham, the keeper of my sanity, and David Green. And last but not least, my deep gratitude to Yasmine Akkari, Micheal Deutscher, Joan Davidson and Dr. Sheila Humphreys for the generosity of their hearts and their continuous encouragements.

Chapter 1
Introduction

The Robust Design Method is a technique aimed at designing high quality products at low cost. It is based on optimizing performance, manufacturability and cost by varying certain decision variables, in order to make the product less sensitive to manufacturing imperfections. Previously, these variations were either ignored or studied in an ad hoc fashion, which often led to long and expensive design cycles. Using a mathematical tool called orthogonal arrays, the Robust Design Method explores many variables in a small number of trials.

This project investigates the application of the Robust Design Method to IC design using the HSPICE circuit simulator, and presents a design methodology to improve the manufacturability of integrated circuits. The developed computer-aided design tool, DORIC (Design of Optimized Robust Integrated Circuits) allows the user to study the effect of certain design parameters (such as transistor sizes) and manufacturing variations (e.g. variations of the thickness of the oxide) on specific circuit performance measures. Upon analyzing the results, the user can choose an optimal setting of the decision variables.

1.1 Previous work

The subject of tolerance design of integrated circuits was first studied in the early 1970s. By the early 1980s, two main techniques had emerged: a deterministic approach and a statistical approach [1].

Both techniques are concerned with determining the "region of acceptability" [2] of a given design. The region of acceptability of a design is defined as a mapping of the specifications onto the component parameter space. While the deterministic approach tries to precisely define the boundaries of that region, statistical methods focus on a rough estimation of the acceptability region, or at least the direction of parameter changes necessary to move towards the center of that region (design centering) [3].

1.1.1 Deterministic Approach

The deterministic approach, one representative of which is the simplicial approximation method (Director and Hechtel, 1977) [4][5] varies one parameter at a time, until the circuit no longer satisfies performance requirements. By varying all parameters similarly, the boundaries of the region of acceptability are discovered. Parameter targets are then set at the center of that region or as close to it as possible.

The biggest disadvantages of this method are that first, statistical process variation is often not stationary, and second, the complexity of this method increases dramatically with the number of adjustable parameters. Because of this, it is not practical to apply the simplicial approximation method to circuits with more than five design parameters [2].

1.1.2 Statistical Approach

The statistical exploration approach to tolerance design [6] is based on Monte Carlo analysis techniques or the Response Surface Method. In the case of the Monte Carlo technique, the actual circuit manufacturing variation is simulated by making random selections of component parameter values, given that the values come from a known statistical distribution. Then, the performance

of each resulting circuit is evaluated by means of a circuit analysis package. The total yield is estimated from the number of those circuits which meet specifications.

An important property of the Monte Carlo analysis is that the accuracy of the result is not dependent on the number of parameters considered. This accuracy, however, depends on the number of simulations performed and increases with the square root of the sample size. Thus the computational requirements of this method inhibits its use for exploring generally large circuits.

The Response Surface Method [15] relies on statistical experimental design techniques such as factorial designs, to determine a number of parameter setting combinations needed for modeling circuit performances. The RSM requires fewer runs than the Monte Carlo analysis, but its accuracy depends on the number of parameters considered and the type of experimental design used.

1.1.3 Design for Manufacturability

Early efforts in the area of Design for Manufacturability (DFM) have focused on modeling the effects of the variability of manufacturing parameters on circuit performances [7][8][9] and drawing conclusions about yield prediction. Little work has been done to provide designers with a complete methodology to optimize designs for robustness to manufacturing variations, along with more traditional circuit performances such as speed, power, area, etc., as early in the design cycle as possible. Moreover, it is important that such a methodology be supported and implemented by an automated framework or a set of CAD tools. Previous work done at the University of Illinois [10] on building an interactive statistical design tool for MOS VLSI circuits has implemented a "Modified Taguchi Method" (MTG) which was based on the concept of minimizing the squared-error loss function. For MTG (as well as for a standard regression analysis), a performance model has to be stipulated. In their example, a quadratic model was used. This introduces two types of problems: one is that of guessing the appropriate model for every performance; and second, in order to estimate second order effects, more simulations are used, even though it is not clear whether they are necessary. Moreover, the i-EDISON [10] approach tries to optimize "automati-

cally", leaving the designer out of the decision loop. Multi-objective optimization with subjective trade-offs, common in actual circuit designs, is not discussed.

Due to the large cost of actual circuit experimentation on the manufacturing line, computer-based experiments have been widely used [11][12]. Some of the problems accompanying simulation-based experimentations are tuning the simulator to match an actual manufacturing line [9] and the lack of random error [10][13]. Some of these issues will be further developed in Chapter 2.

1.2 The Robust Design Method

Our approach consists of a complete design methodology for optimizing circuit performances and robustness to manufacturing variations. It is computer-based, built around the circuit simulator HSPICE [14], and relies on a statistical experimental design method called the Robust Design Method.

The Robust Design Method (RDM) draws on many ideas from design of experiments in order to plan experiments for obtaining information about variables involved in making engineering decisions. Applied in the context of circuit design, this method does not explicitly try to define the region of acceptability, but instead tries to find an optimal setting within the region we are exploring. In several experimental design methods, various types of matrices were used for planning experiments to study several decision variables simultaneously, like full or partial factorial designs [15]. Among them, the RDM makes heavy use of orthogonal arrays, whose use for planning experiments was first suggested by Rao [16]. The fundamental principle of RDM is to improve the quality of a product by minimizing the effect of the causes of variations without eliminating the causes. The RDM relies on the assumption that the model is additive and thus uses orthogonal arrays to define the minimum subset of the design space needed to determine the main effects of parameters. The orthogonal arrays lead to a consistently small number of experiments.

In return for the small number of experiments, orthogonal arrays impose some assumptions on the exploration. These will be discussed later.

The Robust Design Method was introduced by G. Taguchi in Japan, who applied it to a wide variety of engineering problems. AT&T Bell Laboratories introduced Taguchi's method in the United States, by applying it to improve the quality and reduce the cost of window photolithography [17]. This study proposes to apply this method to integrated circuit tolerance design as described in Chapter 2.

1.3 Thesis Organization

Chapter 2 describes the theory of the Robust Design Method and outlines its application to IC design. DORIC (Design of Optimized Robust Integrated Circuits) is presented in Chapter 3 as the CAD tool developed to support a methodology of IC design optimization based on the Robust Design Method. In Chapter 4, examples and results are shown of the use of DORIC to optimize basic analog circuit building blocks such as a comparator and an operational-amplifier. Finally, conclusions are summarized and future work is suggested in Chapter 5.

Chapter 2
Robust Design Method applied to IC design

This chapter describes the Robust Design Method and how it is applied to IC design. Section 2.1 describes the fundamentals of the RDM. In Section 2.2, the details of the RDM are presented and its application to IC design is explained. Section 2.3 discusses the topic of error analysis on computer-based experiments.

2.1 Overview of the Robust Design Method

A design's performance degrades because of variations in process parameters (or *noise factors*) though a complicated, non-linear function. While several combinations of parameter values may give the desired output performance under nominal noise conditions, very different performance characteristics may result under varying noise conditions. The Robust Design Method exploits the non-linearity to find a set of design parameter values that cause the smallest deviation of the quality characteristic from its desired target [18].

In previous work, optimal sets of design parameter values were found by intuition or by trial-and-error or by performing large numbers of simulations. An attempt to study each parameter alone and measure its effect on the product's performance can be costly and time-consuming. The Robust Design Method explores only a subset of the design space and draws conclusions based on the results of that subset. It uses a mathematical tool called the *orthogonal array* to study a large number of decision variables with a small number of trials.

To that end, an *additive model* of factor effects of variables is assumed. An additive model of n parameters P_1 through P_n is of the form $\sum_i (\alpha_i P_i)$. This implies that each parameter (also called factors) has an effect that does not depend on other parameters. This assumption may, at first, seem unjustified, since by experience, we know that many parameters interact. However, on one hand, it is conceivable that even though some parameters may interact, their interaction may be small when compared to other factor effects. On the other hand, parameters that strongly interact can be lumped as one input to the Robust Design Method since a given setting of one has direct impact on the value of the other. Moreover, the assumed model is of a sum of logarithmic functions of the input parameters. Therefore, if parameters may interact, their logarithms will not necessarily do so; $\log(ab) = \log(a) + \log(b)$ therefore an interaction term which might exist in a linear response is translated to an additive term in a logarithmic response. Moreover, even if the logarithms interact, their interaction is likely to be smaller than that of the parameters directly. In any case, the results will show if the parameters picked by the designer have a significant interaction.

2.2 Steps of the Robust Design Method

To solve this optimization problem systematically, the problem is defined, the performances to optimize are chosen and the varying parameters are identified. The orthogonal arrays are used to define the matrix experiment. A matrix experiment consists of a set of trials where settings of various parameters (or factors) are modified from one trial to another. Orthogonal arrays are such that their columns are mutually orthogonal. For the Robust Design Method, this means that, in any

two columns, all combinations of factor levels occur, and they occur an equal number of times. For example, table 1 shows the orthogonal array for a problem of 4 parameters A, B, C and D,

Table 1: Example of an orthogonal array for a 4 parameters at 3 settings problem

run number	Factor A	Factor B	Factor C	Factor D
1	A1	B1	C3	D2
2	A1	B2	C1	D3
3	A1	B3	C2	D1
4	A2	B1	C2	D3
5	A2	B2	C3	D1
6	A2	B3	C1	D2
7	A3	B1	C1	D1
8	A3	B2	C2	D2
9	A3	B3	C3	D3

where each parameter can take on three values (A_1, A_2, A_3, B_1, B_2,...).

Once the orthogonal array is chosen, a quality measure, which we will call *quality metric*, is calculated for every output function to optimize, for each run. Taguchi calls this metric *signal-to-noise ratio* but we do not wish to confuse it with the concept of signal to noise in the circuit design world. The quality metric (QM) for each performance, defined as appropriate ratios of performance value over its sensitivity, is an effective measure of design robustness and specification compliance. For instance, the QM of the sensitivity of speed to changes in the thickness of the oxide (tox) is $QM = 10 \log \left(\frac{\text{speed}}{\text{sensitivity of speed to tox}}\right)$.

The quality metric has two characteristics: first, it is defined for each performance such that the performance is *optimized* when the QM is *maximized*, regardless of whether the physical value of the performance is maximized or minimized; second, the QM is a logarithmic function of the performance metric. This logarithmic function aids in ensuring additivity of the model, while reducing the effect of potential interactions between parameters.

After the experiments dictated by the orthogonal array are completed, the factor effect of every parameter on every output function is calculated. The factor effect of parameter P on the output function F is defined as the amount by which P contributes to the quality characteristic of F. The orthogonality of the experiment matrix simplifies this calculation. The factor effect FE of the parameter P, set at level L, is computed for each performance as shown below:

$$FE_{P_L} = \overline{QM}_{P_L} - \overline{QM}$$

$$FE_{P_L} = (\frac{1}{m}) \sum_{i=1}^{m} QM_{P_L} - (\frac{1}{n}) \sum_{i=1}^{n} QM$$

where $\overline{QM}$ is the output average of all n trials (expressed in "quality metric" units or decibels (db)) and $\overline{QM}_{P_L}$ is the output mean of the m trials where parameter P is set to level L. The factor effect plot of a given function represents then a summary of that performance's variation under the effect of each parameter setting. An example of a factor effect plot is shown in Figure 4 in Chapter 3. The combined graphs of the factor effect plots for all the performances provide a powerful and concise quantitative summary of the design trade-offs. With the factor effect plots, the designer gets a clearer understanding of the impact of engineering compromises on the design at hand.

The final step of the Robust Design Method is the confirmation cycle: the designer picks a combination of parameter settings which optimizes desired output functions. This optimal setting combination might not be one of the trials in the experiment. Given this setting combination, performance predictions can be calculated based on the factor effect model, and those values compared to the actual results obtained from executing the experiment at those settings.

In summary, the objective of applying the Robust Design Method to IC design is to systematize the search in the design space needed to satisfy the many objectives of modern custom IC design. The application is straight-forward: the designer is presented with a design to optimize, given certain performances of interest and some non-interacting parameters to vary. An orthogonal array is derived to serve as an experiment matrix. The experiments are run with the aid of a

circuit simulation tool, such as HSPICE. Based on the factor effect plots, the designer can make a design compromise, with quantifiable trade-offs in mind. The only peculiarity of such an application of the Robust Design Method is the handling of the error analysis. Since the experiment run is a computer simulation, the results of the experiments do not exhibit random error (due to traditional experimental noise), but rather deterministic error (due to numerical lack of fit). This implies that the interpretation of the discrepancy between a predicted performance and the measured results must be slightly modified. The following section discusses how the error is handled in the traditional Robust Design Method and how it is modified for user with computer-based experiments.

2.3 Error analysis in the Robust Design Method

Computer-based experiments are characterized by the lack of noise or random error. The output of computer-based experiments is deterministically replicated with the same inputs. Therefore traditional statistical error analysis (such as R^2 statistics) is inappropriate for deterministic experiments. An estimation of the error is however critical to a model because it is a measure of the "goodness" of the model. One must be reasonably confident that once the model is built, it is predicting accurate results.

In real-world experiments (as opposed to computer-based experiments), the error of a model is of two types: a *random noise error*, which is responsible for real output discrepancies given identical inputs, when the experiment is executed at different times; and a *lack of fit error* which is responsible for output discrepancies between what a model predicts and the mean process response for a given input. In real-world experiments, the assumption is that random error has the same statistical profile all over the design space, it is not localized to the experiment space and can therefore be used as a measure of the goodness of the model over the entire space. Lack of fit error however is very dependent of the location of the experiment in the design space and therefore cannot be a measure of the quality of the model over the entire design space. Some attempts have been made to model the deterministic output of computer-based experiments as a stochastic func-

tions that models experiments whose replication errors are spatially correlated through the experimental space [19][20]. The problem with that approach is that the number of unknowns in the stochastic model is twice the dimension of the input space [20], and this requires many additional experimental points to fit such a model for the error.

One is tempted to resort to heuristics, such as in [10], where it was noted that "since there is no randomness in the circuit simulation, statistical model testing will be inappropriate. Nevertheless, the R^2 statistic [...] are larger than 0.99, suggesting that [the] models fit well." The definition of the R^2 statistic, however relies heavily on statistical distributions which are only meaningful when random error is present.

In this work, we have chosen another heuristic measure, the root-mean-squared (r.m.s.) error of the data taken at the experimental points. It is an engineering measure of the numerical lack of fit of the model. The r.m.s. error is actually an average of the numerical lack of fit of the model at the points of the experiment. The heuristic assumption is that the lack of fit of the model over the entire design space is comparable to the lack of fit of the model over the experimental points. The root-mean-squared error is calculated as:

$$\text{rms} = \sqrt{\frac{\sum_{i}^{n} (y_i - \hat{y}_i)^2}{n}}$$

where n is the number of experimental points, y_i is the measured value of the output function y at the i-th experiment and $\hat{y}_i$ is the model prediction for output y at the i-th experiment.

We generalize the use of r.m.s. to the entire design space by assuming, that if the difference between the predicted output and the measured output of the simulator, at any given point in the space is less than three times the root-mean squared error of that model, then the model is good. If however the discrepancy between predicted and measured values are greater than three times the r.m.s., then the model considered is invalid.

If the model is invalid, this implies that the form of the model is unsuitable for our application. Given that the most restrictive aspect of our model is the assumption of additivity, an invalid model is a most probable indication of a violation of the additivity assumption. If this is the case, the designer has the following options:

a) reduce the size of the experimental space in order to separate the interacting terms and obtain a better fit.

b) pick an appropriate orthogonal array that can support calculation of interaction effects: certain orthogonal arrays have corresponding interaction tables [18] that guide the assignment of parameters to columns of the orthogonal array, so that specific columns if left unused, can help determine the interaction effect between two other columns. Interaction tables only support two-parameters interaction, however, and not all orthogonal arrays have by construction, corresponding interaction tables.

c) use another analysis technique if necessary, that takes into account interactive terms, at the expenses of additional experimental runs (e.g. full factorial, central composite designs, Monte Carlo analysis,...).

Chapter 3
Description of DORIC

3.1 System Overview

DORIC (Design of Optimized and Robust ICs) is the CAD tool developed to apply the Robust Design Method described in Chapter 2 to the problem of optimizing performances of IC circuits. DORIC is intended for IC designers who, given a functional circuit design, wish to increase the robustness of their designs to manufacturing variations, while ensuring equivalent or better performances in the traditional sense (e.g. speed, power, area, etc...). A high level description of this tool is outlined here and more details of the low level description follow in Section 3.2.

DORIC's architecture is composed of three functional units: the front end pre-processor and user interface, the core optimizer and the back end post-processor and user interface. The block diagram of Figure 1 outlines the interaction of the designer with this system.

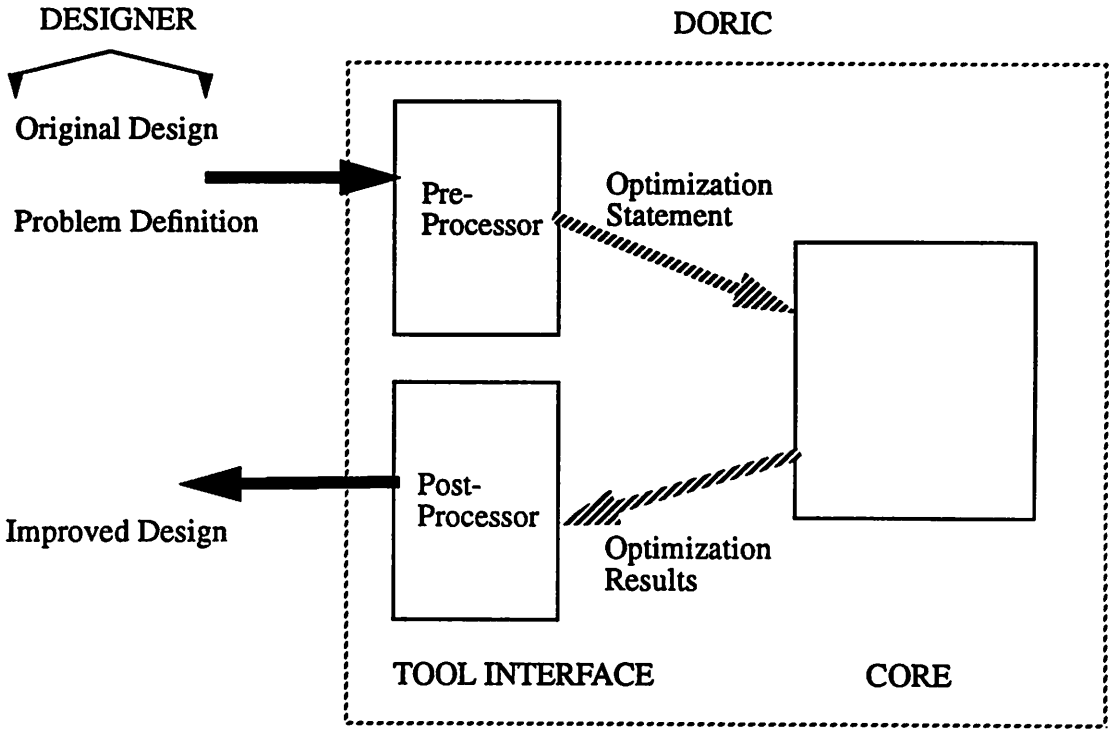


Figure 1 System Block Diagram

The input to DORIC is an original design circuit schematic (in the form of a SPICE netlist). The program is also provided with the problem statement, namely a set of performances to optimize and a set of variables representing the design space to be explored. The output of DORIC is a circuit whose performance has been optimized while its sensitivities to process variations have been reduced.

The pre-processor user interface helps the user specify the problem definition. The optimization problem is internally stated and submitted to the core. Upon completion of the optimization, the results in the form of plots are presented to the user through a graphical post-processor. Some other features and functionalities are included in the pre-processing and post-processing module, as discussed in the next section.

3.2 Details

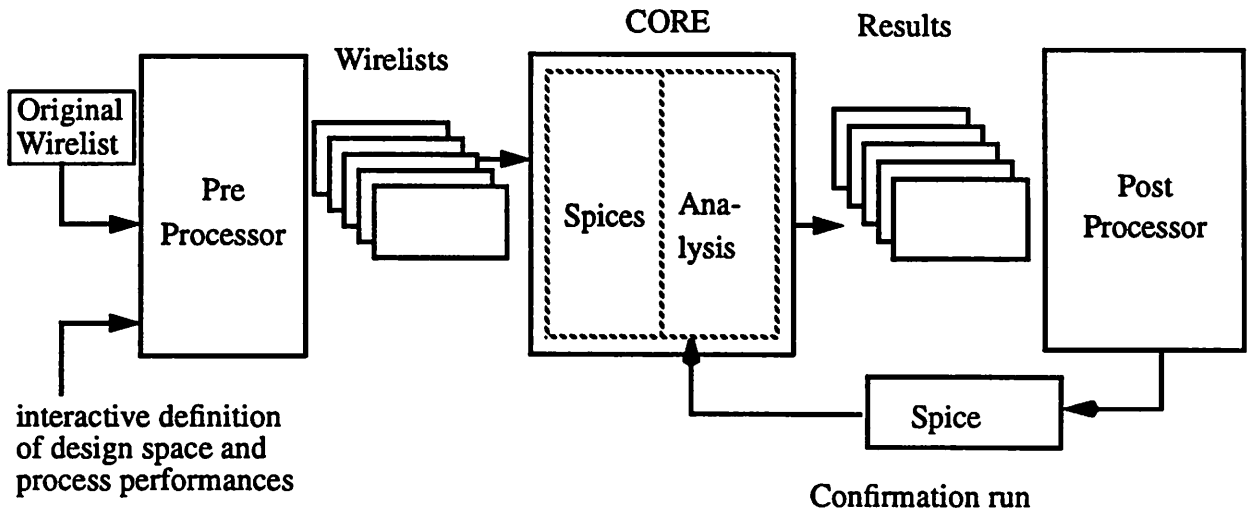


Figure 2 DORIC System Architecture

Figure 2 reveals DORIC's system architecture. The pre-processor and post-processor's main tasks are to help formulate and analyze the optimization problem through a specifically designed user interface, with additional functionality outlined below. The core optimizer is divided into two parts: the wrap-up around HSPICE which schedules the circuit simulation runs and the analysis part which is based on the Robust Design Method.

This section details the optimization procedure flow and highlights the features of DORIC. Four main steps are identified in optimizing a circuit with DORIC: pre-processing, core optimization, post-processing and the confirmation loop.

3.2.1 Pre-processor

The first step in solving the circuit optimization problem is to provide the system with the problem definition. This includes the design(s) of interest in the form of one or more spice decks, the performances to optimize and the design space variables to vary, as well as the range of values they can take. The front end user interface (Figure 3) provides the user with an environment that

facilitates this definition task. Another feature of the pre-processor is to identify the smallest orthogonal array that fits the problem at hand, and based on that determine, the expected number of simulation runs. The number of simulation runs is the number of experiments (or rows in the orthogonal array) or a multiple of it, if sensitivity analysis is requested. Finally, the pre-processor translates the problem definition in a format that is understood by the core optimizer and submits the job to it.

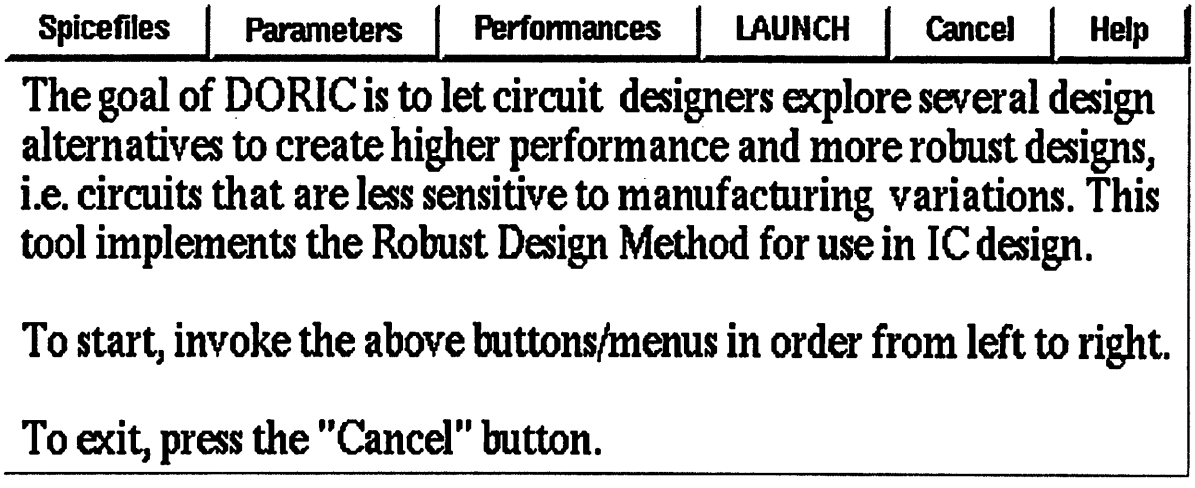


Figure 3 DORIC's pre-processor window

3.2.2 Core

The optimizer's body is divided into two distinct modules: the first is the automatic HSPICE file generator and circuit simulation, the second is the analysis module. Those two modules are independent so as to permit replacement of one or the other with another equivalent module. For instance, HSPICE can be replaced with another circuit simulator, or the analysis module which is now based on the Robust Design Method can be replaced with another statistical method (factorial designs, Monte-Carlo simulations, etc...).

The spicefile generator inside the core creates the entire set of wirelists which are based on the original user provided wirelist(s) and modified according to the experimental matrix (orthogonal array). The number of wirelists is equal to the number of simulation runs determined by the pre-

processor decided on. Next, all the spicefiles are submitted to HSPICE. DORIC currently only supports sequential submission of HSPICE jobs, however it could lend itself very easily to parallel simulation submissions on remote machines across the computer network. The simulations are run and the circuit simulator output is parsed to extract values of the performances of interest. These raw results of the circuit simulation are submitted to the analysis module.

The analysis module implements the Robust Design analysis method. Sensitivities are calculated by perturbation: the sensitivity of performance P on variation in parameter x is defined as $\text{sensitivity } S = \frac{\Delta P}{\Delta x}$. The quality metric, defined as the logarithm of the ratio of the performance value over its sensitivity, is expressed so as to always maximize the QM whenever the performance is improved. For instance, the quality metric of the sensitivity of speed to changes in the thickness of the oxide is:

$$\text{Quality Metric} = 10 \log \left(\frac{\text{speed}}{\text{sensitivity of speed to } \Delta\text{tox}}\right) = 10 \log \left(\frac{\text{speed}}{\frac{\Delta\text{speed}}{\Delta\text{tox}}} \right)$$

Factor effect plots are then generated and passed on to the post-processor for display and interaction with the designer. As explained in Chapter 2, a factor effect plot as the one shown in Figure 4 is a graphical representation of the contribution of every parameter for each setting, to the output performance studied. The plot is normalized so that the mean of the performance is at 0 db. In the example shown below, the width of transistor m1, when set at setting 3 (which was defined in that problem as 2 microns), contributes 0.99 db above the performance mean of the quality metric of the power. Note once again that the QM is defined such that the performance is optimized when the QM is maximized. Therefore, a higher point on the factor effect plot signifies a better performance.

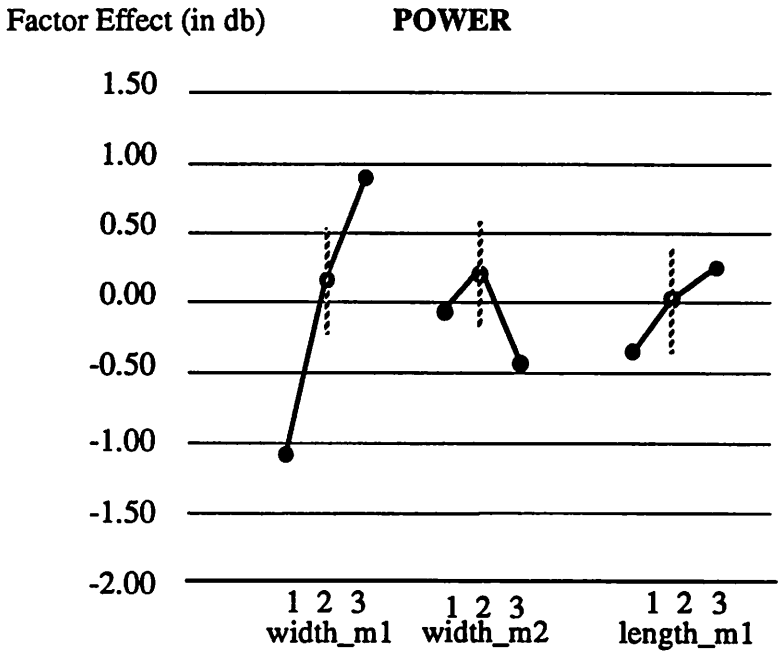


Figure 4 Example of a factor effect plot of power

A factor effect plot represents the effect of each factor on a given output performance. Several objectives can be studied together by simultaneous consideration of multiple factor effect plots.

3.2.3 Post-processor

The post-processor's two main tasks are to display the optimization results and to support the confirmation loop. First, the factor effect plots generated by the analysis module of the core are displayed to the designer, representing a powerful visual tool for summarizing engineering design trade-offs. By viewing the factor effect plots of all performances, the designer is able to select an optimal set of parameter settings with a clear idea of which performances would exhibit the most improvement at these setting as compared to the original design. The user selects a setting with a mouse click on the parameter level of interest. The combination of parameter levels chosen may or may not have been part of the original experimental matrix. The user is then prompted to confirm the assumptions by engaging in the confirmation stage, or confirmation loop. Figure 6 in

Chapter 4 represents the output of the post-processor and the action buttons that support the confirmation stage.

3.2.4 Confirmation stage

For every combination of parameter settings, DORIC can be prompted to either provide only performance predictions (by pressing the button "show prediction") or to predict performance values and confirm the predictions by running the appropriate spicefile (by pressing the button "confirm"). Performance predictions are calculated as the sum of the factor effect of each parameter at its given level on that performance. For confirmation, the appropriate spice wirelist is setup, the simulation is performed, raw spice results are collected and processed to obtain the actual performance values. The predicted performance values and the ones measured from simulation are compared for accuracy. Potential discrepancies between predicted performances and simulation results are explained in Section 2.3 which dealt with the details of the error analysis.

Several such confirmation loops can be executed if desired. The intent is not to exhaustively explore the design space, since that would contradict the initial choice of the Robust Design Method. However, it is recognized that many optimal designs may exist and DORIC allows the designer to explore such alternatives and quantify the trade-offs. When a satisfactory design decision is reached, the user commits the parameter choices and DORIC creates the final wirelist that appropriately reflects the design choices.

3.2.5 On balancing multiple performance objectives

In the development of DORIC, a conscious decision was made to avoid incorporating an automatic decision-making entity which would attempt to pick an optimal design based on some predefined priority criteria. We believe that the designer needs to remain involved in the decision making loop through the confirmation stage to fully explore and understand the design trade-offs. The tool can help by eliminating non-viable options (such as parameter settings that would be det-

rimental to all performances at once), however, the tool must not try to mask the compromises nor bias the choice of an optimal design.

Chapter 4
Examples and Results

This chapter details the use of DORIC to improve performances and robustness of two basic analog building blocks: a comparator and an operational amplifier. These two examples have been hand-optimized by expert circuit designers before the DORIC application, so any improvements that DORIC might suggest were not overlooked by the designers. However, the tool is not intended to have a pre-optimized application, but merely a functional design. The design methodology we have adopted as well as some features of DORIC are highlighted below.

4.1 Example 1: a clocked comparator

4.1.1 Brief circuit description

The clocked comparator circuit chosen for this example is shown in Figure 5. Two non-overlapping clocks connected to inputs 13 and 14 implement set-reset phases. During the set phase, the comparator senses input voltage difference at the differential input (transistors M1 and M2) and amplifies it at the differential output (nodes 92, 93). During the reset phase, the output is

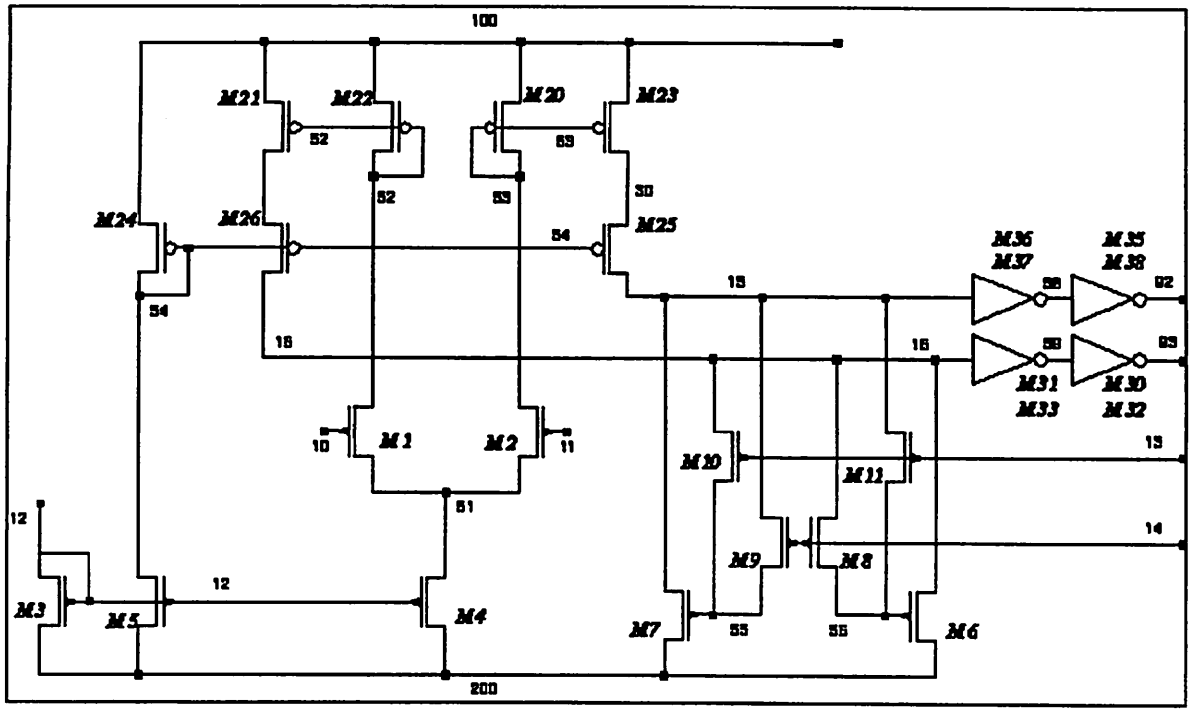


Figure 5 Schematic of the clocked comparator

brought back to zero. The biasing circuitry is implemented by transistors M3, M4, M5 and M24 and the bias current I_1 is 111μA. There is a 5 volt swing between Vdd and Vss and the capacitive loads at the output nodes 92 and 93 are 1pF each. A differential input voltage with zero common-ground is applied to the input pair M1-M2 and is converted into a current differential. The current is mirrored and amplified through the current mirrors and cascode configuration of transistors M20-M26. Transistors M8-M11 act as switches to the set-reset phases and transistors M6 and M7 implement the active loads that convert the amplified current differential into an output voltage differential.

4.1.2 Problem definition

We are interested in reducing the sensitivity of speed and power to changes in the effective channel length (Ld), while still maintaining acceptable speed and power performances. The speed of the comparator switching is measured from the clock (node 14) to the output (node 120) at 50% of the final value. The parameters we chose to vary are the lengths and widths of transistors M4

and M6, the width of transistors M1 and M10. M4 was chosen because it determines the amount of the bias current. M6 is important because it is acting as the active load to transform the differential current into differential output voltage. M8 and M10 act as switches for the set-reset phases. The parameter settings are defined in table 2, namely the second setting is chosen to be the original device size value, while the first and third settings reflect a 1micron change.

Table 2: Definition of parameter settings

factor	setting 1	setting 2	setting 3
width_M1 (in μm)	113	114	115
width_M4 (in μm)	11	12	13
width_M6 (in μm)	4	5	6
width_M10 (in μm)	2	3	4
length_M4 (in μm)	1	2	3
length_M6 (in μm)	1	2	3

In DORIC, it is possible to point to specific devices that must remain equal throughout the experiment. For instance, transistors M6 and M7 are active loads that must be matched. So whenever the size of M6 is varied according to the settings defined in table 2, M7 is set identically. Similarly M10 is matched to M11, and M1 to M2 (differential input pair). Given the circuit and these definitions of factor levels, the objective is to find a set of parameter values which optimize the desired output functions. It is conceivable, indeed likely, that there is not a unique set of values that will optimize all output functions.

4.1.3 Results from the application of DORIC

The orthogonal array that accommodates a problem of 6 parameters each at three settings is the matrix L_{18} shown in Appendix 1. This implies that 18 simulation runs are needed to calculate the main effects of the parameters on speed and power. However, since we are also interested in sensitivity of speed and power to changes in the effective channel length, 18 additional runs (hence a total of 36 runs) were performed in order to evaluate sensitivities by perturbing the value

of Ld with respect to the original 18 runs. As a comparison, a full factorial design would have required 2 times 3^6 (2 times 729) or 1458 simulation runs!

Within the framework of DORIC, the task of interactively defining the problem is taken care of by the pre-processing module (Figure 3), which also decides on the smallest orthogonal array needed to execute the design space search. Once the problem definition is completed, the designer sends the information to the core, after having confirmed the choices.

The core then executes its task without exposing the user to any of the theoretical and practical details described in Chapters 2 and 3. The post-processor returns to the user the factor effect plots as shown in Figure 6. The back end window contains one factor effect plot per chosen performance (four in our case), and the action buttons at the bottom of the screen. The user highlights points on the graphs as combination input settings for the prediction or the confirmation of the performances.

Looking across the parameters of one factor effect plot, we can pick out the best parameter settings to optimize that function. So for instance, in order to reduce the sensitivity of speed to changes in Ld, the length of transistor M6 had better be set to its first level (or 1 μm). At the same time, we can get a measure of the relative importance of certain parameters with respect to the output function. For instance, it is clear that the length of transistor M6 has a bigger effect on the sensitivity of speed to Ld than other parameters. Moreover, the width of transistor M4 does not have a very relevant effect on the sensitivity of power to Ld, given that all three setting produce almost identical values, and the difference between any two settings is less than the magnitude of the error bar. Table 3 summarizes the best choices of each parameter for every performance.

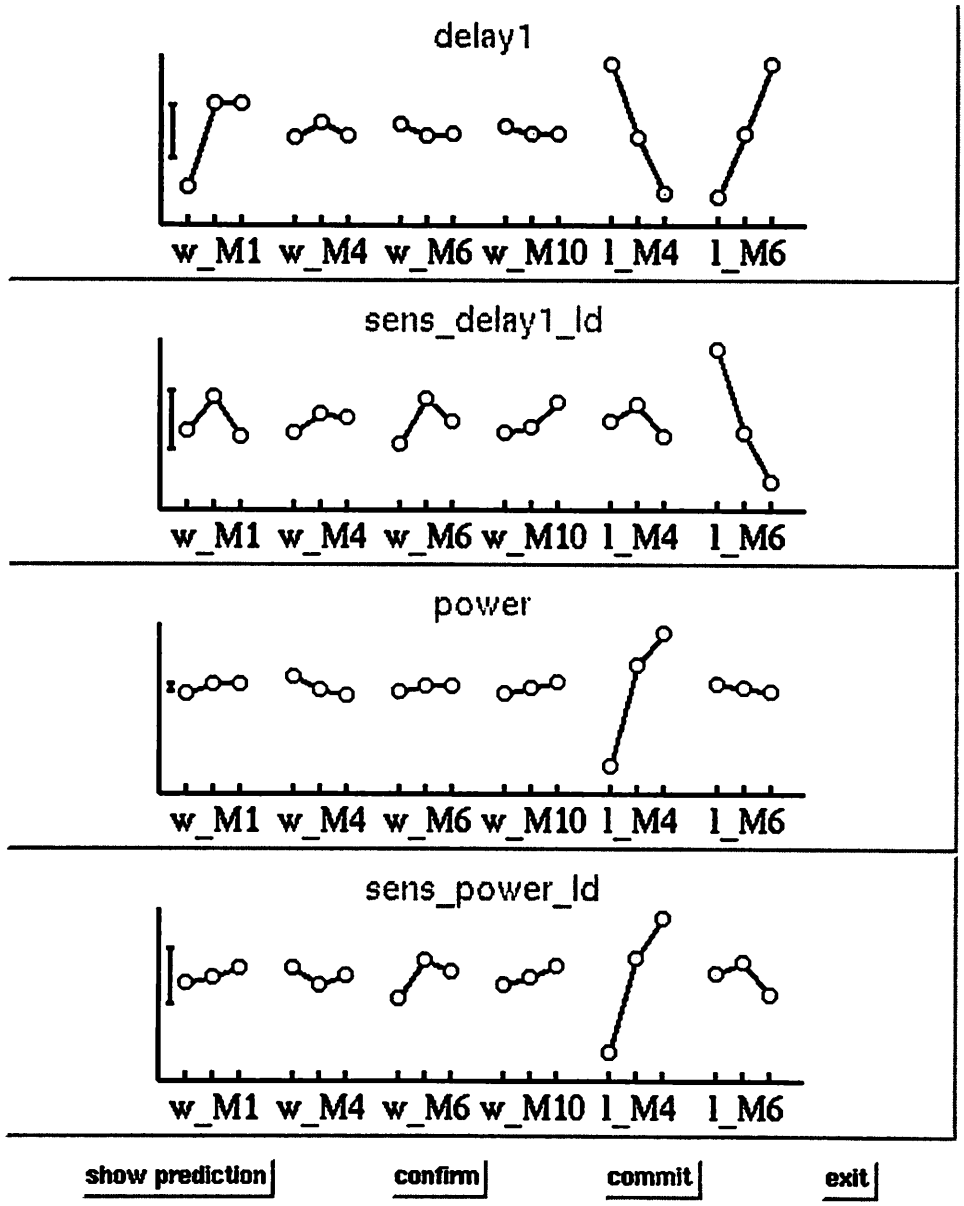


Figure 6 DORIC's post-processor: combined factor-effect plots

Table 3: Optimal parameter settings (for the comparator)

factors	speed	sens_speed _ ld	power	sens_power _ld	optimal choices
width_M1	2 or 3	any	any	any	2 or 3
width_M4	any	any	1	any	1
width_M6	any	any	any	any	any
width_M10	any	any	any	any	any
length_M4	1	any	3	3	3
length_M6	3	1	any	any	1 or 3

The last column is a selection of parameter settings that represent a few optimal design choices. Several choice criteria could be applied to select an optimal design. For instance, on one hand, if the designer was mostly concerned about reducing the circuit's speed sensitivity to variations in the thickness of the oxide, he or she can pick the optimal choice reflecting the setting combination that improve that performance. If, on the other hand, all performances are to be simultaneously optimized, as is the case in this example, the parameter settings were chosen that optimized most performances. The following confirmation runs were performed as indicated in table 4 and the quality metric (in db) collected from the output of the circuit simulator are gathered in Table 4, and compared to the original design (settings [2,2,2,2,2,2]).

Table 4: Confirmation Runs

run #	Input combination	delay (in db)	sens_delay_ld (in db)	power (in db)	sens_power_ld (in db)
0	[2222222] original	71.37	94.20	32.73	17.75
1	[212233]	71.32	91.40	33.43	22.22
2	[221131]	71.34	98.75	33.52	21.44
3	[311331]	71.33	99.51	33.62	21.58
4	[321131]	71.34	98.75	33.52	21.41

It is not necessary to run other confirmation runs, but it is encouraged to explore some of the options around the chosen optimal design point, in the hope of finding a combination that reduces all the sensitivities while not hurting the design performances. We chose to that end the settings in confirmation run number 3. It improves the robustness and the power with little sacrifice to the speed of the circuit.

An example of the interface showing the predicted and measured performance values is shown in Figure 7 for the input selection of [3,1,1,3,3,1]. Note that the discrepancy between predicted and measured values (calculated in Table 5) is much less than the error margin we set for accepting a performance model (which was three times the root mean squared error at the experimental points). We are therefore reasonably confident that the models are acceptable.

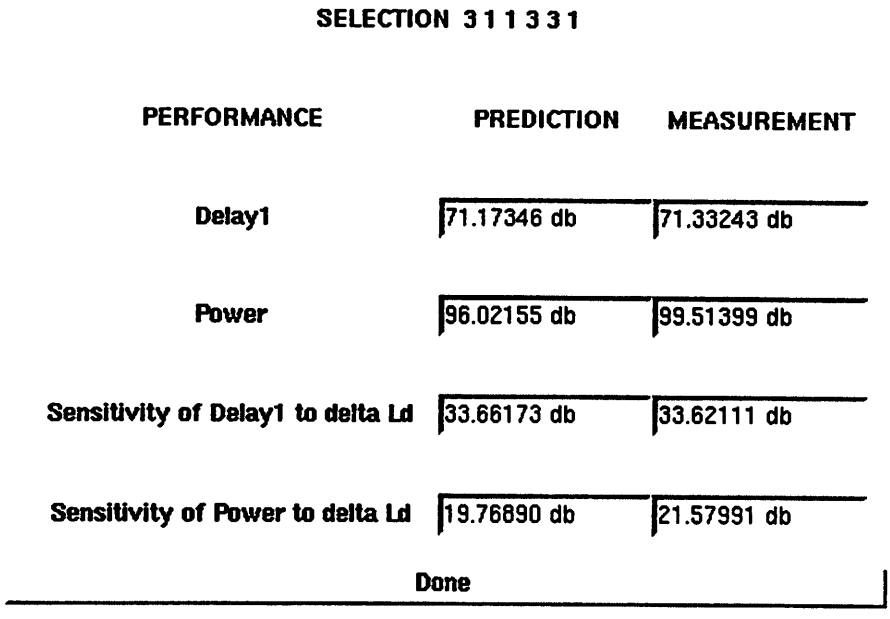


Figure 7 Predicted versus Measured performances

Table 5: Validating the models

performance	difference = | Pred - Meas |	acceptable error
delay	0.040217	0.473961
power	3.49244	6.851325
sens_delay_ld	0.04062	0.230679
sens_power_ld	1.81101	10.450419

4.1.4 Example 1 conclusions

An optimal design is then chosen to be the one reflecting the combination [3,1,1,3,3,1] because it improves the sensitivities over the original design [2,2,2,2,2,2], especially the sensitivities of power to manufacturing variations, while still not sacrificing the traditional performances. The improvement is further highlighted in Table 6.

Table 6: Improvements of the DORIC-optimized final design over the original

performance	original	optimal	improvement
delay (in ns)	72.95	73.62	-0.9%
sens_delay_ld (in ns/nm)	$5.2\ 10^{-3}$	$1.5\ 10^{-3}$	70.8%
power (in mW)	5.33	4.34	18.6%
sens_power_ld (in mW/mm)	31.48	16.00	49.2%

In spite of the fact that our original comparator design was hand optimized before it was used with DORIC, this methodology has successfully identified an equivalent circuit with significantly better manufacturability and equal or better performance.

4.2 Example 2: two operational-amplifiers

The first example illustrated some basic concepts and the use of DORIC in creating more robust and optimized circuits. The following example expands on the methodology and shows how DORIC (and the RDM method) supports *categorical* parameters, such as choices of circuit topologies. For this example, some of the details of DORIC which were presented previously are skipped. Emphasis is placed on the iterative methodology and results are shown at every stage of the process.

4.2.1 Problem Definition

The two operational amplifiers under study are a class A amplifier (Figure 8) and a class AB

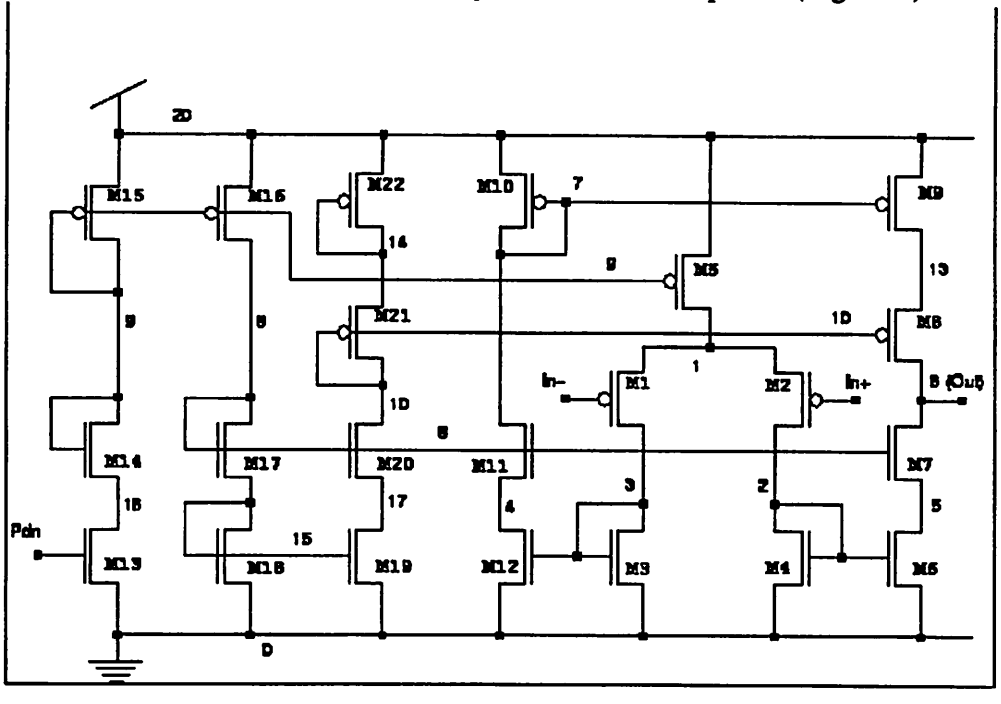


Figure 8 Class A amplifier schematic

amplifier (Figure 9). This is meant as an illustration of DORIC's feature of supporting different choices of topology, and it is understood that designers choose between these two amplifiers depending on their design application. The performances of interest are the following: power, gain and their sensitivities to changes in the effective channel length (Ld) and variations in the

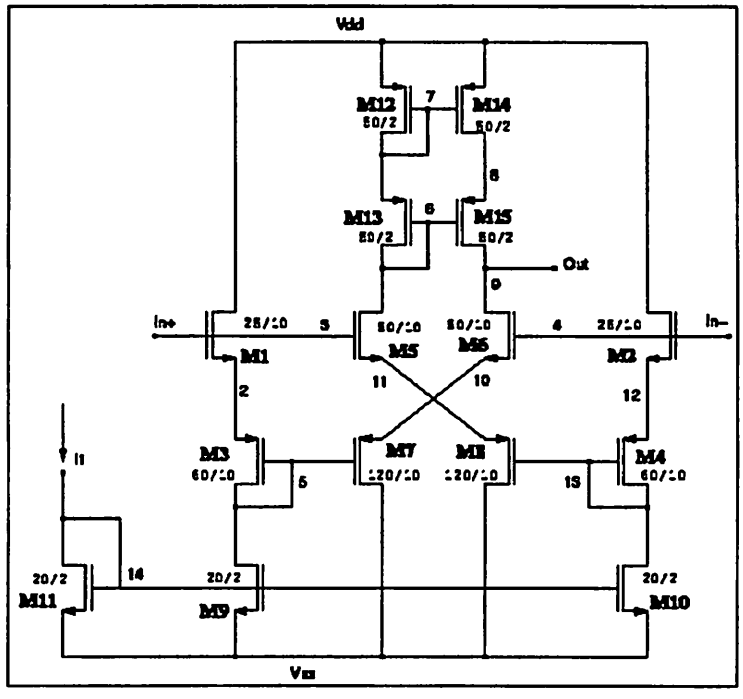


Figure 9 Class AB amplifier schematic

thickness of the oxide (Tox); and bandwidth. The varying parameters are the sizes of the differential input (transistors M1, matched to M2), the sizes of transistors M3 (which is matched to M4), M8 (matched to M7) and M5 (matched to M16). The setting for each numerical parameter were chosen: the original level S_0, S_0 + 1 μm and S_0 - 1μm. The total parameter input is then 7 numerical variables at 3 levels each and one categorical parameter (namely the topology) at 2 levels.

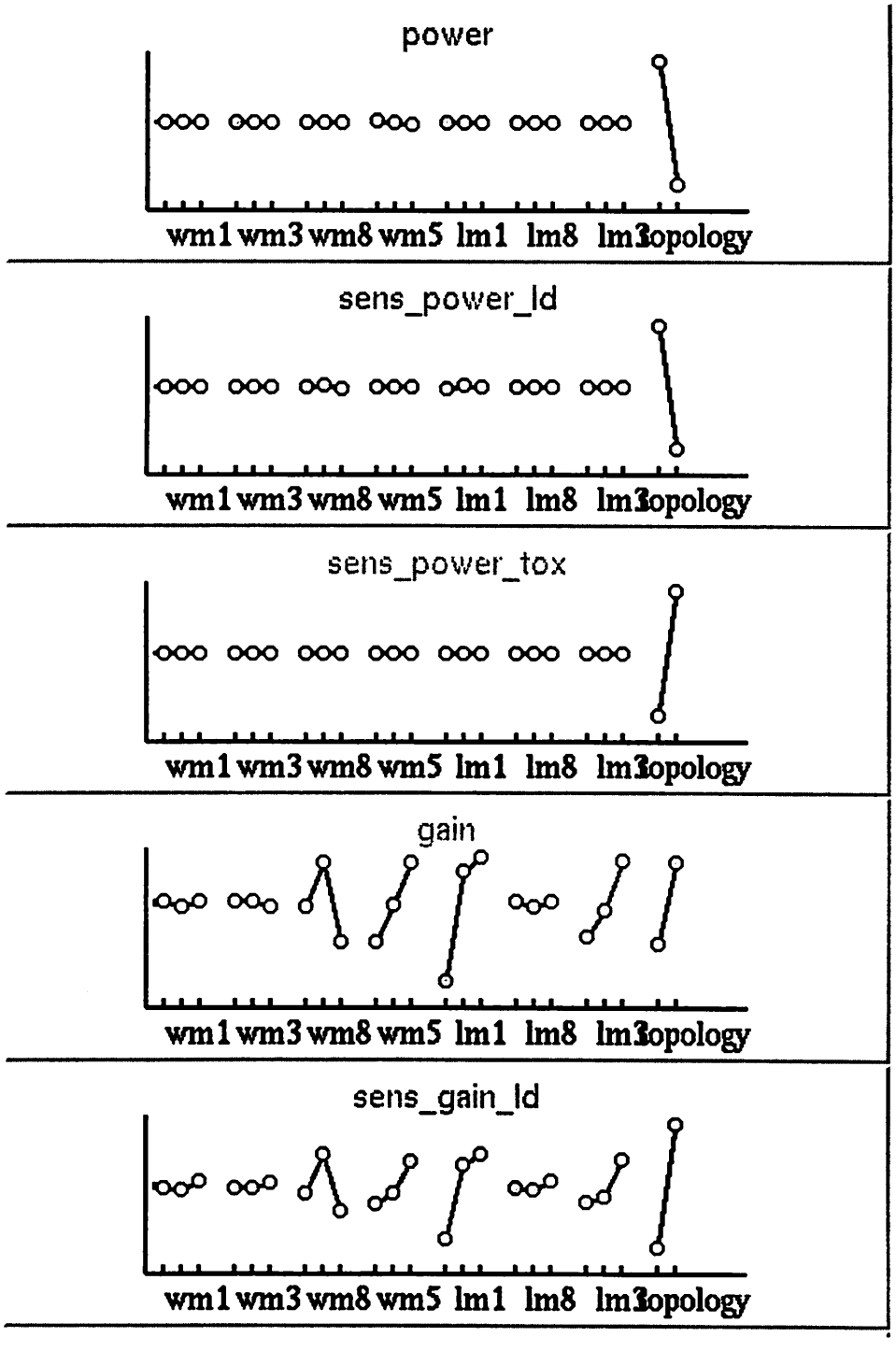
4.2.2 Results

The orthogonal array L_{18} was used, meaning that 18 simulation runs are needed to estimate nominal main effects. Two additional sets of 18 runs were required for perturbations of Tox and Ld, bringing the total to 54 simulation runs for 8 parameters.

The factor effect plot is shown in Figure 10. It is clear that the most important parameter is the choice of topology. This suggests that a more detailed study needs to be carried out on a particular circuit topology. Noting the trade-off of more power consumption for higher gain and bandwidth

which is apparent in these factor-effect plots, we choose higher gain and bandwidth, therefore setting the topology to its second choice (class AB amplifier).

This window displays the factor effect values of all the parameters (with various levels) on all the performances. Click on a point to select one level for each parameter as input to a confirmation run.



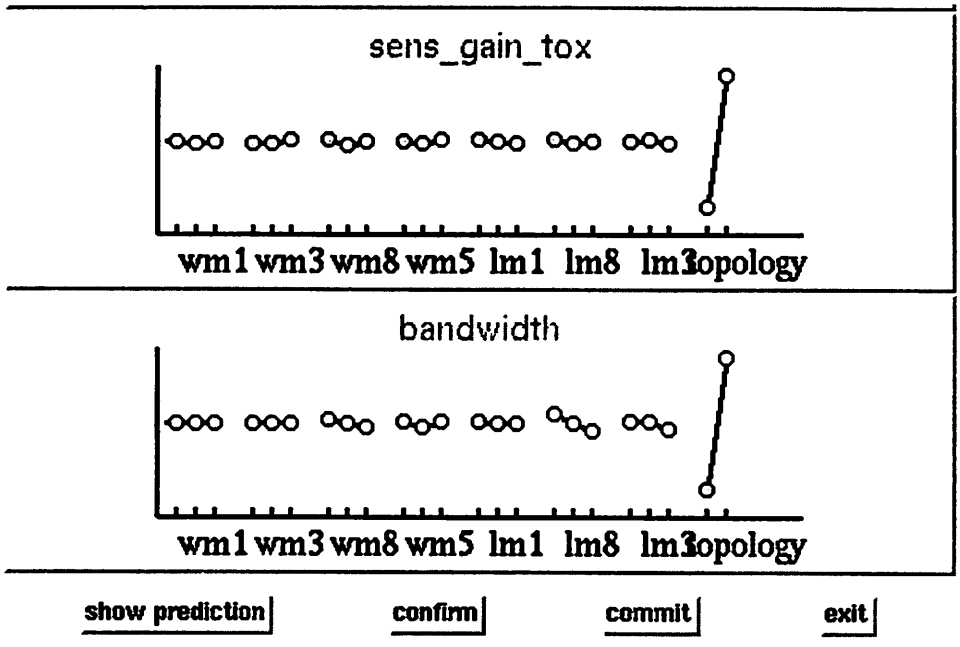


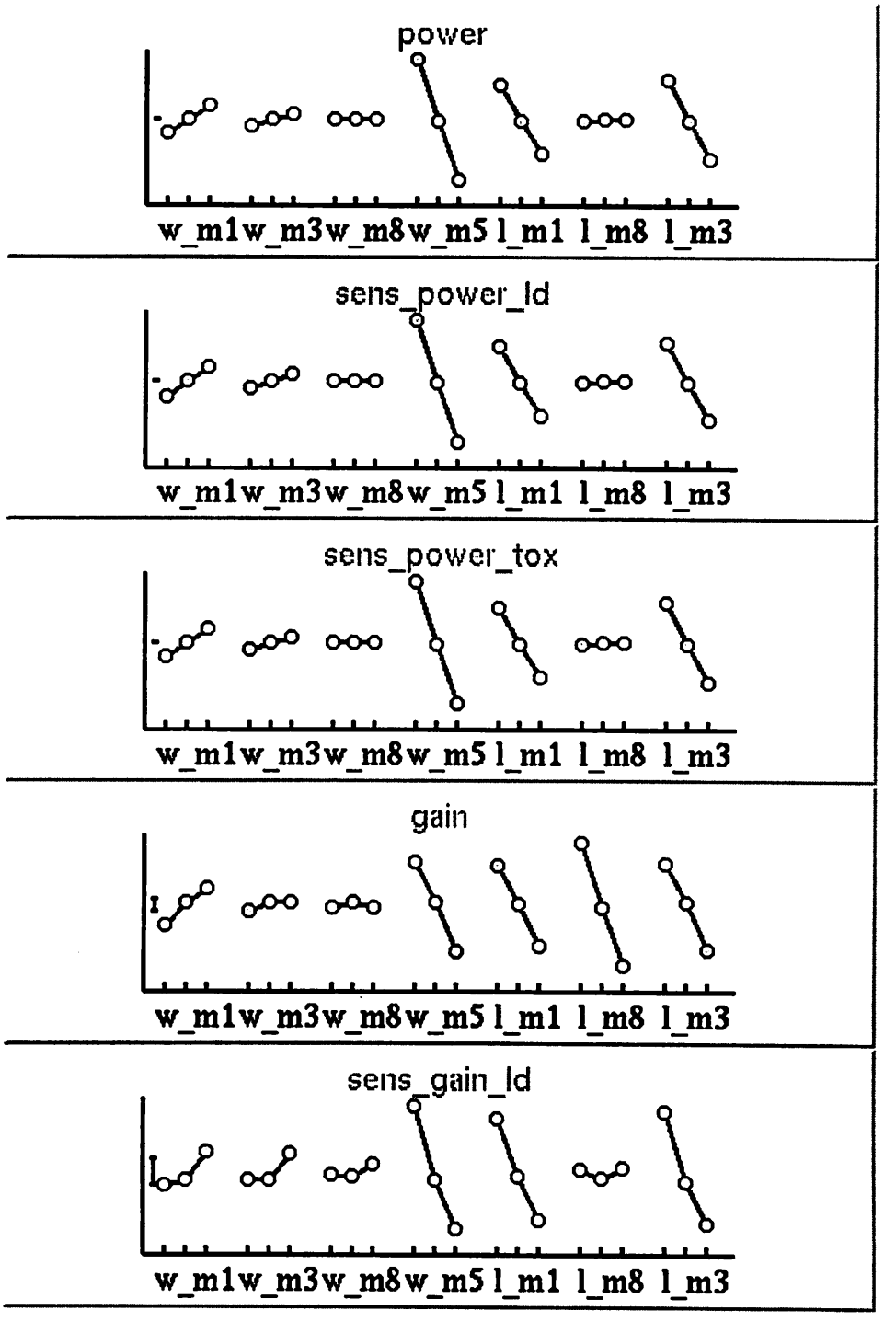
Figure 10 Factor-effect plots for example 2

A second iteration of this methodology is performed on the class AB amplifier, varying only the transistor sizes. Figure 11 shows the new set of factor-effect plots. The individual parameter settings to optimize one function at a time are summarized in Table 7.

Table 7: Optimal parameter settings (for class AB amplifier)

parameter	power	sens_P_ld	sens_P_tox	gain	sens_G_ld	sens_G_tox	bandwidth	optimal
w_m1	3	3	3	3	3	3	1	3
w_m3	3	3	3	2 or 3	3	-	1	3
w_m8	-	-	-	-	-	-	1	1
w_m5	1	1	1	1	1	1	3	1
l_m1	1	1	1	1	1	1	3	1
l_m8	-	-	-	1	-	1	1	1
l_m3	1	1	1	1	1	1	3	1

This window displays the factor effect values of all the parameters (with various levels) on all the performances. Click on a point to select one level for each parameter as input to a confirmation run.



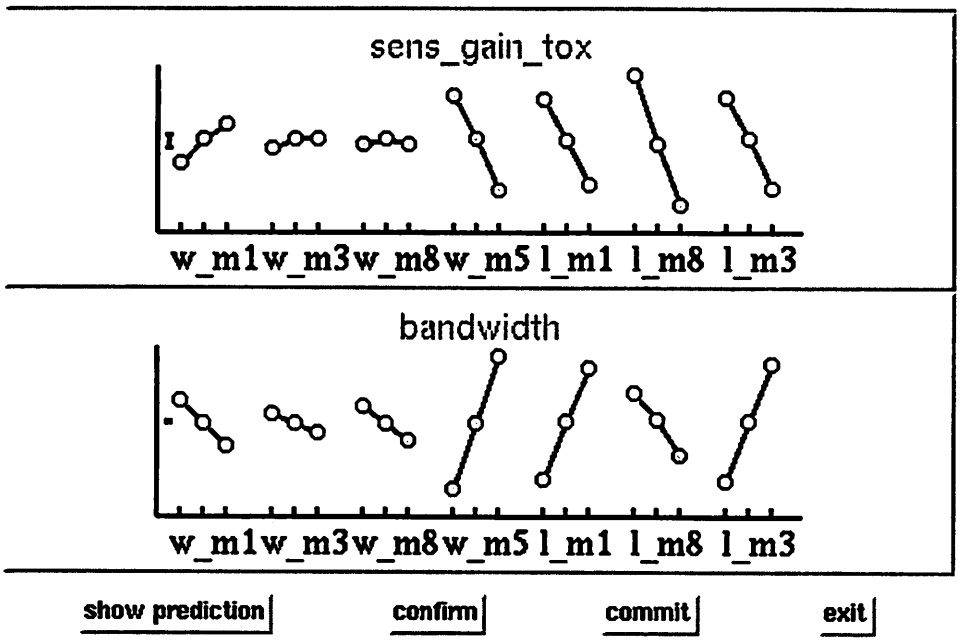


Figure 11 Factor-effect plots for example 2, second iteration

The results of the confirmation run at our chosen optimal point (3,3,1,1,1,1,1) are presented in Figure 12. We confirm that the difference between predicted and measured values is less than the acceptable error for each performance. Table 8 highlights the improvement of the chosen optimal design over the original hand-optimized design (2,2,2,2,2,2,2).

Table 8: Improvements for the class AB amplifier

	power (in μW)	sens_P_ld (in μW/ μm)	sens_P_tox (in mW/ mm)	gain	sens_G_ld (in /mm)	sens_G_tox (in /pm)	bandwidth (in MHz)
(2,2,2,2,2,2,2) original	36.2	458.3	$1.07\ 10^{-20}$	126.8	71.92	$1.01\ 10^{-20}$	3.71
(3,3,1,1,1,1,1) optimal	34.8	378.2	10^{-20}	128.6	73.0	10^{-20}	3.47
improvement	3.9%	17.5%	6.8%	1.5%	1.5%	0%	-6.4%

SELECTION 3 3 1 1 1 1 1

PERFORMANCE	PREDICTION	MEASUREMENT
Power	34.58558 db	34.58305 db
Sensitivity of Power to delta Ld	8.80666 db	8.80514 db
Sensitivity of Power to delta Tox	234.58558 db	234.58305 db
DC Gain	21.10282 db	21.09083 db
Sensitivity of Gain to delta Ld	-40.07495 db	-27.53883 db
Sensitivity of Gain to delta Tox	221.10282 db	221.09083 db
Bandwidth	65.40026 db	65.40791 db

Done

Figure 12 Predicted versus Measured performances at the optimal design

Chapter 5
Conclusions and Future Plans

In this report, the fundamentals of the Robust Design Method were discussed and its application to integrated circuit design was presented. DORIC was developed as the CAD tool to support circuit optimization based on the Robust Design Method. We showed how this experimental design method and the use of DORIC helps improve several manufacturability and performance characteristics of VLSI circuits.

The orthogonal experiment matrix, based on the additive model of factor effects, allows us to study a large number of decision variables with a much smaller number of experiments than with other statistical methods, or by trying all possible combinations of parameter settings. By examining the quality metric of output functions, a few sets of optimal parameter values emerge. Confirmation runs let the designer explore them and pick the best setting. This approach lets the designer take into account variations in both design parameters and manufacturing processes.

The Robust Design Method provides circuit designers with an efficient, simple and systematic way of improving their circuit performance, and increase robustness of their circuits to manu-

facturing variations. DORIC presents a user-friendly CAD environment that supports this methodology and summarizes the results in a powerful graphical representation.

Two practical examples illustrated the application of RDM to IC design optimization and showed the improvements obtained as a result of use of DORIC, over original hand-optimized designs. Some of the advantages of RDM are the use of a minimal number of simulation runs to explore a large design space, and the support of optimization over categorical (non-numerical) parameters, such as topology choices. Its main disadvantage is the assumption of additivity, which is tested at the last stage of the process. Several solutions were proposed to address this problem.

Several issues remain to be explored, some of which are highlighted next. The following points deal with methods for better evaluating the additivity assumption and obtaining a measure of the validity of the models:

• "Dummy level" technique: when parameter P is defined at p levels is assigned to an orthogonal array column C that can fit c levels, and $p < c$, the additional levels (p-c) are called "dummy" levels. They are usually assigned randomly to one of the p "real" levels of P. For example, if P has 2 levels (p1 and p2) and is fit in a column of 3 levels (c1, c2 and c3), the third level c3 is set for instance to p2. To calculate the factor-effect FE of parameter P set at level 2, the orthogonal array rows containing c2 are usually considered. In this case, the rows containing c3 are also mapped to level p2 and should ideally give a factor-effect FE' equal to FE, if all parameters were truly independent. A discrepancy between FE and FE' is an indication of the presence of interactions that need to be identified and quantified, to get a measure of the validity of the model.

• Orthogonal array with unused columns: there are two subtleties in this point. First, some orthogonal arrays are accompanied with interaction tables that identify which columns are confounded with the interaction of two other columns. Such an interaction-confounding column can strategically be picked to remain unused in order to obtain some measure of parameter interaction, and use it as a measure of the validity of the model. The second point is that any unused column of an orthogonal array (regardless of whether it is an interaction-confounding column) can be

viewed as representing a parameter where all the levels are equal. Thus, the "dummy level" technique can be applied to all the levels which should ideally give identical factor-effect values.

The next points relate to adding general constraints to the circuits and dealing with sub-circuit topology changes:

• Adding circuit constraints: It is currently possible to assign matching circuit devices to any parameter, such as two transistor sizes have to remain identical throughout the design search. Sizes of matching devices are treated as one parameter. This idea needs to be expanded to include not just equality between the sizes but also setting a constant ratio between devices sizes that are to be considered as one parameter. Taking this one step further, one can use this ratio as an additional design variable. Another form of device size constraints is specifying ratios $\frac{W}{L}$ for a given transistor.

• Sub-circuit topologies: although DORIC proved successful at taking topology choices into consideration, it has only been applied to an entire circuit topology change. A more useful feature would be to allow selective replacement of independent sub-circuit topologies. To make this transition seamless, additional functionality needs to be developed. Specifically the issue of identifying functionally equivalent circuit elements across topologies needs to be addressed.

We believe that DORIC sets the basis for a powerful methodology that will help bridge the gap between design and manufacturing and facilitate the creation of more robust designs, whether used as a stand-alone tool or integrated in a CAD framework.

References

[1] R. K. Brayton, G. D. Hechtel and A. L. Sangiovanni-Vincentelli, "A Survey of Optimization Techniques for Integrated Circuit Design," Proc. IEEE, Vol. 69, No. 10, pp 1334-1363, 1981.

[2] R. Spence and R. S. Soin, "Tolerance Design of Electronic Circuits," *Addison Wesley*, 1988.

[3] J. W. Bandler and H. L. Abdel-Malek, "Optimal centering, tolerancing, and yield determination via updated approximations and cuts," *IEEE Trans. Circuits Syst.*, vol. CAD-25, pp. 853-871, Oct. 1978.

[4] S. W. Director and G. D. Hachtel, "The simplicial approximation approach to design centering," *IEEE Trans. Circuits Syst.*, vol. CAS-24, pp. 363-372, July 1977

[5] S. W. Director and L. M. Vidigal, "Statistical Circuit Design: a somewhat biased survey," *Proc. Eur. Conf. Cct. Theory Design (ECCTD)*, The Hague, pp 15-24, 1981.

[6] R. Spence, L. Gefferth, A. I. Ilumoka, N. Maratos and R. S. Soin, "The Statistical Exploration Approach to Tolerance Design," *Proc. IEEE Int. Conf. Ccts. Computers*, New York, pp 582-585, 1980.

[7] W. Maly, A. J. Stojwas and S.W. Director, "VLSI Yield Prediction and Estimation: A Unified Framework," *IEEE Trans. on Computer-Aided Design*, vol. CAD-5, no.1, pp. 114-130, January 1986.

[8] P. Cox, P. Yang, S. S. Mahant-Shetti and P. Chatterjee, "Statistical Modeling of Efficient Parametric Yield Estimation of MOS VLSI Circuits," *IEEE Journal of Solid-State Circuits*, Vol. 20, No. 1, February 1985.

[9] E. D. Boskin and C. J. Spanos, "A Method for Modeling the Manufacturability of IC Designs," *Proceedings of IEEE Int. Conf. on Microelectronic Test Structures*, vol. 6, March 1993.

[10] T. K. Yu, S. M. Kang, I. N. Hajj and T. N. Trick, "iEDISON: An Interactive Statistical Design Tool for MOS VLSI Circuits," *Proceedings of Int. Conf. on Computer-Aided Design*, 1988

[11] S. R. Nassif, A. J. Stojwas and S. W. Director, "Fabrics II: A Statistically Based IC Fabrication Process Simulator," *IEEE Trans. on CAD*, Vol. 3, No. 1, pp 40-47, January 1984.

[12] D. A. Antoniadis, S. E. Hansen and R. W. Dutton, "SUPREM II - A Program for IC Process Modeling and Simulation," Report 5019-2, Stanford University, 1978.

[13] J. Sacks, W. J. Welch, T. J. Mitchell and H. P. Wynn, "Design and Analysis of Computer Experiments," *Statist. Sci.*, Vol. 4, pp 409-435, November 1989.

[14] Meta-Software, Inc., "HSPICE Users Manual H9001," 1990.

[15] G. E. P. Box, W. G. Hunter and J. S. Hunter, "Statistics for Experimenters," John Wiley and Sons Inc., 1978.

[16] C. R. Rao, "Factorial Experiments Derivable from Combinatorial Arrangements of Arrays," *Journal of Royal Statistical Society*, Series B, Vol. 9, pp 128-139, 1947.

[17] M. S. Phadke, R. N. Kackar, D. V. Speeney and M. J. Grieco, "Off-Line Quality Control in Integrated Circuit Fabrication Using Experimental Design," *The Bell System Technical*

Journal, Vol. 62, No. 5, pp 1273-1309, May-June 1983.

[18] M. S. Phadke, "Quality Engineering using Robust Design," *Prentice Hall*, AT&T Bell Laboratories, 1989.

[19] M. C. Bernardo, R. Buck, L. Liu, W. A. Nazaret, J. Sacks and W. J. Welch, "Integrated Circuit Design Optimization Using a Sequential Strategy," *IEEE Trans. on CAD*, Vol. 2, No. 3, March 1992.

[20] Z. Daoud and C. J. Spanos, "Using Stochastic Functions for Modeling Computer-Based Experiments," Special Issues in Semiconductor Manufacturing, Memorandum No. UCB/ ERL M92/84, chapter 8, August 1992.

[21] W. J. Welch, T. K. Yu, S. M. Kang and J. Sacks, "Computer experiments for quality control by parameter design," Journal of Quality Technology, Vol. 22, No. 1, pp 15-22, 1990.

Appendix I

Orthogonal Array L_{18}

Table 1: Experiment Matrix L_{18}

trial	topology	width_out	length_out	width_in
1	1	1	1	1
2	1	1	2	2
3	1	1	3	3
4	1	2	1	1
5	1	2	2	2
6	1	2	3	3
7	1	3	1	2
8	1	3	2	3
9	1	3	3	1
10	2	1	1	3
11	2	1	2	1
12	2	1	3	2
13	2	2	1	2
14	2	2	2	3
15	2	2	3	1
16	2	3	1	3
17	2	3	2	1
18	2	3	3	2

Appendix II

Quality metric definitions

Quality metrics (QM) are a measure of the quality of the performance. They are defined such that the QM is always maximized when the performance is optimized. For example, a smaller delay value is more desirable, so the QM is a function of the inverse of the delay.

$$QM_{speed} = 10 \times \log\left(\frac{1}{\text{nominal delay}}\right)$$

$$QM_{area} = 10 \times \log\left(\frac{1}{\text{area}}\right)$$

$$QM_{power} = 10 \times \log\left(\frac{1}{\text{power}}\right)$$

$$QM_{gain} = 10 \times \log(\text{gain})$$

$$QM_{bandwidth} = 10 \times \log(\text{bandwidth})$$

$$QM_{phase\ margin} = 10 \times \log(\text{phase margin})$$

To calculate the quality metric of the sensitivity of speed to variations in the thickness of the oxide (Tox) or the effective channel length (Ld), first the sensitivity is defined as the ratio of the change in delay to the change in Tox (or Ld).

$$\text{sensitivity to tox} = \frac{\text{tox delay} - \text{nominal delay}}{\text{change in tox}}$$

Then QM of sensitivity of speed to variations in tox (or ld) is:

$$QM_{\text{sensitivity of speed to tox}} = \log\left(\frac{\text{speed}}{\text{sens to tox}}\right)$$

Similarly, to define the sensitivity of a performance to variations in the Tox or Ld, first the sensitivity to Tox or Ld is defined as the ratio of the change in the performance over the change in

Tox (or Ld). Then the QM is defined with the performance to maximize on the numerator and the sensitivity to minimize on the denominator.

Appendix III

DORIC's software low-level design

There are 3 main parts to DORIC's system architecture (shown in Figure 1): the front end graphical user interface, implemented in Tcl/Tk, the core implemented in C, and the back end graphical user interface in Tcl/Tk.

From a functional point of view, the front end inputs the user problem definition, selects the appropriate orthogonal array to use, and submits the job to the core. The core creates the SPICE wirelists according to the orthogonal array's specifications, submits them to HSPICE, collects the results of HSPICE, performs the analysis to come up with models and factor effect plots, and returns to the front end. The back end is then invoked to display the factor effect plots and execute the user specified confirmation runs. The pseudo-code below illustrates how the program control starts with the front end GUI, who calls the C-core and then calls the back end GUI. Once the back end is invoked, the control is relinquished from the front end to the back end GUI, in order to support event driven programs. The back end GUI is then user driven to call specific portions of the C code to execute the confirmation runs.

- DORIC front end (Tcl/Tk)

• creates a menu to help the user define the problem: reads in the spicefile name(s), the performance(s) to optimize and the parameters to vary.

• when the user is done defining the problem and wants to submit it to the core, the number of simulation runs is estimated by selecting the appropriate orthogonal array and the user is prompted to acknowledge the problem definition. The results of the definition phase are written into a file ("Inputfile").

• the problem is submitted to the core (to create_model.c), in the form of the definition file created above ("Inputfile"). When the core program is done, it returns the factor effect plots.

• factor effect plots are submitted to the back end GUI (mkPlot.tcl) and the control is relinquished to it.

• exit.

Table 2: DORIC front end menu choices

menu choice	actions	calls
SPICEFILES	Specify the number of circuits to optimize and the names of the corresponding spicefiles.	mkTopology.tcl
PARAMETERS	Identify varying parameters (sizes of transistors, resistors and capacitors), the incremental values and their matching devices (if any)	mkSize.tcl mkLevels.tcl
PERFORMANCES	Specify performances of interest among a given menu of traditional performances (speed, power) and sensitivity performances (sensitivity of speed to changes in tox)	mkPerformance.tcl
LAUNCH	Submit the problem to the core program, and when that returns, it calls up the back end for display of factor effect plots.	mkInputConfirm.tcl mkFileInterface.tcl create_model.c mkPlot.tcl
CANCEL	Exit from the entire program.	
HELP	Additional information	

- DORIC back end (Tcl/tk)

• displays the factor effect plots and waits for the user input in event-driven mode. Depending on the user input, it calls back_end.c procedures to execute various actions: predict, confirm, commit or exit.

• program is exited when the user presses on "commit" or "exit". Commit creates a final design (spicefile) reflecting the user's choice, then exits, and the "exit" button simply exits.

The back end GUI calls mkSelection.tcl to read the input the user has highlighted by clicking on the screen. It then call back_end.c with a different code for each action (code predict, code commit, etc.). Back_end.c shares many of the procedures of create_model.c (such as parsing spicefile inputs and outputs, etc.)

Table 3: DORIC's back end menu choices

button	action
PREDICT	Get the predicted values for every performance based on the models previously derived by the Robust Design Method.
CONFIRM	Creates a spicefile, runs it, gathers the actual values of performances, and compares then to the predicted values.
COMMIT	Create a final design reflecting the user choices and exit.
EXIT	Exit from the entire program.

- Core program (in C)

There are two parts in the core program that share many procedures. The first part is create_-model.c is the one invoked from the front end in order to actually apply the Robust Design Method implementation (run spice, create models, generate factor effect plot, etc.) The second part is back_end.c is called from the back end GUI menus to predict, confirm or commit a design choice. A listing of the procedures invoked by each part follows:

Table 4: Contents of the core program

	calls	actions
create_ model.c	input.c	reads the problem definition file and creates the data structures needed for the rest of the program
	matrix_column_assign.c	assign parameters to the columns of the orthogonal array being used.
create_ model.c and back_ end.c	generate_spicefiles.c	creates all the spicefiles by parsing and modifying the original spicefiles to reflect the orthogonal array entry levels for each parameter. Also create the tox and ld versions of the same files if needed.
	run_spice.c	submits all the spicefiles to HSPICE.
	read_spice_ output.c	parses HSPICE's output files (.mt0 and ma0 for transient and ac analysis) looking for raw performances values.
	calculate_SN.c	calculates signal-to-noise ratios (i.e. quality metric) for every performance at every run.
create_ model.c	calculate_factor-effects.c	calculates factor-effect plots of every parameter at every level, on each performance.
	error_calculation.c	calculates the root-mean squared error for every performance model. The accepted error is (as a heuristic) three times the r.m.s.
	output.c	creates output files containing the factor-effect plots.
back_ end.c	setup_confirmation.c	reads the user selected input from the plots and sets up a one-row confirmation matrix similar to the orthogonal array.

Appendix IV

File formats

No comments, no empty lines are allowed.

- Inputfile: problem definition

- Format: line 1: n parameters
- The next n blocks are the descriptions of the n parameters:

the parameter P is a transistor width, length and a resistor or a capacitor size. Each description contains: on the first line, the parameter name (width, length, resistor, capacitor), followed by the numbers i of incremental values this parameter can take on, followed by its name. On the next i lines are the i actual incremental values. The last line of this description specify matching devices and is of the form "matchings <d1> <d2> ..." where d1, d2, etc. are matching devices to parameter P.

- next is "topology t" where t is the number of topology choices. The next t lines are the names of the t spicefiles.

CAUTION: if the topology has more than 1 choice, it is then considered a varying parameter and the number of parameters specified on line 1 must be incremented by one!

- p performances where p is the number of performances and the next p lines are the names of the performances.
- then is the line: "file xxx" where xxx is the name of the file containing the appropriate orthogonal array, followed by a line containing the number of rows in that array.

EXAMPLE:

3 parameters
width 3 m1
-1
0

1
matching m2
length 2 m1
-1
1
matching m2
topology 1
ab.sp
ota731.sp
3 performances
Power
Sens_Power_Ld
Gain
file: 4.3
9 rows

- SN (signal-to-noise) file: factor-effect plots

• line 1 contains only one number n: the number of performances. The next n lines have the name of the performances, one performance per line.

• then 1 number p: the number of parameters. The next p statements are formed of 2 lines each: 1 line for the name of the parameter, the other line for the number of levels L this parameter can take on.

• one line containing: "SN values" followed by n groups of numbers, separated by a line containing a 0. Each group G defines the factor-effect plot for a given performance.

• Each group G is formed of p subgroups (each subgroup represents the factor effect of a given parameter on that performance), separated by a line containing a 0.

• Each subgroup is formed of L values where L is the number of levels the given parameter can take on. The only exception to that is the first subgroup in each group: it contains L+1 numbers, where the first number is the root mean squared error (which will be used to create the error bar on the plot for that given performance) and the next L values are the actual factor-effect values of that parameter at that level on that performance.

EXAMPLE:

2 -> 2 performances: power and sens_power_ld
power
sens_power_ld
2 -> 2 parameters w_m1 and w_m3 at 3 levels each
w_m1

3
w_m3
3
SN VALUES:
6.481935e-04 -> value of the r.m.s for the power
-4.187336e-02 -> factor effect of w_m1 set at level 1 on the power
6.632154e-04
4.121015e-02
0
-1.977620e-02 -> factor effect of w_m3 set at level 1 on the power
6.134280e-04
1.916277e-02
0
7.239526e-04 -> value of the r.m.s for the sens_power_ld
-6.692613e-03 -> factor effect of w_m1 set at level 1 on
-2.221450e-04 sens_power_ld
6.914758e-03
0
-2.783204e-03
7.154148e-04
2.067789e-03 -> factor effect of w_m3 set at level 3 on
0 sens_power_ld

Appendix V

Data structures (refer to experm.h)

All data structures are dynamically allocated because, either it is not known until run-time the number of elements in the structure, or the structure needs to be saved after the procedure has exited (and therefore cannot be a local variable).

- **Performances:** There is an enumerated type "outputs" which contains the names of the performances P. The array of integers "Performance" is set to 1 if Performance[P] has been selected by the user, and 0 otherwise. For example, if the user selected "Gain" then Performance[Gain] = 1. The size of the array is MAX_NUM_OUTPUTS. Although the size is fixed before run-time, this structure is allocated because it is needed later on in the program (see "record of pointers").

-Parameters: "Param" is a dynamically allocated array of structures "parameters". The size of the array is num_param. Each structure contains:

- the name of the factor (which is a string containing "resistor", "capacitor", "width" or "length");
- the device name which is a string containing the unique HSPICE device name ("R1", "m2", etc.);
- the number of the orthogonal array column that this parameter is assigned to;
- the actual number of level settings that the parameter is defined to take on ("real_num_-levels");
- the number of levels that the column assigned to this parameter can take on (for example if a parameter P which has 2 levels is assigned to a column that can take up to 3 levels, P's real_num_levels = 2, but its num_levels = 3. Dummy levels (such as the third level in this example, are assigned equal to the last real level; so P's dummy level 3 is equal to its real level 2);
- a dynamically allocated array of doubles (of size num_levels) containing the level values that the parameter takes on;
- a dynamically allocated array of strings containing the names of the devices that must be matched to this parameter.

- **Spicefile name(s):** "Spicefiles" is an array of strings containing the name of the original HSPICE input files. At least one, maybe more filenames can be specified.

- **Output of HSPICE: raw data:** a dynamically allocated array of structures containing raw data information gathered from the output of HSPICE. Each structure contains:

  - the name of the HSPICE output ("delay1", "delay2", "power", "gain", etc.);
  - the performance value at the nominal tox and ld;
  - the performance at incremented tox (but nominal ld);
  - the performance at incremented ld (but nominal tox).

- **Signal-to-Noise ratios (SN):** a dynamically allocated array of structures containing the SN values of the performances. The size of the array is equal to the actual number of performances the user has selected, one structure per performance selected. Each structure contains:

  - the name of the performance;
  - an array of n double values (where n is the number of experiments), the i-th value is the signal-to-noise ratio of this performance for experiment i, as calculated directly from the output of HSPICE;
  - an array of n double values where the i-th value is the difference between the SN_value (defined above) and the predicted value (derived from the model) at the i-th experiment;
  - a double value equal to the root mean squared error of the n experiments (calculated as the average of the square of the n error values (defined above))

- **Factor Effect values (FE):** a dynamically allocated array of structures containing the factor-effect values for every performance. The size of the array is the number of performances the user selected. Each structure contains:

  - the name of the performance;
  - the mean of the factor effect values of all parameters at all levels on that performance;
  - a pointer to an array of p structures (where p is the number of parameters), one structure per parameter. Each of these structures contains the name of the parameter and an array of L factor-effect values (where L is the number of levels this parameter takes on) of that parameter on that performance.

- **Column mapping:** a dynamically allocated array of structures to represent information regarding the columns of the orthogonal array. The size of the array is the number of columns in the orthogonal array. Each structure contains:

• the number of levels that are used for "real" parameter levels;

• the maximum number of levels this column can support (the difference between the "real" levels and the max number of levels is by definition the number of "dummy" levels);

• an identification of the parameter that is assigned to this column, which is the parameter's index in the "param" array.

- **Delay signal names:** Currently, a delay/speed performance is defined as the average of the risetime and the falltime. (I recommend that it be changed to simply be the risetime or falltime at the user's wish). So each delay signal is identified by two waveform names, so d delay signals require 2*d waveform names. The delay signal names array contains the names of the 2*d waveform names in order to identify them when reading HSPICE's output.

- **a record of pointers:** A number of data structures need to be shared between the Core program part 1 (create_model.c) and part 2 (back_end.c). The way these structures get passed through the Tcl script is by encapsulating them in a record of pointers to these structures, that gets passed as Tcl-specific "ClientData". The data structures created in create_model.c that get passed to back_end.c, through this "ClientData" scheme are:

• number of user-specified factors;
• number of user-specified performances;
• number of columns in the orthogonal array;
• number of pairs of delay signals;
• number of experiments (which equals the number of rows of the orthogonal array);
• pointer to the "Performance" array;
• pointer to the "Spicefile names" array;
• pointer to the "delay_names" array;
• pointer to the "Parameters" array;
• pointer to the "factor-effect" array.