IC DESIGN FOR MANUFACTURABILITY I

by

Professor: Costas J. Spanos

Students:
Eric Boskin, Raymond Chen, Zeina Daoud,
and Hao-Cheng Liu

Memorandum No. UCB/ERL M92/17

5 February 1992

IC DESIGN FOR MANUFACTURABILITY I

by

Professor: Costas J. Spanos

Students:
Eric Boskin, Raymond Chen, Zeina Daoud,
and Hao-Cheng Liu

Memorandum No. UCB/ERL M92/17

5 February 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

Preface

This report contains three projects that aim at improving our understanding of IC design for manufacturability. These projects were completed by students in the Berkeley Computer-Aided manufacturing (BCAM) group, in the context of EE219 and EE244, two graduate Computer-Aided Design courses. These courses were given by professor Sangiovanni-Vinc. (219), professor Brayton (219) and Dr. L. Sheffer (244) in the fall of 1991.

The objective of the first project, entitled "Computation of Process Parameter Sensitivity Using Spice", is to incorporate the evaluation of IC performance sensitivities within Spice. These sensitivity calculations are to be used in conjunction with an experimental circuit design technique that might lead to circuits with reduced sensitivity to production, as well as to environmental variations. The technique presented here has been demonstrated to work with only about a 10% computational penalty over the standard cost of Spice simulation.

The second project, entitled "Application of the Robust Design Method to IC Design for Manufacturability" focuses on the novel use of experimental design techniques towards improving the manufacturability of an IC standard cell. The technique used is the "orthogonal array" method introduced by Genichi Taguchi as an economical method of experimentation for the improvement of industrial processes. Several novel ideas have been introduced in this project that aim to adapt this method for application on computer experiments using Spice.

The third project, entitled "A Fuzzy Evaluator for Technology Mapping" focuses on the optimal selection of off-the-shelf EPLD technology for the mapping of simple IC functions. The objective of this project was two-fold: the first objective was to introduce a useful automated tool to perform this selection. The second objective was to experiment with fuzzy set theory for the representation of the expert knowledge that must be employed in the course of this technology mapping.

These projects represent a sampling of the research within the Berkeley Computer-Aided manufacturing group. Our purpose is to blur the traditional boundaries between IC design and production, in order to streamline the profitable introduction of new IC designs to be produced on state of the art technologies.

Costas J. Spanos

February 3, 1992

Table of Contents

Computation of Process Parameter Sensitivity Using SPICE

Eric D. Boskin

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

December 17, 1991

Abstract

The variations in process parameters seen on a modern semiconductor fabrication line cause a decrease in the yield of manufactured ICs. As the dimensions of transistors continue to shrink, the problem grows as the variation becomes a larger percentage of the feature size. Although researchers and manufacturers are developing improved methods of process control to directly reduce the variation, techniques to decrease the sensitivity of new IC designs to process variations have not been widely used. This project investigates the computation of time domain sensitivities of MOS circuits to process parameter variation.

The computation of time domain sensitivities during circuit simulation has been well established. The sensitivities to design parameters, notably the widths and lengths of MOSFETs in MOS circuits, has been calculated for use in circuit optimization. Here, the time domain sensitivities to process parameters such as oxide thickness and effective channel length are considered. The direct differentiation approach is used to calculate the sensitivities concurrently with a transient simulation in SPICE. At each time step, the linearized Jacobian is used to solve for the unknown vector of node voltage and branch current sensitivities. For a simple test circuit, good agreement is found for the output voltage sensitivity as calculated using SPICE and simple perturbation. The sensitivity information is intended for use in investigating the manufacturability of MOS circuit designs.

1.0 Introduction

The transient sensitivity of MOS circuits using direct differentiation has been extensively studied [1,2]. The use of the sensitivity information was to guide circuit optimization, and therefore the sensitivities were calculated with respect to circuit design parameters, such as MOS transistor length and width [3]. However, the calculation of sensitivity to process parameter variation has been relatively neglected. In this project, the calculation of the transient sensitivity of MOS circuits to process parameter variation using the circuit simulator SPICE3e2, is examined. Examples of process parameters include gate oxide thickness (Tox) and transistor channel length shrink (known as ΔL, or LD in the SPICE model). It has been well established that the variation in these processing parameters causes performance fluctuations in manufactured ICs [4].

The intended application of precess sensitivity is for use in a manufacturability metric for use by VLSI circuit designers. Process variation is the cause of the performance variation seen in high volume VLSI circuit manufacturing. Understanding the sensitivity of circuits to process variation early in the design phase will allow designers to choose circuit topologies and/or transistor sizes which minimize process sensitivity while still meeting performance goals.

This document is organized as follows: in section 2, an overview of sensitivity analysis is presented. In section 3, the process parameter sensitivity calculation is explicitly shown for a simple MOS circuit. Next, results are presented for the sensitivity calculation. More specifically, the results from a modification of SPICE3e2 are compared with a perturbation analysis of the same circuit. In section 5, the development of circuit level sensitivities from node voltage and branch currents will be discussed. Section 6 discusses the application for process sensitivity in a circuit manufacturability metric. Finally, in section 7, conclusions and future work will be presented.

2.0 Overview of Sensitivity Analysis

This section follows the work done by Hocevar, *et al* [1] and Choudhury [2]. In general, the circuit simulation problem can be expressed as the solution to the system of equations:

$$f(x(p,t), \dot{z}(x(p,t)), p, t) = 0 \tag{1}$$

where $x(t)$ is the vector of all node voltages and Modified Nodal Analysis (MNA) mandated currents, z is the vector of capacitor charges and inductor fluxes, p is the vector of process parameters, such as Tox and LD, and t is time.

2.1 Adjoint Approach

Early work in sensitivity analysis was done using the adjoint network approach [5,6]. Using this method, an "adjoint network" is found, whose node voltages and branch currents represent the sensitivities of the original circuit's nodes and branches. The problem with using the adjoint approach for transient circuit sensitivity is that the circuit must be solved backwards in time, and therefore the time steps used in the numerical solution of (1) by SPICE may not be appropriate for the solution of the adjoint network. Using different time steps for the solution of the system would basically increase the simulation time by 100% as opposed to the 10% penalty in using the linearized Jacobian while retaining the time steps of the transient solution [1,3].

The advantage of the adjoint approach is that the solution of the adjoint network results in a vector which is the sensitivities of a given node voltage (or MNA appended branch current) to all design parameters. As will be shown later, the direct method results in a vector which is the sensitivity of all node voltages (and branch currents) to one design parameter. Therefore, if the sensitivity of a few key nodes is desired with respect to several process parameters, then the adjoint again becomes a viable solution.

2.2 Direct Differentiation

Direct differentiation has been the preferred approach to sensitivity analysis, due to its ease of implementation in existing circuit simulators [1,2]. The general circuit equation in (1) is differentiated with respect to the process parameters:

$$F_{\dot{z}}\frac{d\dot{z}}{dt} + F_x x_p + F_p = 0 \tag{2}$$

where F_x is the Jacobian matrix used to solve (1) [2]. However, a new right hand side (RHS) is used to find the branch (and node) sensitivities, x_p. The new RHS includes a term at each branch for any component connected to that branch which has a sensitivity to the process parameter being investigated. These terms become the driving sources of the sensitivity circuit.

3.0 Calculation of Process Parameter Sensitivity

The calculation of the sensitivity equations will be demonstrated through the use of a simple circuit, a CMOS inverter. The inverter is shown in Figure 1. For this analysis, the Level 1 SPICE model will be used and the MOS threshold voltage (Vt) will be assumed constant. The parasitic capacitances which are important in this situation are the gate to drain capacitance, Cgd, because of the Miller effect, and the drain junction capacitance, because the charge storage on that node effects switching time. Note that the gate to source capacitance Cgs, is unimportant for the sensitivity analysis, because the sources are tied to

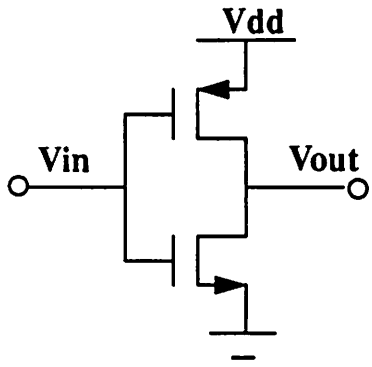


Figure 1. CMOS Inverter

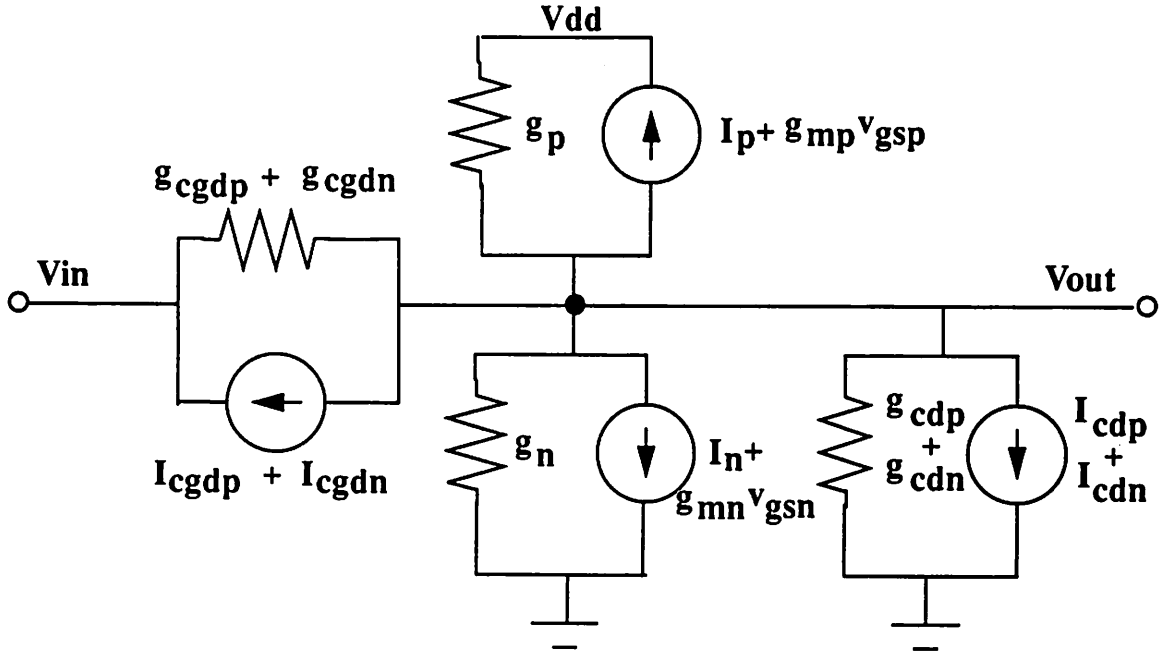


Figure 2. Companion Circuit Including Parasitics Cgd and Cdb

Vdd or Ground, and the input is a voltage source with unlimited current drive. The companion circuit for the inverter including parasitic capacitors is shown in Figure 2.

The MNA equations for this circuit are given in (3). The variables are identified in Figure 2. Note that a subscript starting with 'c' implies the term is due to a parasitic capacitor, followed by a *d* if it is at the drain, or else a *gd* if it between the gate and drain, and that the last letter in the subscript will be a *p* or an *n* to identify which transistor it is associated with. The conductances of the capacitors will be a function of their charge, q.

$$\begin{bmatrix} g_p+g_{mp} & -g_p & -g_{mp} & -1 & 0 \\ -g_p-g_{mp} & g_n+g_p+g_{cdp}+g_{cdn} & g_{mn}+g_{mp} & 0 & 0 \\ 0 & g_{cgdp}+g_{cgdn} & -g_{cgdp}-g_{cgdn} & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} V_{dd} \\ V_{out} \\ V_{in} \\ i_{vdd} \\ i_{vin} \end{bmatrix} = \begin{bmatrix} I_{totp} \\ -I_{tot}-I_{cd}-I_{cgd} \\ I_{cgd} \\ V_{dd} \\ V_{in} \end{bmatrix} \tag{3}$$

where the g_x's and the I_x's are the linearized companion model parameters, and:

$$I_{totp} = I_p^k - g_{mp}v^k{}_{gsp} - g_p v^k{}_{dsp} \tag{4}$$

$$I_{totn} = I_n^k - g_{mn}v^k{}_{gsn} - g_n v^k{}_{dsn} \tag{5}$$

$$I_{tot} = I_{totn} + I_{totp} \tag{6}$$

$$I_{cd} = I_{cdp}^k + I^k{}_{cdn} \tag{7}$$

$$I_{cdg} = I_{cdgp}^k + I^k{}_{cdgn} \tag{8}$$

and a superscript '*k*' denotes the value from the previous iteration.

To find the sensitivity equations, the derivative of the above system of equations is taken with respect to the process parameter of interest, for example, Tox. When differentiated with respect to process parameters, the entire RHS of the original system becomes zero. This is because the original sources

(including linearized sources) are constants with respect to the process parameter. For each term in the Jacobian, however, the derivative is given by the chain rule. For example:

$$\frac{\partial}{\partial Tox}(g_p V_{out}) = (g_p)\frac{dV_{out}}{dT_{ox}} + \frac{dg_p}{dTox}(V_{out}) \tag{9}$$

Note that the first term of the derivative in (9) is the same Jacobian element (g_p) times the desired sensitivity, and the second term will become a driving source of the sensitivity equations.

As has been shown in [1,2] and can be seen in the above equation, the sensitivity equations utilize the same linearized Jacobian. The vector of unknowns contains the desired sensitivities, and a new RHS is generated, consisting of the sensitivities of the circuit components to the process parameters. The new system of equations is:

$$\begin{bmatrix} g_p + g_{mp} & -g_p & -g_{mp} & -1 & 0 \\ -g_p - g_{mp} & g_n + g_p + g_{cdp} + g_{cdn} & g_{mn} + g_{mp} & 0 & 0 \\ 0 & g_{cgdp} + g_{cgdn} & -g_{cgdp} - g_{cgdn} & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{dV_{dd}}{dT_{ox}} \\ \frac{dV_{out}}{dT_{ox}} \\ \frac{dV_{in}}{dT_{ox}} \\ \frac{di_{vdd}}{dT_{ox}} \\ \frac{di_{vin}}{dT_{ox}} \end{bmatrix} = \begin{bmatrix} I'_{totp} \\ -I'_{tot} - I'_{cd} - I'_{cgd} \\ I'_{cgd} \\ 0 \\ 0 \end{bmatrix} \tag{10}$$

where:

$$I'_{totp} = \frac{dg_p}{dT_{ox}}(V_{out} - V_{dd}) + \frac{dg_{mp}}{dT_{ox}}(V_{in} - V_{dd}) \tag{11}$$

$$I'_{totn} = \frac{dg_n}{dT_{ox}}(V_{out}) + \frac{dg_{mn}}{dT_{ox}}(V_{in}) \tag{12}$$

$$I'_{tot} = I'_{totn} + I'_{totp} \tag{13}$$

$$I'_{cd} = \frac{dg_{cd}}{dT_{ox}}(V_{out}) \tag{14}$$

$$I'_{cgd} = \frac{dg_{cgd}}{dT_{ox}}(V_{out} - V_{in}) \tag{15}$$

The calculation of the new RHS vector will be discussed in section 4.1, along with the implementation of this scheme.

4.0 Transient Sensitivity Implementation

To verify the correctness of these results, SPICE3e2 was modified. At each time step, the linearized Jacobian was solved with a new RHS vector. The SPICE modifications were not general purpose, but only sufficient for this simple test case. Unix shell scripts were written to process raw output. These results will be compared with the result of a simple perturbation analysis. The perturbation result was found by running SPICE a second time with a perturbed value for the oxide thickness in the SPICE model, and calculating the sensitivity at each time step using finite difference:

$$Sensitivity_{Perturbation} = \frac{(V_{out} - V_{out_{perturbed}})}{(T_{ox} - T_{ox_{perturbed}})} \tag{16}$$

The oxide thickness was perturbed by a value that was large enough to produce a change in performance readily extracted above the numerical noise of the simulator, but small enough that the effect of the perturbation on the device characteristic is linear. An important comment on perturbing process parameters in SPICE models in given in section 4.4.

4.1 SPICE3e2 Implementation

The RHS vector for the sensitivity equations were found by differentiating the SPICE Level 1 equations. For the transistors:

$$g_x = \frac{W}{L} C_{ox} \times f(V_{gs}, V_{ds}, V_t) \tag{17}$$

and for the capacitors:

$$g_x = \frac{2C_{ox}}{h} \tag{18}$$

that is, both are linear functions of Cox. Since Cox = εox/tox. Therefore:

$$\frac{d}{dT_{ox}}(C_{ox}) = -\frac{\varepsilon_{ox}}{t_{ox}^2} = -\frac{C_{ox}}{t_{ox}} \tag{19}$$

and

$$\frac{d}{dT_{ox}}(g_x) = -\frac{g_x}{t_{ox}} \tag{20}$$

Therefore, for example:

$$I'_{totn} = \frac{dg_n}{dT_{ox}}(V_{out}) + \frac{dg_{mn}}{dT_{ox}}(V_{in}) = (-\frac{1}{t_{ox}})\,(g_n V_{out} + g_{mn} V_{in}) = -\frac{I_n}{t_{ox}} \tag{21}$$

that is, the terms in the new RHS vector are the total currents through each device found in the solution for the current time step divided by the oxide thickness.

SPICE was modified to create this new RHS vector for this specific example circuit. As the parasitic capacitors play an important role in the analysis, a simplified version of the piecewise linear Meyer parasitic capacitance model was used [7].

4.2 Comparison of Results

Figure 3 shows the results of the perturbation calculation and the SPICE sensitivity calculation. The comparison is done for the sensitivity of the output node voltage to variations in Tox. A pulse is applied at the input of the inverter, as shown. In general, good agreement can be seen between the two waveforms. The mismatch between the two results is mainly due to the simplifications used in the Meyer capacitance model introduced in the calculation of the new RHS in SPICE.

4.3 Generalizing the RHS Vector Calculation

For this project, symbolic differentiation of the active device equations was used to generate the required partial derivatives of the Jacobian. As discussed by Choudhury [1], this introduces a model dependence which can be eliminated by using perturbation inside of SPICE to calculate. Finite difference can be used accurately at each time step to compute the derivative using finite difference. Very small perturbation factors can be used, because the derivative can be computed to the accuracy of the floating point representation. One key difference here is that *every* instance of a device must be modified when computing the derivatives, unlike the case when you are finding the derivative with respect to a specific transistor channel width or length.

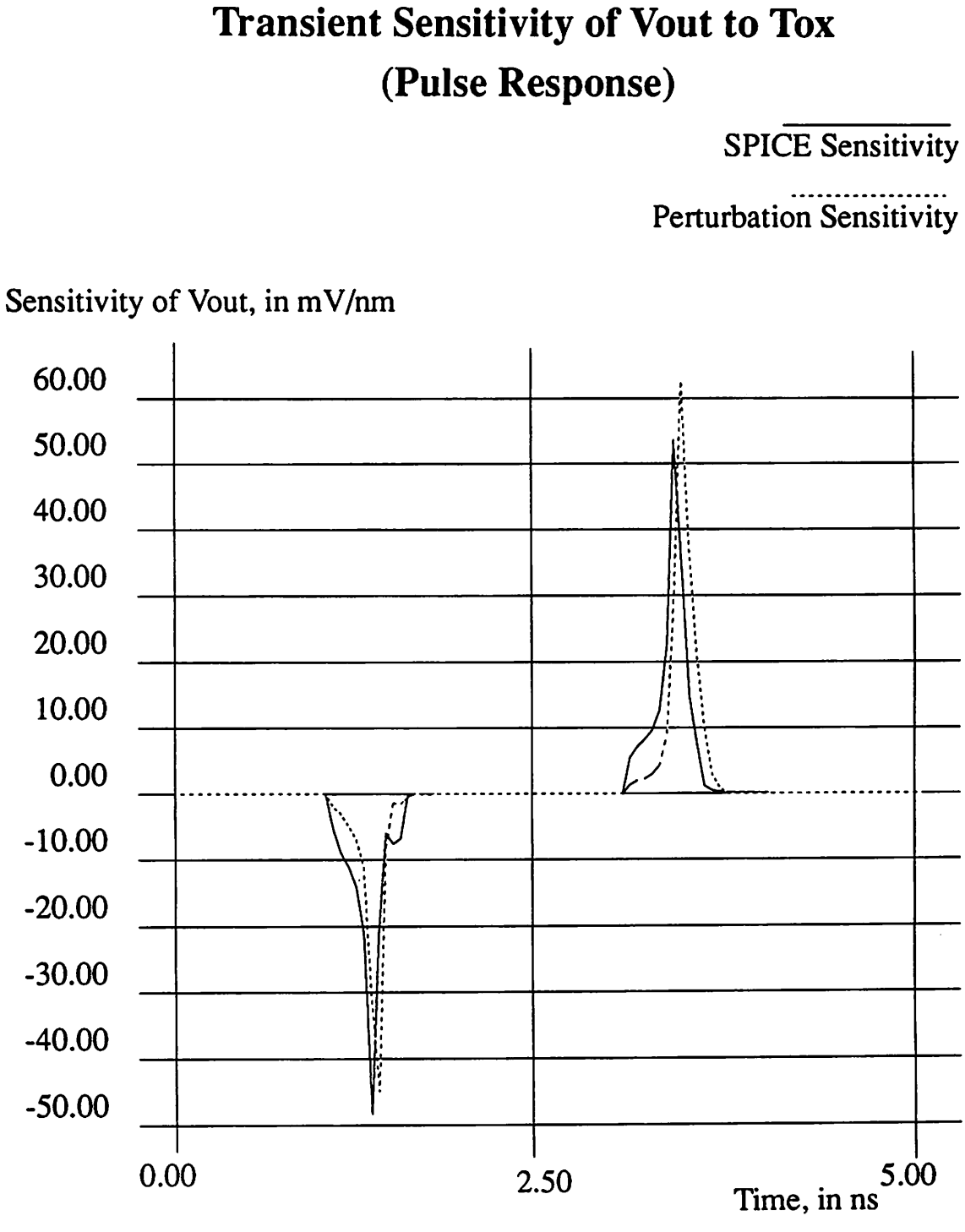


Figure 3. Comparison of SPICE and Perturbation Results

4.4 SPICE Model Considerations

In order for the perturbation results to be accurate, there are several important SPICE model considerations which must be taken into account. Most importantly, there are high level model parameters, such as the device transconductance, KP, which are dependent on Tox, which must be explicitly recalculated in order for any perturbation of Tox to accurately reflect the device performance change. Such a model has been conceived for statistical circuit simulation [8].

A general, physically based device modeling scheme has been developed by the author for use in statistical modeling and manufacturability evaluation [9]. This model would work well for the calculation of process sensitivities. In a physically based model, the physical SPICE model parameters, such as Tox, are set to their measured value. Parameters such as ETA and KAPPA are fitted to create an accurate model, but parameters such as KP are omitted, so that the performance variation caused by process variation (in Tox or LD, for example) are accurately modeled by changing the corresponding physical model parameter.

5.0 Circuit Level Sensitivities

In order to apply the sensitivity information to the design of integrated circuits, circuit performance measures such as delay, switching time, and power need to be considered. In order to optimize a design, the sensitivity to performance must be computed. The performance criteria are usually related directly to node voltage or branch current sensitivities. For example, since Vdd is not dependent on process variation, and circuit power P = (Vdd)(Idd):

$$\frac{dP}{dT_{ox}} = V_{dd}\frac{dI_{dd}}{dT_{ox}} \tag{22}$$

that is, a power sensitivity is just a constant times the branch current sensitivity.

The sensitivity of switching time, τ, can be calculated from the sensitivity of the node voltage, as shown in [1]. The result is:

$$\frac{\partial \tau}{\partial T_{ox}} = -\frac{\partial V_{out}}{\partial T_{ox}} / \dot{v}(t) \tag{23}$$

showing how the sensitivity of switching time can be computed from the node voltage process sensitivity and the time derivative of the voltage waveform, which can also be computed in SPICE.

6.0 A VLSI Circuit Manufacturability Metric

An EE244 project was done in conjunction with this project, in which a CAD tool was implemented to use the Robust Design Method to optimize the performance and manufacturability of VLSI circuit blocks [10]. The objective was to select a design from a group of design choices the one which met the performance criteria for speed, area and power, and minimized the sensitivity of the circuit to manufacturing process variation. The method uses orthogonal arrays to design an experiment (consisting of a series of circuit simulations) and assumes an additive model of effects to compute the results.

An example of an adder bit slice was used to demonstrate the new tool. Static and transmission gate adder topologies were compared, and the lengths and widths of several critical transistors were also included as design parameters. The performance functions considered were speed, area, power, sensitivity to Tox variation, and sensitivity to LD variation. Because the two projects were done concurrently, the sensitivity calculation presented below was done using perturbation.

Taguchi style signal to noise ratios were calculated to compare the performances of the designs under consideration. The performance metric most relevant to this project is the "Signal-to-Noise" (S/N) ratio of speed to variations in tox, defined as:

$$S/N\ PerformanceMetric = \log\left(\frac{speed}{sensitivity\ to\ tox}\right) \tag{24}$$

The result for this performance metric for the adder experiment is shown in Figure 4. It can be seen that the two circuit topologies display a strong difference in their speed to sensitivity ratio, whereas the output device width has little effect.

7.0 Conclusions and Future Work

This project has shown the feasibility of implementing process parameter sensitivity calculations in SPICE. Good agreement was obtained between theoretical and perturbation sensitivity results. The sensitivity was applied on an example circuit, to show its use in a design for manufacturability tool.

8.0 References

[1] D.E. Hocevar, P. Yang, T.N. Trick, B.D. Epler, "Transient Sensitivity Computation for MOSFET Circuits," *IEEE Trans. on CAD*, Vol. CAD-4, No. 4, pp 609-620, Oct. 1985.

[2] U. Choudhury, "Sensitivity Analysis in SPICE3," Master's Thesis, UC Berkeley, 1988.

[3] W. Nye, D.C. Riley, A. Sangiovanni-Vincentelli, A.L. Tits, "DELIGHT.SPICE: An Optimization-Based System for the Design of Integrated Circuits," *IEEE Trans. on CAD*, Vol. 7, No. 4, pp 501-518, April 1988.

[4] P. Cox, P. Yang, S. Mahant-Shetti, P. Chatterjee, "Statistical Modeling for Efficient Parametric

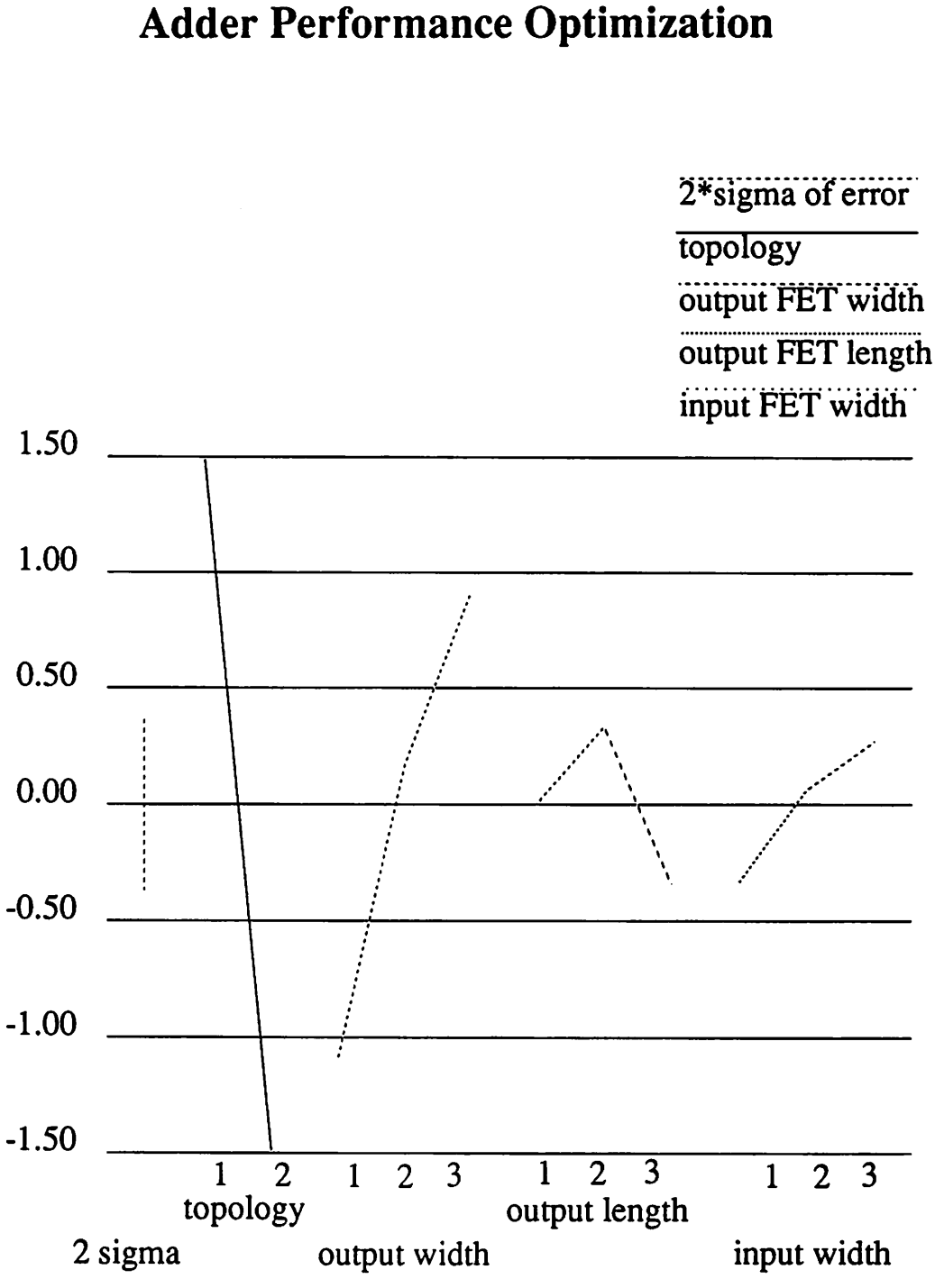


Figure 4. Circuit S/N Performance Metric Result

Yield Estimation of MOS VLSI Circuits," *IEEE Journal of Solid-State Circuits*, Vol. SC-20, No. 1, February, 1985, pp. 391-398.

[5] S.W Director and R.A. Rohrer, "The Generalized Adjoint Network and Network Sensitivities," *IEEE Trans on Circuit Theory*, Vol CT-16, No. 3, pp 318-323, August, 1969.

[6] R.K. Brayton and S.W. Director, "Computation of Delay Time Sensitivities for Use in Time Domain Optimization," *IEEE Trans. on Circuits and Systems*, Vol. CAS-22, No. 12, pp 910-920, Dec. 1975.

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

[8] P. Yang and P.K. Chatterjee, "SPICE Modeling for Small Geometry MOSFET Circuits," *IEEE Trans. on CAD*, Vol CAD-1, No. 4, October 1982, pp 169-182.

[9] E.D Boskin, "Physically Based SPICE Models for Statistical Circuit Design," EE231 Final Project Report, UC Berkeley, May 1991.

[10] Z. Daoud, "Application of Robust Design Techniques to IC Tolerance Design," EE244 Final Project Report, UC Berkeley, December 1991.

Application of the Robust Design Method to IC Design for Manufacturability

Zeina Daoud

Advisor: Prof. Costas Spanos

Department of Electrical Engineering and Computer Sciences

University of California, Berkeley

January 27, 1992

Abstract

The Robust Design Method is a technique aimed at designing high quality products at lower 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 studied ad hoc 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. The developed CAD tool allows the user to study the effect of certain design parameters and manufacturing variations on specific circuit performance measures. Upon analyzing the results, the user can choose an optimal setting of the decision variables.

1.0 Introduction

Every component of a manufactured product is subject to variation. Parameter variations may cause product samples to fall outside the performance specifications, and hence be rejected. Since a low manufacturing yield is economically unacceptable, there is a need to either reduce the tolerance of the components or to reduce the effect of component variations on product performance. Reducing component tolerance or manufacturing variation can be very costly. Tolerance design methods attempt to design products less sensitive to variations, in order to increase manufacturing yield.

Another advantage of tolerance design is discovering which parameters, if any, are not critical to the design. By relaxing the constraints of these parameters, a lower cost product can be manufactured with no compromise in performance or quality.

The Robust Design Method [1] is a tolerance design technique. It uses an orthogonal array of experiments to explore several decision variables with a small number of experiments. It assumes an additive model for factor effects, and no cross correlation between parameters.

The computer-aided-design tool developed applies the Robust Design Method to the design of integrated circuits using the HSPICE [2] circuit simulator. This report illustrates how this method helped to improve the design of a common VLSI circuit block, an adder bit slice. The design variables considered are circuit topology, width and length of the carry output buffer transistors, and width of the carry input buffer transistors. The output functions to optimize are speed, area, power, sensitivity of speed to changes in the thickness of the oxide, and sensitivity of speed to variations in the channel length. The thickness of the oxide and the channel length reduction are important causes of performance variation in modern VLSI designs [3]. After the analysis of the experiment, the accuracy of the additive model of factor effects is tested and conclusions are drawn about the use of such a tool.

2.0 Previous Work

The subject of tolerance design was first studied in the early 1970s. By the early 1980s, two main techniques had emerged: a deterministic approach and a statistical approach [4].

Both techniques are concerned with determining the region of acceptability [5] 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.

2.1 Deterministic Approach

The deterministic approach, also called simplicial approximation method (Director and Hachtel, 1977) [6] 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 disadvantage of this method is that its complexity increases dramatically with the number of adjustable parameters. Because of this, it is not practical to apply this method to circuits with more than five design parameters [5].

2.2 Statistical Approach

The statistical exploration approach to tolerance design [7] is based on Monte Carlo analysis techniques. The actual circuit manufacturing process is simulated by making random selections of component parameter values, given 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 these circuits which pass specifications.

An important property of such a 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.

2.3 Robust Design Method

The Robust Design Method draws on many ideas from statistical experimental design in order to plan experiments for obtaining information about variables involved in making engineering decisions. 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. Among them, the Robust Design Method makes heavy use of orthogonal arrays, whose use for planning experiments was first suggested by Rao [8]. The fundamental principle of Robust Design is to improve the quality of a product by minimizing the effect of the causes of variations without eliminating the causes.

The Robust Design Method was founded 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 [9]. This study proposes to apply this method to integrated circuit tolerance design as described in the next section.

3.0 Robust Design Technique applied to IC design

This section describes the Robust Design Method and how it is applied to IC design. Section 3.1 describes the fundamentals of the Robust Design Method. In section 3.2, the problem is presented. Sections 3.3 and 3.4 describe how the design is optimized based on the Robust Design Method. The results are explained in section 3.5.

3.1 Overview of the Robust Design Method

A product's performance degrades because of variations in product parameters and 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 this non-linearity to find a set of design parameter values that cause the smallest deviation of the quality characteristic from its desired target [1].

In previous work, optimal sets of design parameter values were found by intuition or by trial-and-error. 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 that space and draws conclusions based on the results of that subset. It uses a mathematical tool called *orthogonal arrays* 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. This implies that each parameter 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. Either way, the results will show if the parameters picked by the designer have a significant interaction.

The orthogonal arrays are used to define the matrix experiment. A matrix experiment consists of a set of trials where we change settings of various parameters (or factors) 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 each trial, a quality measure, called *signal-to-noise* ratio, is calculated for every output function to optimize.

The factor effect of every parameter on every function is then 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 as shown next:

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

$$FE_{P_L} = \sum_{i=1}^{m} SN_{P_L} - \sum_{i=1}^{n} SN$$

where $\overline{S\,N}$ is the output mean of all n trials (expressed in "signal-to-noise" units) and $\overline{SN}_{P_L}$ is the output mean of the m trials where parameter P is set to level L. A plot of factor effects for all output functions helps select the parameter settings that optimize the desired output functions. This optimal setting combination might not be one of the trials.

3.2 An Example of the Application of the Robust Design Method

The objective is to design a transmission gate adder bit slice. The five functions to optimize are speed, area, power, sensitivity of speed to variations of the thickness of the oxide, and sensitivity of speed to variations of the channel length. The speed of the ripple carry-out was chosen as a measure of the speed of the circuit. The four decision variables are the topology of the circuit, the width and the length of the carry output buffer transistors, and the width of the carry input buffer transistors. The levels of each factor are defined as follows:

- topology can be either that of a full static adder or of a transmission gate adder

- the width of the carry input and output buffer transistors are set to 8, 10 or 12 microns for n type transistors and 20, 22 or 24 microns for p type transistors

- the length of the carry output buffer transistors are set to 1.8, 2 or 2.2 microns.

We are thus exploring two different settings of topology, three levels of width and three levels of length of transistors, as summarized in the table 1 next:

Table 1: Definition of parameter levels

factor	level 1	level 2	level 3
topology	transm. gate	full static	--
width_out	W_0	$W_0 + i$	$W_0 + 2i$
length_out	L_0	L_0 - 0.2 μm	L_0 + 0.2 μm
width_in	W_0	$W_0 + i$	$W_0 + 2i$

where W_0 and L_0 represent respectively, the original width and length of the transistors and i is an increment of 2 μm to the width of n-type transistors, 3 μm to the width of p-type transistors.

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. Instead of making an automated choice about the relative importance of the output function (which would involve associating weight functions with outputs), the designer is offered the results of the factor effect plots so that she or he can decide which trade-offs are more appropriate. Confirmation runs are made available for the user to check the results for the combination of parameter values selected.

3.2.1 Experiment

The matrix experiment corresponding to four input parameters, one at two levels and three at three levels utilizes the L_{18} orthogonal array as given in [1], and shown in table 2.

Table 2: Experiment Matrix

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

An input to the tool is two circuit descriptions, one corresponding to a full static adder topology and another corresponding to a transmission gate adder topology. For every experiment, three HSPICE circuit simulator runs are performed. First, the set of parameter values is read from the matrix experiment and the appropriate SPICE deck is modified. This is the "nominal" SPICE deck. From the results of this first run, the nominal delay of switching between the carry in and the carry out, and the power dissipated in that circuit are obtained. The area (or change in area) is calculated from the values of length and width of the carry buffer transistors. A second HSPICE input reflects a change in the value of the thickness of the oxide of all transistor models from the nominal circuit. This second run gives a new value for the delay, *tox_delay*. Similarly, the value of the channel length is modified in all transistor models from the nominal circuit and a new value for the delay, *ld_delay*, is extracted from the HSPICE run.

3.2.2 Calculations

Given the raw data extracted from the simulations results, signal-to-noise (SN) ratios of output functions are calculated. They are defined in a way so as to *maximize* SN. SN of speed is the logarithm of the inverse of the nominal delay; SN of area and power are the logarithms of the inverse of area and power respectively.

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

$$SN_{area} = \log\left(\frac{1}{\text{area}}\right)$$

$$SN_{power} = \log\left(\frac{1}{\text{power}}\right)$$

Maximizing signal-to-noise ratios is equivalent to minimizing delay, area and power. To calculate the signal-to-noise ratio of the sensitivity of speed to variations in *tox* (or *ld*), the sensitivity is first defined as the ratio of the change in delay to the change in *tox* (or *ld*).

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

The SN of sensitivity of speed to variations in tox (or ld) is then:

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

where speed is the inverse of the nominal delay. Taking the logarithm of functions represents the additive property of factor effects. Similar equations are obtained for the sensitivity to variations of the channel length.

For every experiment, all signal-to-noise output functions are calculated. At the end of the matrix experiment, the factor effect of every parameter on every output function is computed as defined in section 3.0, and factor effect plots are generated for every output function.

Predicted output values are then calculated for every experiment by adding up the factor effects of every parameter set at the levels defined by that experiment. If the parameters picked for the study were absolutely independent, there should be no discrepancy between predicted values and HSPICE experimental values. Since HSPICE has zero experimental error, the difference between expected and experimental results can only be attributed to the fitting error of this model, as applied to the given input.

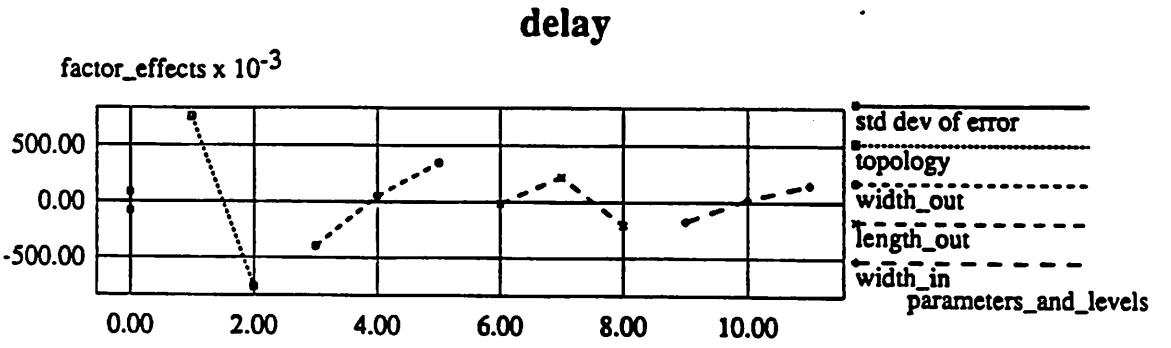
This lack of experimental error makes it difficult to quantify the "goodness" of the model in any statistical sense. One heuristic measure is to calculate the standard deviation of the prediction error, and add it to the factor effect output plots. Assume, for instance, that the plot is indicating that setting parameter P at level i instead of i+1 gives an incremental advantage A on the output function. If A is bigger than twice the standard deviation of the prediction error, then the result is correct more than 95% of the time. If however, A is less than twice the sigma of the prediction error, then setting parameter P at level i is essentially the same as setting it to level i+1. With this knowledge, the factor effect plots, including the range of plus or minus twice the sigma of the prediction error, are given to the user to make a choice of setting for each parameter. The designer can explore equivalent level settings, if any, and verify an optimal choice by running confirmation runs and looking at the theoretical and experimental results, and the error between them.

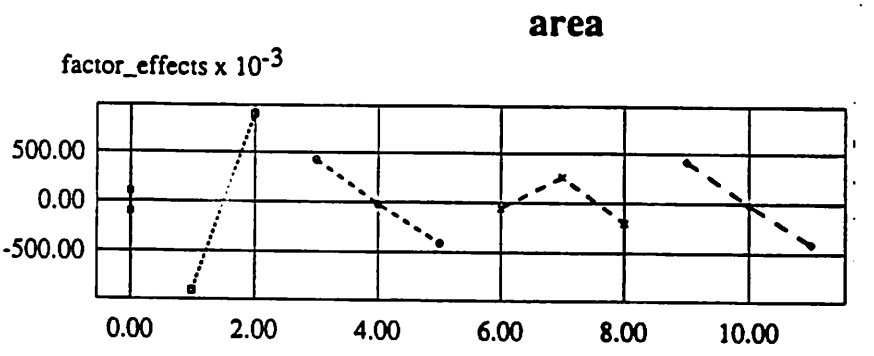
3.2.3 Results

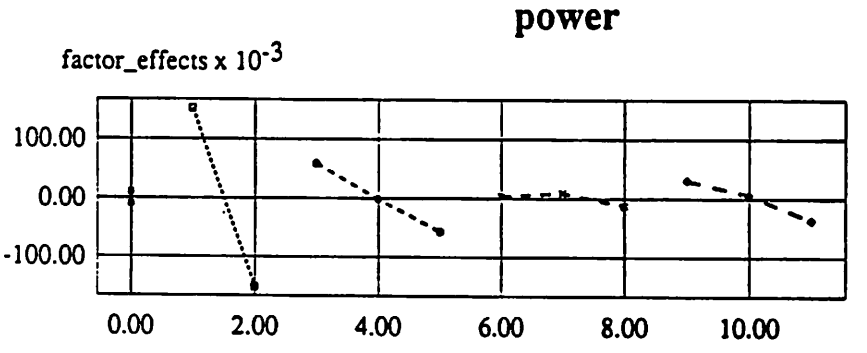
The table of results in the Appendix shows for all experiments the various parameter settings and the experimental signal-to-noise ratios of all output functions. For every output function, the mean of the experimental results, as well as the standard deviation of the prediction error, is calculated. The plots shown on the next page are a graphical representation of the factor effects of every design parameter on the five output functions. By looking at the factor effect plots, the designer gains knowledge on two design aspects:

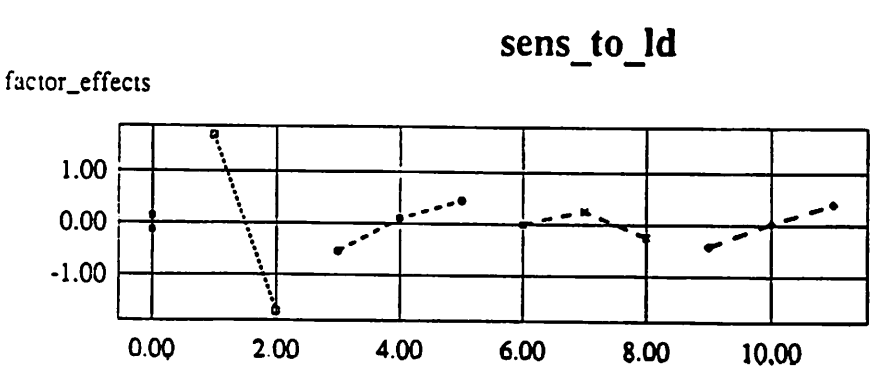
1) Optimal parameter settings can be chosen: because signal-to-noise ratios are maximized, an optimal parameter setting for a given function is the level with the largest factor effect. For example, in order to maximize speed, the topology has to be set at level one.

2) The designer can conclude, from the plots, that a parameter has an insignificant effect on an output function, if the variations between different levels is smaller than twice the sigma of the error. For instance, the factor effect plot on power shows that the width of the input buffer transistors has little impact on the power of the circuit.


delay
factor_effects x 10-3
500.00
0.00
-500.00
0.00
2.00
4.00
6.00
8.00
10.00
std dev of error
topology
width_out
length_out
width_in
parameters_and_levels


area
factor_effects x 10-3
500.00
0.00
-500.00
0.00
2.00
4.00
6.00
8.00
10.00


power
factor_effects x 10-3
100.00
0.00
-100.00
0.00
2.00
4.00
6.00
8.00
10.00


sens_to_ld
factor_effects
1.00
0.00
-1.00
0.00
2.00
4.00
6.00
8.00
10.00

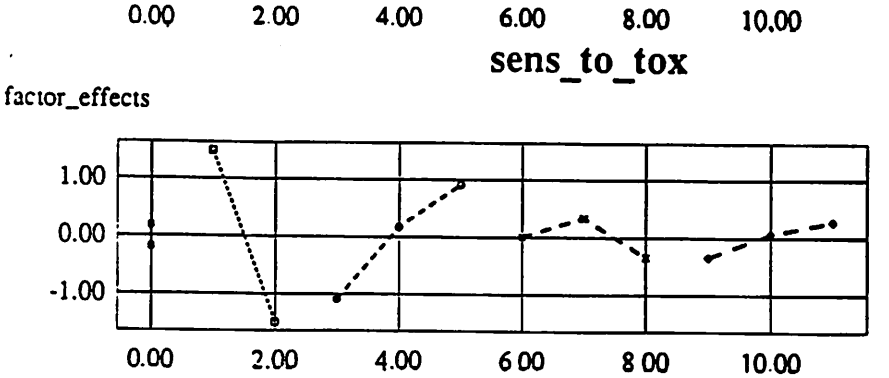

sens_to_tox
factor_effects
1.00
0.00
-1.00
0.00
2.00
4.00
6.00
8.00
10.00

Table 3 below shows the best parameter settings for each output function, where the parameter levels are defined in table 1.

Table 3: Optimal parameter settings

factors	speed	area	power	sens_to_ld	sens_to_tox
topology	1	2	1	1	1
width_out	3	1	1	3	3
length_out	2	2	2 or 1	2 or 1	2 or 1
width_in	3 or 2	1	1	3	3 or 2

Note that some parameters have several equally acceptable settings, with little impact on the output function. If the decision making was left to the automated tool, there would have been a need to incorporate in the tool, some idea of the relative importance of the functions to optimize. As stated above, it is found to be more useful to give the designer the flexibility to make such trade-offs and verify them with confirmation runs.

One way of picking an optimal set of parameter values is choosing settings that optimize the largest number of output functions, regardless of their relative importance. The first confirmation run thus picked on our problem was done with the set of parameter levels (1,3,2,3) in that order. The results are shown in the Appendix. The absolute error of every output function on that confirmation run is smaller than twice the standard deviation of the prediction error. Therefore, there is no significant interaction between the parameters. The improvement over the original design (parameter levels (1,1,1,1)) is noticeable for all output functions except the area.

This suggests another heuristic for picking optimal values for confirmation runs, by ignoring one (or more) output function. For instance, if area considerations were ignored, the set of values (1, 3,1,3) would show results as good as the first confirmation run performed.

The problem becomes that of picking appropriate factors to vary prior to running the experiments. The prediction capabilities of the additive model can be impaired if input parameters are dependent. However, the method does detect the presence of strongly interacting parameters and provides a way to quantify the interaction through the standard deviation of the prediction error. Moreover, in practice, designers often have a good idea of parameter dependencies and appropriate parameter choices are made.

4.0 Conclusions and Future Work

In this report, the fundamentals of the Robust Design Method were discussed and its application to integrated circuit design was presented. We showed how this experimental design method helped improve several performance characteristics of an adder bit slice.

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 by trying all possible combinations of parameter settings. By examining the signal-to-noise ratios of output functions, a few sets of optimal parameter values emerge. Confirmation runs let the designer explore them and pick the best one. This tool lets the designer account for 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. As more applications of the Robust Design Method to IC design are undertaken, the usefulness of this tool will become even more apparent.

5.0 References

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

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

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

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

[9] 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.

APPENDICES

I - Tabulated Results

II - HSPICE Data

III - Two Adder Topologies and Corresponding Spice Models

APPENDIX I

factor	level_1	level_2	level_3
topology	tgaddr	staddr	--
w_out	W	W + i	W + 2i
l_out	L	L - j	L + j
w_in	W	W + i	W + 2i

where W is the starting width (in meters)
i = 2 microns for n_type, and 3 microns for p_type;
and L is the starting length (in meters)
j = 0.2 microns for both n and p types

Signal-to-Noise results (in db)

exp	top	w_out	l_out	w_in	delay	area	power	S/ld_sens	S/tox_sens
1	1	1	1	1	86.26	97.75	35.32	113.01	101.14
2	1	1	2	2	86.67	97.58	35.29	113.73	101.94
3	1	1	3	3	86.39	96.85	35.22	113.50	101.31
4	1	2	1	1	86.57	97.38	35.26	113.51	102.31
5	1	2	2	2	87.00	97.26	35.24	114.02	102.83
6	1	2	3	3	86.72	96.52	35.16	114.17	102.58
7	1	3	1	2	87.01	96.72	35.17	114.26	103.16
8	1	3	2	3	87.40	96.64	35.15	114.97	103.71
9	1	3	3	1	86.59	96.79	35.19	113.31	102.33
10	2	1	1	3	84.82	98.79	34.95	110.03	98.28
11	2	1	2	1	84.80	100.14	35.01	109.64	98.18
12	2	1	3	2	84.54	99.13	34.99	109.60	97.89
13	2	2	1	2	85.31	98.79	34.91	110.56	99.56
14	2	2	2	3	85.63	98.53	34.89	111.13	100.17
15	2	2	3	1	84.91	99.08	34.94	109.95	98.85
16	2	3	1	3	85.80	97.90	34.83	111.39	100.85
17	2	3	2	1	85.69	99.07	34.89	110.82	100.45
18	2	3	3	2	85.47	98.09	34.85	110.75	100.27
mean of experiments					85.98	97.95	35.07	112.13	100.88
sigma of error					0.08	0.10	0.01	0.13	0.18

Confirmation run: theoretical & experimental results

theo	1	3	2	3	87.45	96.50	35.14	114.94	103.89
expm	1	3	2	3	87.40	96.64	35.15	114.97	103.71
absolute error					0.06	-0.15	-0.01	-0.03	0.18

Confirmation run: theoretical & experimental results

theo	1	3	1	3	87.22	96.18	35.13	114.68	103.56
expm	1	3	1	3	87.18	96.42	35.13	114.75	103.71
absolute error					0.04	-0.24	0.00	-0.07	-0.15

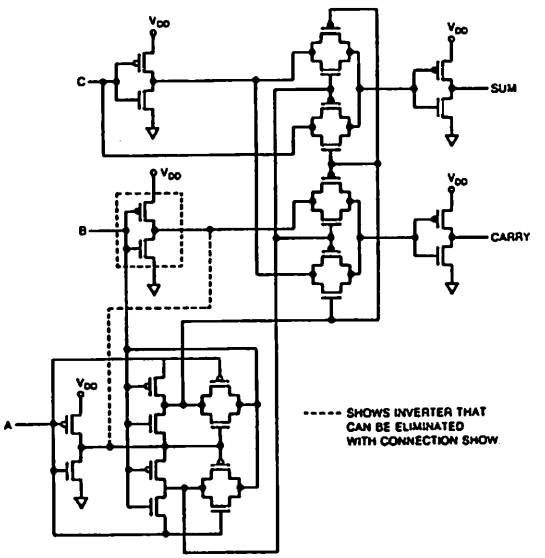
APPENDIX II

```
        experiment 1 :
   topology = t,  w_out = 1,  l_out = 1,  w_in = 1
nominal delay = 2.36624e-09
area = 1.68e-10
power = -0.000293988
ld_delay = 2.47192e-09
tox_delay = 2.44756e-09
delta_ld_delay / delta_ld = 0.0021137
delta_tox_delay / delta_tox = 0.032526

        experiment 2 :
   topology = t,  w_out = 1,  l_out = 2,  w_in = 2
nominal delay = 2.15195e-09
area = 1.746e-10
power = -0.000295574
ld_delay = 2.2504e-09
tox_delay = 2.2263e-09
delta_ld_delay / delta_ld = 0.0019689
delta_tox_delay / delta_tox = 0.02974

        experiment 3 :
   topology = t,  w_out = 1,  l_out = 3,  w_in = 3
nominal delay = 2.29697e-09
area = 2.064e-10
power = -0.000300826
ld_delay = 2.39432e-09
tox_delay = 2.3775e-09
delta_ld_delay / delta_ld = 0.0019469
delta_tox_delay / delta_tox = 0.03221

        experiment 4 :
   topology = t,  w_out = 2,  l_out = 1,  w_in = 1
nominal delay = 2.20271e-09
area = 1.83e-10
power = -0.000297643
ld_delay = 2.30388e-09
tox_delay = 2.26941e-09
delta_ld_delay / delta_ld = 0.0020233
delta_tox_delay / delta_tox = 0.02668

        experiment 5 :
   topology = t,  w_out = 2,  l_out = 2,  w_in = 2
nominal delay = 1.99656e-09
area = 1.881e-10
power = -0.00029898
ld_delay = 2.09587e-09
tox_delay = 2.06186e-09
delta_ld_delay / delta_ld = 0.001986
delta_tox_delay / delta_tox = 0.02612

        experiment 6 :
   topology = t,  w_out = 2,  l_out = 3,  w_in = 3
nominal delay = 2.12571e-09
area = 2.229e-10
power = -0.000304683
ld_delay = 2.21583e-09
tox_delay = 2.19057e-09
delta_ld_delay / delta_ld = 0.0018024
delta_tox_delay / delta_tox = 0.025946

        experiment 7 :
   topology = t,  w_out = 3,  l_out = 1,  w_in = 2
nominal delay = 1.99218e-09
area = 2.13e-10
power = -0.000304064
ld_delay = 2.08623e-09
tox_delay = 2.05275e-09
delta_ld_delay / delta_ld = 0.001881
delta_tox_delay / delta_tox = 0.024228

        experiment 8 :
   topology = t,  w_out = 3,  l_out = 2,  w_in = 3
nominal delay = 1.82166e-09
area = 2.166e-10
power = -0.000305517
ld_delay = 1.90914e-09
tox_delay = 1.88009e-09
delta_ld_delay / delta_ld = 0.0017496
delta_tox_delay / delta_tox = 0.023374

        experiment 9 :
   topology = t,  w_out = 3,  l_out = 3,  w_in = 1
nominal delay = 2.19276e-09
area = 2.094e-10
power = -0.000302781
ld_delay = 2.29924e-09
tox_delay = 2.25946e-09
delta_ld_delay / delta_ld = 0.0021297
delta_tox_delay / delta_tox = 0.026678

        experiment 10 :
   topology = s,  w_out = 1,  l_out = 1,  w_in = 3
nominal delay = 3.2961e-09
area = 1.32e-10
power = -0.000319654
ld_delay = 3.44674e-09
tox_delay = 3.40887e-09
delta_ld_delay / delta_ld = 0.0030127
delta_tox_delay / delta_tox = 0.045106

        experiment 11 :
   topology = s,  w_out = 1,  l_out = 2,  w_in = 1
nominal delay = 3.31491e-09
area = 9.69e-11
power = -0.000315559
ld_delay = 3.47883e-09
tox_delay = 3.4296e-09
delta_ld_delay / delta_ld = 0.0032782
delta_tox_delay / delta_tox = 0.045874

        experiment 12 :
   topology = s,  w_out = 1,  l_out = 3,  w_in = 2
nominal delay = 3.51652e-09
area = 1.221e-10
power = -0.000316898
ld_delay = 3.67247e-09
tox_delay = 3.63215e-09
delta_ld_delay / delta_ld = 0.0031192
delta_tox_delay / delta_tox = 0.046254

        experiment 13 :
   topology = s,  w_out = 2,  l_out = 1,  w_in = 2
nominal delay = 2.94424e-09
area = 1.32e-10
power = -0.000322494
ld_delay = 3.09337e-09
tox_delay = 3.03812e-09
delta_ld_delay / delta_ld = 0.0029826
delta_tox_delay / delta_tox = 0.037552

        experiment 14 :
```

```
   topology = s,  w_out = 2,  l_out = 2,  w_in = 3
nominal delay = 2.73296e-09
area = 1.404e-10
power = -0.000324059
ld_delay = 2.87397e-09
tox_delay = 2.82087e-09
delta_ld_delay / delta_ld = 0.0028203
delta_tox_delay / delta_tox = 0.035168

        experiment 15 :
   topology = s,  w_out = 2,  l_out = 3,  w_in = 1
nominal delay = 3.22611e-09
area = 1.236e-10
power = -0.000320343
ld_delay = 3.38275e-09
tox_delay = 3.32721e-09
delta_ld_delay / delta_ld = 0.0031327
delta_tox_delay / delta_tox = 0.04044

        experiment 16 :
   topology = s,  w_out = 3,  l_out = 1,  w_in = 3
nominal delay = 2.62955e-09
area = 1.62e-10
power = -0.000328615
ld_delay = 2.76759e-09
tox_delay = 2.70767e-09
delta_ld_delay / delta_ld = 0.0027607
delta_tox_delay / delta_tox = 0.031246

        experiment 17 :
   topology = s,  w_out = 3,  l_out = 2,  w_in = 1
nominal delay = 2.7004e-09
area = 1.239e-10
power = -0.000324142
ld_delay = 2.8537e-09
tox_delay = 2.7839e-09
delta_ld_delay / delta_ld = 0.003066
delta_tox_delay / delta_tox = 0.033402

        experiment 18 :
   topology = s,  w_out = 3,  l_out = 3,  w_in = 2
nominal delay = 2.83844e-09
area = 1.551e-10
power = -0.00032704
ld_delay = 2.98674e-09
tox_delay = 2.92122e-09
delta_ld_delay / delta_ld = 0.0029661
delta_tox_delay / delta_tox = 0.03311


        Confirmation run
   topology = t,  w_out = 3,  l_out = 2,  w_in = 3
nominal delay = 1.82166e-09
area = 2.166e-10
power = -0.000305517
ld_delay = 1.90914e-09
tox_delay = 1.88009e-09
delta_ld_delay / delta_ld = 0.0017496
delta_tox_delay / delta_tox = 0.023374


        Confirmation run
   topology = t,  w_out = 3,  l_out = 1,  w_in = 3
nominal delay = 1.91581e-09
area = 2.28e-10
power = -0.000306879
ld_delay = 2.0032e-09
tox_delay = 1.97129e-09
delta_ld_delay / delta_ld = 0.001748
delta_tox_delay / delta_tox = 0.022194
```

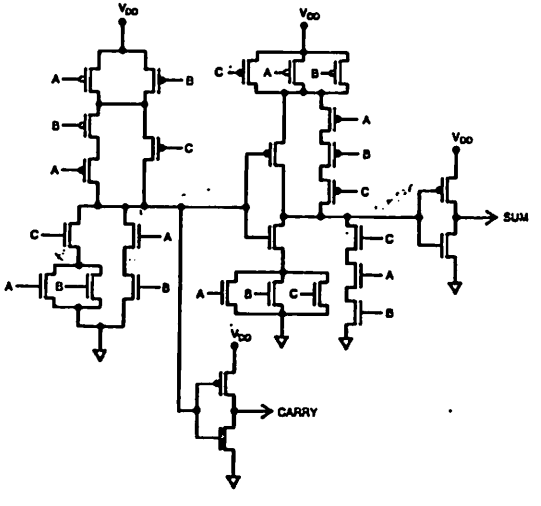
APPENDIX III

```
Manufacturability Project, Transmission Gate Ripple Adder

.option brief nomod

.model n nmos level=2 ld=0.138260e-6 tox=398.0e-10
+nsub=5.36726e+15 vto=0.743469 gamma=0.486502
+phi=0.6 uo=655.881 uexp=0.157282 ucrit=31443.8
+delta=2.39824 vmax=55260.9 xj=0.25u lambda=0.0367072
+nfs=1e+12 neff=1.001 nss=1e+11 tpg=1.0
+rsh=36.87 cgdo=1.19953e-10 cgso=1.19953e-10
+cj=0.0001595 mj=0.658500 cjsw=5.249e-10 mjsw=0.240200 pb=0.580

.model p pmos level=2 ld=0.061533e-6 tox=398.0e-10
+nsub=4.3318e+15 vto=-0.738861 gamma=0.437062
+phi=0.6 uo=261.977 uexp=0.323932 ucrit=65719.8
+delta=1.79192 vmax=25694 xj=0.250u lambda=0.0612279
+nfs=1e+12 neff=1.001 nss=1e+11 tpg=-1.0
+rsh=146.6 cgdo=5.33853e-11 cgso=5.33853e-11
+cj=0.000255 mj=0.505200 cjsw=3.119e-10 mjsw=0.24170 pb=0.6400

.model diode d tt=0 rs=0 cjo=0

* Power supplies and input
vdd 1 0 DC 5V
va 2 0 pwl(0ns 5V 20ns 5V)
vb 3 0 pwl(0ns 0V 20ns 0V)
vc 4 0 pwl(0ns 0V 5ns 0V 7ns 5V 13ns 5V 15ns 0V 20ns 0V)

* the full adder - need not(A) and not (C)
minn 5 4 0 0 n w=8e-6 l=2u ps=36u pd=12u as=40p ad=24p
minp 5 4 1 1 p w=20e-6 l=2u ps=56u pd=12u as=100p ad=60p
m3 6 2 0 0 n w=3u l=2u ps=11u pd=11u as=12p ad=12p
m4 6 2 1 1 p w=8u l=2u ps=16u pd=16u as=32p ad=32p
* the xor/xnor
m5 7 3 2 0 n w=3u l=2u ps=11u pd=11u as=12p ad=12p
m6 8 3 6 0 n w=3u l=2u ps=11u pd=11u as=12p ad=12p
m7 8 6 3 0 n w=3u l=2u ps=11u pd=11u as=12p ad=12p
m8 7 2 3 0 n w=3u l=2u ps=11u pd=11u as=12p ad=12p
m9 6 3 7 1 p w=8u l=2u ps=16u pd=16u as=32p ad=32p
m10 2 3 8 1 p w=8u l=2u ps=16u pd=16u as=32p ad=32p
m11 8 2 3 1 p w=8u l=2u ps=16u pd=16u as=32p ad=32p
m12 7 6 3 1 p w=8u l=2u ps=16u pd=16u as=32p ad=32p
* produce sum with buffer
m13 5 7 9 0 n w=3u l=2u ps=11u pd=11u as=12p ad=12p
m14 4 8 9 0 n w=3u l=2u ps=11u pd=11u as=12p ad=12p
m15 5 8 9 1 p w=8u l=2u ps=16u pd=16u as=32p ad=32p
m16 4 7 9 1 p w=8u l=2u ps=16u pd=16u as=32p ad=32p
* sum buffer
m17 11 9 0 0 n w=3u l=2u ps=11u pd=11u as=12p ad=12p
m18 11 9 1 1 p w=8u l=2u ps=16u pd=16u as=32p ad=32p

* produce carry
m19 6 7 10 0 n w=3u l=2u ps=11u pd=11u as=12p ad=12p
m20 5 8 10 0 n w=8u l=2u ps=16u pd=16u as=32p ad=32p
m21 6 8 10 1 p w=8u l=2u ps=16u pd=16u as=32p ad=32p
m22 5 7 10 1 p w=20u l=2u ps=28u pd=28u as=80p ad=80p

* carry out buffer
* change w of m23 and m24 to adjust speed
moutn 12 10 0 0 n w=8e-6 l=2e-6 ps=36u pd=12u as=40p ad=24p
moutp 12 10 1 1 p w=20e-6 l=2e-6 ps=56u pd=12u as=100p ad=60p

* the load - sum then carry
c1 11 0 0.5pF
c2 12 0 0.5pF

.tran 0.1ns 20ns

.MEASURE TRAN DELAY1 TRIG V(4) VAL=2.5 RISE=1
+                    TARG V(12) VAL=2.5 RISE=1
.MEASURE TRAN DELAY2 TRIG V(4) VAL=2.5 FALL=1
+                    TARG V(12) VAL=2.5 FALL=1

.MEASURE TRAN POWER AVG I(VDD) FROM=0NS TO=20NS

.print tran v(4) v(12) (0,5)
.width out=80
.END
```



from Weste & Eshraghian "Principles of CMOS VLSI Design", 1985

```
Manufacturability Project, Full Static Adder

.option brief nomod

.model n nmos level=2 ld=0.138260e-6 tox=398.0e-10
+nsub=5.36726e+15 vto=0.743469 gamma=0.486502
+phi=0.6 uo=655.881 uexp=0.157282 ucrit=31443.8
+delta=2.39824 vmax=55260.9 xj=0.25u lambda=0.0367072
+nfs=1e+12 neff=1.001 nss=1e+11 tpg=1.0
+rsh=36.87 cgdo=1.19953e-10 cgso=1.19953e-10
+cj=0.0001595 mj=0.658500 cjsw=5.249e-10 mjsw=0.240200 pb=0.580

.model p pmos level=2 ld=0.061533e-6 tox=398.0e-10
+nsub=4.3318e+15 vto=-0.738861 gamma=0.437062
+phi=0.6 uo=261.977 uexp=0.323932 ucrit=65719.8
+delta=1.79192 vmax=25694 xj=0.250u lambda=0.0612279
+nfs=1e+12 neff=1.001 nss=1e+11 tpg=-1.0
+rsh=146.6 cgdo=5.33853e-11 cgso=5.33853e-11
+cj=0.000255 mj=0.505200 cjsw=3.119e-10 mjsw=0.24170 pb=0.6400

* Power supplies and input
vdd 1 0 DC 5V
va 8 0 pwl(0ns 5V 20ns 5V)
vb 9 0 pwl(0ns 0V 20ns 0V)
vc 10 0 pwl(0ns 0V 5ns 0V 7ns 5V 13ns 5V 15ns 0V 20ns 0V)

* The circuit
m1 2 8 1 1 p w=12u l=2u ps=56u pd=12u as=100p ad=60p
m2 2 9 1 1 p w=12u l=2u ps=56u pd=12u as=100p ad=60p
m3 3 9 2 1 p w=12u l=2u ps=56u pd=12u as=100p ad=60p
m4 4 8 3 1 p w=12u l=2u ps=56u pd=12u as=100p ad=60p
minp 4 10 2 1 p w=12e-6 l=2u ps=56u pd=12u as=100p ad=60p
minn 4 10 5 0 n w=5e-6 l=2u ps=16u pd=16u as=32p ad=32p
m7 4 8 6 0 n  w=5u l=2u ps=16u pd=16u as=32p ad=32p
m8 5 8 0 0 n w=5u l=2u ps=16u pd=16u as=32p ad=32p
m9 5 9 0 0 n w=5u l=2u ps=16u pd=16u as=32p ad=32p
m10 6 9 0 0 n w=5u l=2u ps=16u pd=16u as=32p ad=32p

* carry out buffer
* change w of m11 from nominal 5u (to 3u or 7u)
* change w of m12 from nominal 12u (to 8u or 15u)
moutp 7 4 1 1 p w=12e-6 l=2e-6 ps=56u pd=12u as=100p ad=60p
moutn 7 4 0 0 n w=5e-6 l=2e-6 ps=16u pd=16u as=32p ad=32p


m13 11 10 1 1 p w=12u l=2u ps=56u pd=12u as=100p ad=60p
m14 11 8 1 1 p w=12u l=2u ps=56u pd=12u as=100p ad=60p
m15 11 9 1 1 p w=12u l=2u ps=56u pd=12u as=100p ad=60p
m16 14 4 11 1 p w=12u l=2u ps=56u pd=12u as=100p ad=60p
m17 12 8 11 1 p w=12u l=2u ps=56u pd=12u as=100p ad=60p
m18 13 9 12 1 p w=12u l=2u ps=56u pd=12u as=100p ad=60p
m19 14 10 13 1 p w=12u l=2u ps=56u pd=12u as=100p ad=60p
m20 14 4 15 0 n w=5u l=2u ps=16u pd=16u as=32p ad=32p
m21 15 8 0 0 n w=5u l=2u ps=16u pd=16u as=32p ad=32p
m22 15 9 0 0 n w=5u l=2u ps=16u pd=16u as=32p ad=32p
m23 15 10 0 0 n w=5u l=2u ps=16u pd=16u as=32p ad=32p
m24 14 10 16 0 n w=5u l=2u ps=16u pd=16u as=32p ad=32p
m25 16 8 17 0 n w=5u l=2u ps=16u pd=16u as=32p ad=32p
m26 17 9 0 0 n w=5u l=2u ps=16u pd=16u as=32p ad=32p

* sum buffer
m27 18 14 1 1 p w=12u l=2u ps=56u pd=12u as=100p ad=60p
m28 18 14 0 0 n w=5u l=2u ps=16u pd=16u as=32p ad=32p

* the load - sum then carry
c1 18 0 0.5pF
c2 7 0 0.5pF

.tran 0.1ns 20ns

.MEASURE TRAN DELAY1 TRIG V(10) VAL=2.5 RISE=1
+                    TARG V(7) VAL=2.5 RISE=1
.MEASURE TRAN DELAY2 TRIG V(10) VAL=2.5 FALL=1
+                    TARG V(7) VAL=2.5 FALL=1

.MEASURE TRAN POWER AVG I(VDD) FROM=0NS TO=20NS

.print tran v(10) v(7) (0,5)
.width out=80
.END
```



from Weste & Eshraghian "Principles of CMOS VLSI Design", 1985.

A Fuzzy Evaluator for Technology Mapping

Hao-Cheng Liu & Raymond L. Chen

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

December 17, 1991

Abstract

Tools for technology mapping are critical because it is important for a designer to determine which technology is more appropriate for implementing a design early in the design cycle. This will help improve the interface between design and manufacturing, and will result in more manufacturable designs. Here we present a novel approach of utilizing fuzzy logic in order to map specific logic system designs to specific technologies by determining which technology will best fit the characteristics of a given design.

1.0 Introduction

1.1 Motivation

During the 1980s, Japanese electronics firms have enjoyed a competitive advantage they have achieved over their U.S. counterparts by being able to greatly shorten their product design cycle times. These shorter design cycle times are typically the results of better interfaces between design and manufacturing. Design cycle times can be greatly reduced if a designer can determine early in the design cycle some basic properties involved in the eventual manufacturing of a design. This is the role of "Technology Mapping".

Tools for technology mapping are critical because it is important for a designer to be able to determine which technology to use for the implementation of a design early in the design cycle. This will help improve the interface between design and manufacturing, and thus result in better designs for manufacturability. Therefore, we see that tools for technology mapping can aid tremendously in the shortening of design cycle times. And with this shortening of the product design cycle times, maybe U.S. electronics firms will be able to recapture some of the competitive advantage lost to the Japanese during the 1980s.

1.2 Approach

Here we present a novel approach to technology mapping by using fuzzy logic to evaluate designs and then mapping them to various technologies based on some characteristics of the designs and of the technologies. The types of technologies discussed above include various types of field-programmable gate arrays (FPGAs), application-specific integrated circuits (ASICs), multichip modules (MCMs), etc. A study must be done into each specific technology in order to determine their particular characteristics. Given the particular characteristics of each of the given technologies, appropriate fuzzy membership functions (to be discussed in more detail in Section 3.1) can be derived for each aspect of the technology. We then use fuzzy inference (to be discussed in more detail in Section 3.1) in order to map our designs to the specific technologies.

We have applied this approach of technology mapping to some field-programmable gate arrays. The results obtained show great promise in the functionality of this fuzzy evaluator for technology mapping.

1.3 Organization

We will first present some background information in Section 2. In Section 3, we will give a brief introduction to fuzzy logic and demonstrate how we have applied it toward technology mapping. The software implementation and the results of the application of this fuzzy evaluator to technology mapping for some field-programmable gate arrays will also be presented in this section. We will close with conclu-

sions and ideas for future work in Sections 4 and 5, respectively. We have included an appendix with the *C* code used for our application example.

2.0 Background

As far as we know, fuzzy logic has not been used in tools for technology mapping. Most tools seem to use simple functions, which may be derived empirically, for mapping designs to technologies.

Technology mapping tools that use arbitrary functions may be more accurate in some cases than tools that use fuzzy logic. However, the greater complexity involved in the derivation of these functions may prove the use of fuzzy logic to be superior. The simplicity of fuzzy logic is its greatest attribute. Fuzzy rules (to be defined in Section 3.1) follow simple human reasoning. Therefore, derivations of new fuzzy rules for technology mapping can be made simpler, and therefore at a lower cost, than equations. The advantages and simplicity of fuzzy logic will be discussed in the following sections.

3.0 A Fuzzy Evaluator for Technology Mapping

3.1 An Introduction to Elementary Fuzzy Logic

The concept of Fuzzy Logic was first introduced by Professor Lotfi A. Zadeh of the University of California at Berkeley, in June 1965. However, it was not very well-known to the science and technology community until recent years. In the last few years, however, the subject has flourished and applications of this theory can now be found in many disciplines. In this section, we will explain the basics of fuzzy logic and a fuzzy inference decision-making system. For brevity, we will focus on what we need for this project.

3.1.1 Fuzzy sets and membership function

Fuzzy logic is based on the concept of the fuzzy set. The fuzzy set theory is in many ways a generalization of the classical set theory. A *classical* (crisp) set A is normally defined as a collection of elements or objects $x \in X$ which satisfy certain conditions specifying A. Each element $x \in X$ can either belong to or not belong to the set A, where $A \subseteq X$. To generalize this definition, we can introduce a membership function μ (on X) for each elements to specify its belongness to the set A, i.e. $\mu(x)=1$ if $x \in A$ and $\mu(x)=0$ if $x \notin A$. If we allow this membership function to be continuous, we can define a *fuzzy* set B as a collection of elements $x \in X$ with membership function $\mu(x)$, where $\mu(x)$ can be any real number between 0 and 1:

$$B = \{ (x, \mu_B(x)) \mid x \in X \} \qquad (3\text{-}1)$$

3.1.2 Fuzzy logic and rules

In Boolean logic, the most basic logic operations we need to consider are "PASS", "COMPLEMENTARY", "AND" and "OR". The rules of those operations can expressed as the following table:

Table 1: Boolean logic rules

Name	Rule	Equivalent Fuzzy Membership Value
PASS	IF $x \in A$, THEN $z \in Z$	IF $\mu_A(x)=1$, THEN $\mu_Z(z)=1$
COMPL.	IF $x \notin A$, THEN $z \in Z$	IF $\mu_A(x)=0$, THEN $\mu_Z(z)=1$
AND	IF $x \in A \vee y \in B$, THEN $z \in Z$	IF $\mu_A(x)=1$ OR $\mu_B(y)=1$, THEN $\mu_Z(z)=1$
OR	IF $x \in A \wedge y \in B$, THEN $z \in Z$	IF $\mu_A(x)=1$ AND $\mu_B(y)=1$, THEN $\mu_Z(z)=1$

where "1" can be defined as "true" and "0" as "false".

Fuzzy logic can be regarded as a generalization of the classical Boolean logic by allowing the "membership function" $\mu(x)$ to be any number between 0 and 1. Equivalently, Boolean logic is a special case of the fuzzy logic. Thus, we can define the equivalent fuzzy logic rules ("fuzzy rules") for the above four basic logic operations as the follows:

Table 2: Fuzzy logic rules

Name	Rule	Membership Value
PASS	IF $(x, \mu_A(x)) \in A$, THEN $(z, \mu_Z(z)) \in Z$	$\mu_Z(z) = \mu_A(x)$
COMPL.	IF $(x, \mu_A(x)) \in A$, THEN $(z, \mu_Z(z)) \in Z$	$\mu_Z(z) = 1-\mu_A(x)$
AND	IF $(x, \mu_A(x)) \in A$ OR $(x, \mu_B(x)) \in B$, THEN $(z, \mu_Z(z)) \in Z$	$\mu_Z(z) = \max(\mu_A(x), \mu_B(x))$
OR	IF $(x, \mu_A(x)) \in A$ AND $(x, \mu_B(x)) \in B$, THEN $(z, \mu_Z(z)) \in Z$	$\mu_Z(z) = \min(\mu_A(x), \mu_B(x))$

where all of the values of membership functions μ's are between 0 and 1.

3.1.3 The concept of the linguistic variable

A linguistic variable has a name, and a set of *linguistic* values. Each of these linguistic values is associated with a value of a membership function. For example: a person's *age* A(name) can be a linguistic variable having linguistic values as *juvenile, young, middle-aged, old, very old* with membership functions for all ages between 10 and 100 as shown in Figure 1:

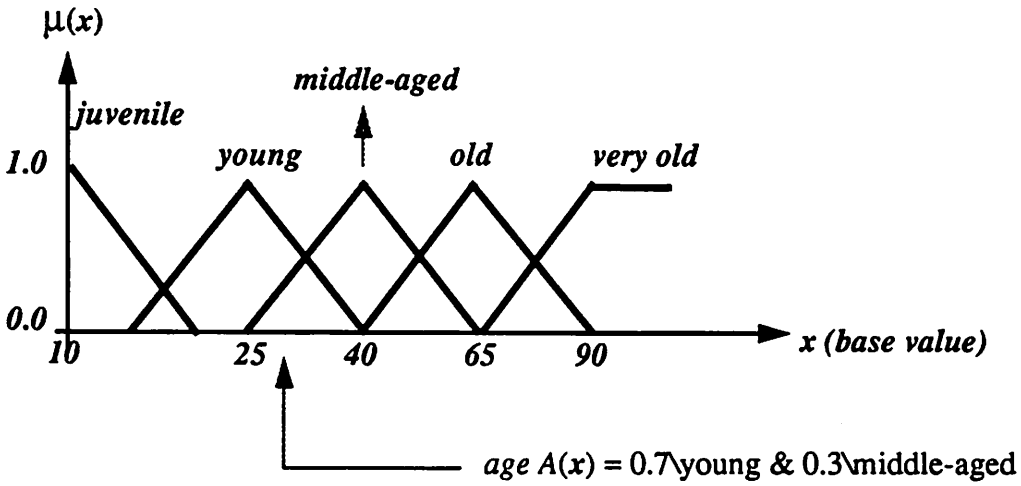


Figure 1: Membership functions of the various age groups

For example, if x=30, then the value of the linguistic "age" is $A(x)$ = 0.7\young & 0.3\middle-aged, which means that for x=30, the linguistic variable A has the value "young" with a membership of 0.75 and the value "middle-aged" with a membership of 0.25.

Now suppose that we have a rule that says: IF A is young THEN P is energetic, where P stands for a person's physical status. In the fuzzy logic terminology, this rule can be expressed as:

$$\mu_{\text{energetic}}(P) = \mu_{\text{young}}(A).$$

Figure 2 may offer a clearer picture:

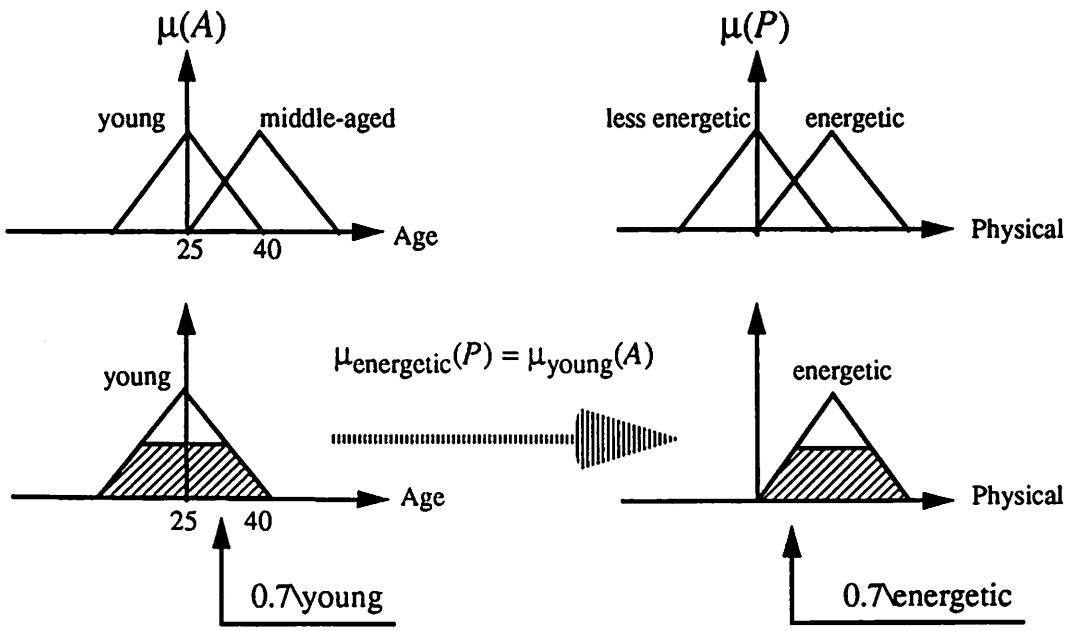


Figure 2: Example of fuzzy logic inference rule

This is actually a simple case of fuzzy inference with only one input (A) and one output (P). We will discuss more general cases of fuzzy inference in the next section.

3.1.4 Fuzzy inference

We can now illustrate the concept of fuzzy inference, which is an "approximate reasoning" technique, based on fuzzy logic rules, linguistic variables, and their membership functions.

Consider for example a professor that evaluates circuits designed by the students. First, assume that all the circuits are functionally correct. After functionality has been established, the professor uses two grading criteria: the propagation delay time t_p, and the total power consumption w. Expressed as "fuzzy" rules, these criteria are:

1. IF t_p small AND w small, THEN the grade g=A;
2. IF t_p small AND w big, THEN the grade g=B;
3. IF t_p median AND w small, THEN the grade g=B;
4. IF t_p median OR w median, THEN the grade g=B;
5. IF t_p big OR w big, THEN the grade g=C;

We can use a rule table (rule base) to express these rules more clearly:

Table 3: Example of rule base

	w small	w median	w big
t_p small	AND: g=A		AND: g=B
t_p median	AND: g=B	OR: g=B	
t_p big			OR: g=C

Remember that t_p, w and g are *linguistic variables*, and "small, median, big", or "A, B, C" are their *linguistic values*. We will use $N(\cdot)$ as our "defuzzifier" function: $N(t_p)$, $N(w)$ and $N(g)$ are the corresponding numerical values of the linguistic variables t_p, w and g, respectively. The relationship between the numerical value and the linguistic value of a linguistic variable is determined by the respective membership function.

To illustrate how the fuzzy inference concept works, let us examine the project of one student that resulted 15 *nsec* and 500 *mW* for delay time $N(t_p)$ and power consumption $N(w)$, respectively. From the available membership function profiles of the linguistic variables t_p and w, we obtain the following linguistic values:

t_p = 0.3\smalll & 0.8\median;
w = 0.7\median & 0.2\big.

Now after applying the fuzzy rules of Table 3, we obtain the following information about the grade:

$g_1 = \min(0.3, 0.0) = 0.0;$
$g_2 = \min(0.3, 0.2) = 0.2;$
$g_3 = \min(0.8, 0.0) = 0.0;$
$g_4 = \max(0.8, 0.7) = 0.8;$
$g_5 = \max(0.0, 0.2) = 0.2.$

It looks like different rule uses lead to "conflicting" results. To solve this problem, we need to apply a "fuzzy inference" approach which is a "weighted average" method to obtain the final linguistic value of g:

$$g = \frac{g_1 \cdot A + g_2 \cdot B + g_3 \cdot B + g_4 \cdot B + g_5 \cdot C}{g_1 + g_2 + g_3 + g_4 + g_5} \qquad (3\text{-}2)$$

In this case, we have $g = 0 + B/6 + 0 + 4B/6 + C/6 = 5B/6 + C/6$, or $g = 0.833\backslash B$ & $0.167\backslash C$ as in our convention. Sometimes we need the numerical value of an output linguistic variable after fuzzy inference, then we need to "defuzzify" the output to obtain its numerical values from its membership function definition. The above algorithm is illustrated by Figure 3 (on page 8).

We now should be ready to apply the above fuzzy logic algorithm to our technology mapping approach.

3.2 Applying Fuzzy Logic to Technology Mapping

3.2.1 General application of fuzzy logic to technology mapping

The application of fuzzy logic to technology mapping is simple. Engineers typically look at various design characteristics, such as area, input/output propagation delay, power dissipation, cost, etc., and based on those characteristics, make a decision on whether a particular technology is "acceptable" or "unacceptable" for the implementation of their design. By creating fuzzy sets and accurate membership functions for technology characteristics and using the appropriate fuzzy rules, we can use the simple idea of fuzzy inference to map a design to the various technologies, and determine whether each technology is "acceptable" or "unacceptable" for the implementation of our design. This idea is illustrated graphically in Figure 4.

3.2.2 Nonlinear fuzzy membership functions

It should be pointed out here that the fuzzy membership functions need not be simple linear functions as shown before. They may be nonlinear functions or empirically-derived functions. A S-curve membership function is illustrated in Figure 5.

Because lower gate counts, therefore smaller area, tend to be more "acceptable" for implementation, and higher gate counts (larger area) tend to be more "unacceptable" for implementation due to excessive

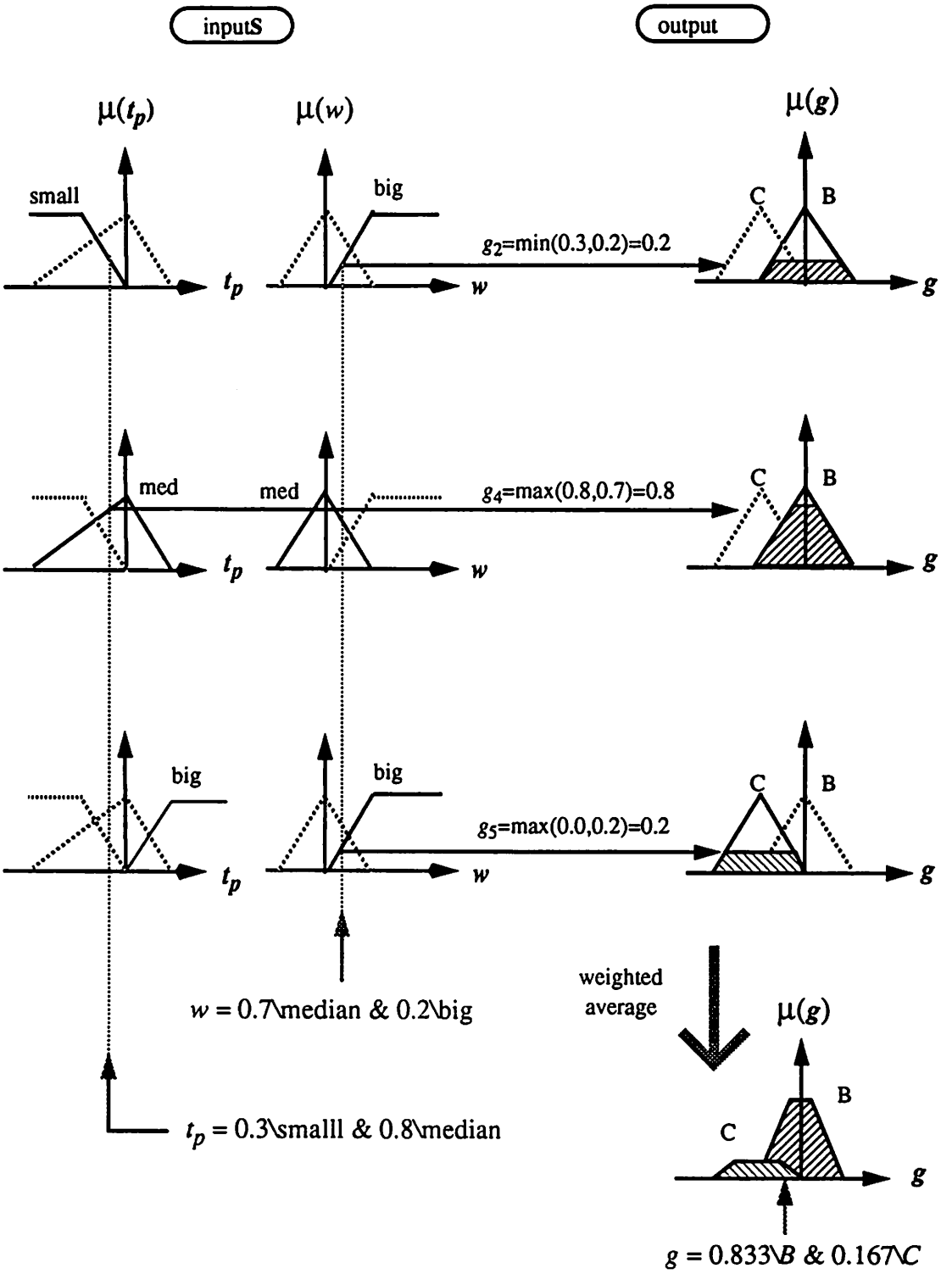


Figure 3: Fuzzy inference for conflicting rules

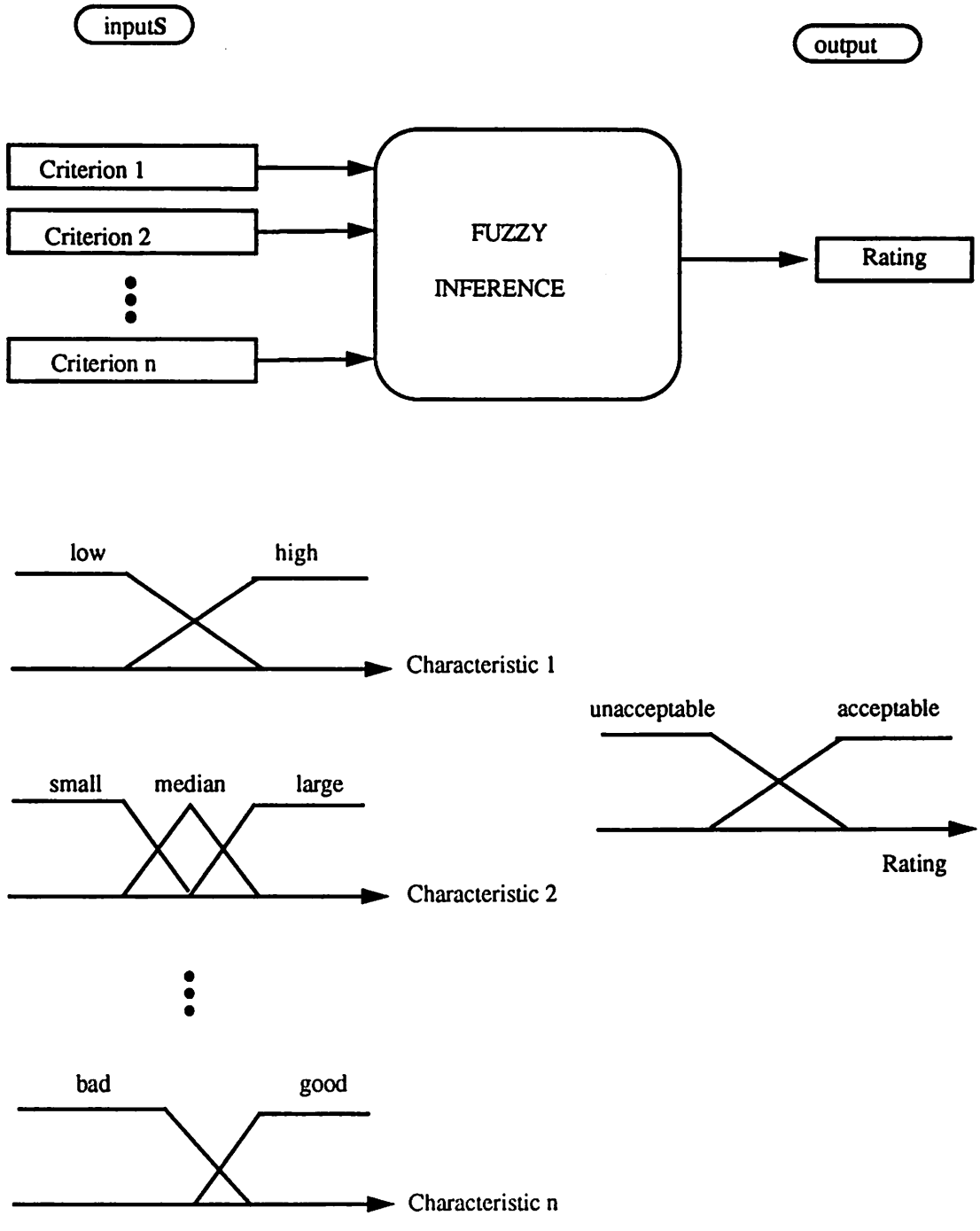


Figure 4: Fuzzy inference for technology mapping

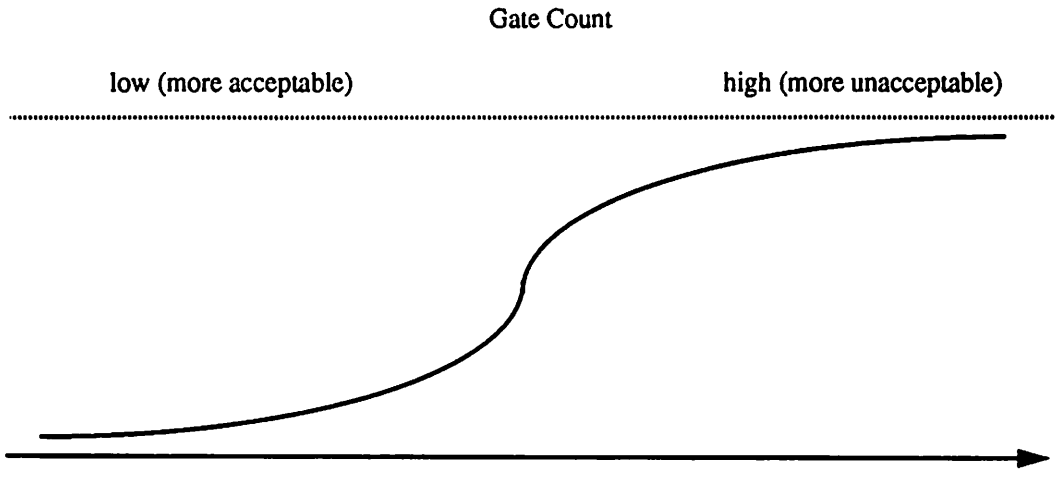


Figure 5: Example of a non-linear membership function

area required, it is clear that a S-curve membership function will be more appropriate for the "gate count" characteristic. This is because the level of acceptability for low gate counts will be high regardless of the actual count. Similarly, the level of acceptability for high gate count will be low regardless of the actual count, since one will be pushing the limits of this particular technology.

3.2.3 A fuzzy algorithm for technology mapping

Although non-linear membership function might be used, we will use linear membership functions in our technology mapping algorithm. But one must recognize that any fuzzy logic algorithm is only as good as its knowledge base. In our case, the knowledge base consists of the fuzzy sets and membership functions derived for each of the technology characteristics. Our fuzzy algorithm is shown graphically in Figure 6.

Using the upper- and lower-limits for each characteristic of a technology, we can set up linear membership functions with two fuzzy sets for each characteristic. The linguistic values for all design variables will be "high" and "low". For example, the gate count characteristic can be described as being "high" or "low", corresponding to the upper- and lower-limits of the characteristic (labelled as LL and UL in Figure 6).

The output rating for each technology will determine the level of acceptability for the implementation of a given design using that particular technology. We have defined the two linguistic variables for the output rating as "acceptable" and "unacceptable"; and we have assigned a numerical rating of 10 as being "acceptable" and a rating of 0 as being "unacceptable".

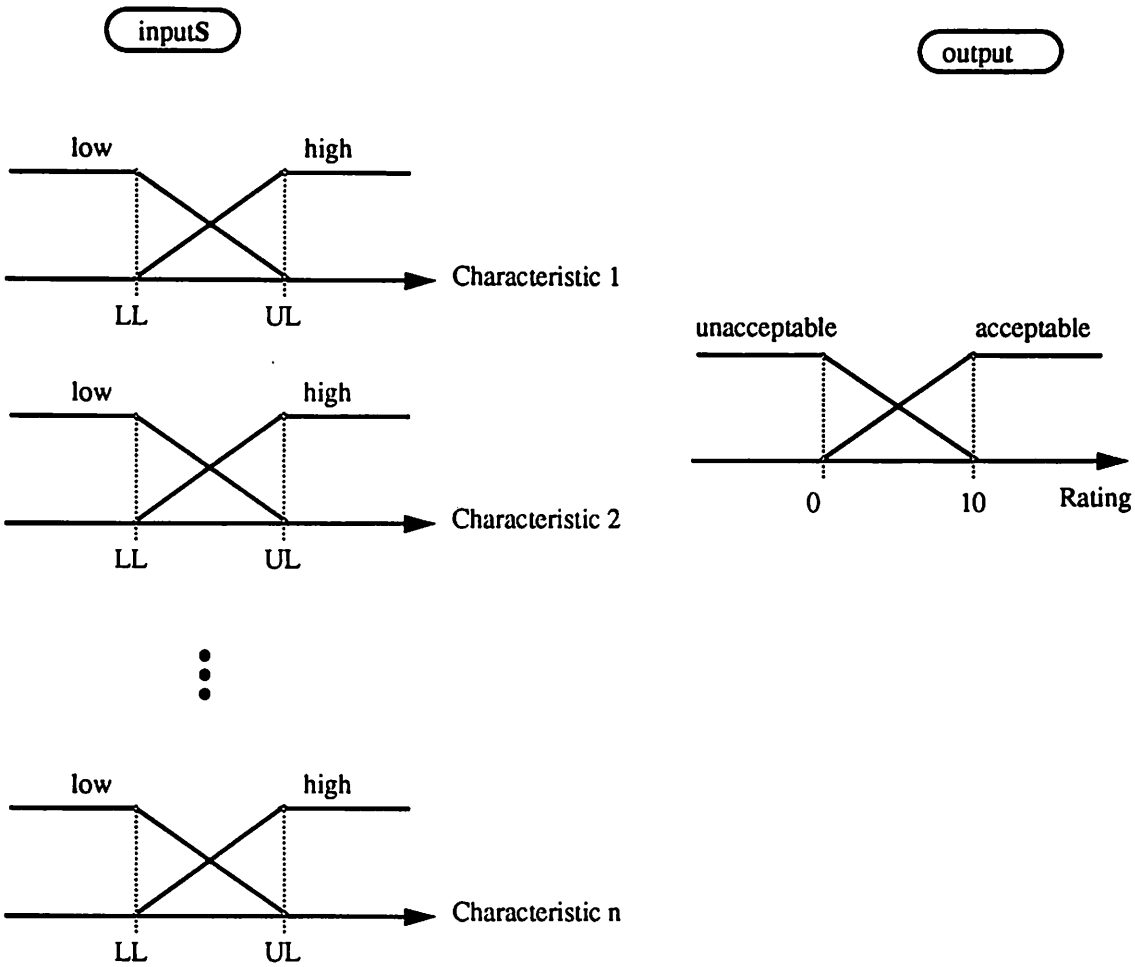


Figure 6: Simple Fuzzy inference algorithm for technology mapping

Now, we need a set of fuzzy rules that we can use to map designs to technologies using the membership functions derived above. Some sample rules follow:

- IF *C1* low AND *C2* low AND *C5* high, THEN rating = acceptable;
- IF *C1* high AND *C2* high AND *C5* low, THEN rating = unacceptable;
- IF *C1* low OR *C3* HIGH, THEN rating = acceptable;
- and so forth...

where *C1*, *C2*, ... stand for *Characteristic 1*, *Characteristic 2*, ...

One can create a rule table and use it along with fuzzy inference in order to determine numerical values for the output ratings. We will illustrate this fuzzy algorithm in the following section when we apply it to technology mapping for some field programmable gate arrays.

3.3 Application Example: Mapping a Design on Commercially Available Field-Programmable Gate Arrays

3.3.1 Field-programmable gate arrays

We have applied the Fuzzy Evaluator for Technology Mapping to a set of twenty-three field-programmable gate arrays (FPGAs) from Signetics Corporation. FPGAs uses the AND/OR/INVERT architecture which allows the custom implementation of Sum of Product logic equations (eg. $D = AB + AC + BC$). Some FPGAs are even fully-implemented Mealy State Machines on a chip with P-terms (the number of ANDs in the FPGA) and State Registers built-in.

3.3.2 Application algorithm

In order to make our application simple, we will use only three characteristics for the FPGAs. These are the gate count, the input/output propagation delay and the power dissipation. Signetics Corporation suggests that each P-term of the FPGA is equivalent to two 8-input AND gates and one 2-input AND gate, and that each OR matrix is equivalent to sixteen 4-input OR gates. Therefore, we conclude that each FPGA holds the following number of gates:

$$\text{Gate Count} = 3 \times \text{Number of P-terms} + 16 \times \text{Number of OR Matrices} \qquad (3\text{-}3)$$

Using the above equation and taking the average power dissipation $\pm 10\%$, we were able to determine the upper- and lower-limits of each characteristic for the twenty-three FPGAs in the *1987 Signetics Programmable Logic Data Manual.* They are shown in Table 4:

Table 4: Characteristics for FPGAs

#	Parts	Gate Count (SSI gates)		I/O Propagation Delay (*ns*)		Power Dissipation (*mW*)	
		LL	UL	LL	UL	LL	UL
	Series 20:						
1	PLS151	0	237	7	25	517.5	632.5
2	PLS153	0	286	20	40	585	715
3	PLS153A	0	286	10	30	585	715
4	PLHS153	0	286	8	20	585	715
5	PLS155	0	471	30	50	675	825
6	PLS159A	0	471	15	35	675	825
7	PLHS18P8A	0	344	8	20	675	825
	Series 24:						
8	PLS161	0	272	20	50	540	660
9	PLS162	0	80	10	30	450	550
10	PLS163	0	144	10	30	540	660
11	PLS167	0	544	72	102	540	660
12	PLS167A	0	544	50	80	540	660
13	PLS168	0	608	72	102	540	660
14	PLS168A	0	608	50	80	540	660
15	PLS173	0	286	10	30	675	825
16	PLS179	0	471	15	35	675	825
17	PLHS473	0	424	10	20	630	770
18	PLC473	0	248	35	60	210	270
	Series 28:						
19	PLS100	0	272	20	50	540	660
20	PLS103	0	144	5	35	540	660
21	PLS105	0	272	72	102	540	660
22	PLS105A	0	272	50	80	540	660
23	PLUS405A	0	320	25	35	855	1045

Using the upper- and lower-limits (written as LL and UL in the table) in Table 4, we were able to determine the fuzzy membership functions needed for our Fuzzy Evaluator.

Based on the fact that lower gate count, higher propagation delay and higher power dissipation are more "acceptable" for implementation, and that higher gate count, lower delay and lower power are more "unacceptable" for implementation, we came up with two sets of fuzzy rules in order to illustrate the difference between AND and OR rules.

The AND rule is as follows:

- IF Gate Count low AND Delay high AND Power high, THEN rating = acceptable;
- IF Gate Count high AND Delay low AND Power low, THEN rating = unacceptable.

The OR rule is as follows:

- IF Gate Count low OR Delay high OR Power high, THEN rating = acceptable;
- IF Gate Count high OR Delay low OR Power low, THEN rating = unacceptable.

The rule tables for the AND and OR rules are shown in Tables 5 and 6, respectively.

Table 5: "AND" rule table

Power low	Gate Count low	Gate Count high
Delay low		AND: Rating = U
Delay high		

Power high	Gate Count low	Gate Count low
Delay low		
Delay high	AND: Rating = A	

Table 6: "OR" rule table

Power low	Gate Count low	Gate Count high
Delay low		OR: Rating = U
Delay high		

Power high	Gate Count low	Gate Count low
Delay low		
Delay high	OR: Rating = A	

where "U" and "A" stand for "Unacceptable" and "Acceptable", respectively.

The ratings for each FPGA are calculated using a software implementation of the above algorithm. This implementation will be discussed in the following section.

3.3.3 Software implementation of the fuzzy technology mapping system

The software application of the fuzzy algorithm discussed in the previous section is done by using the C programming language, SQL (structured query language used for communicating with the data-

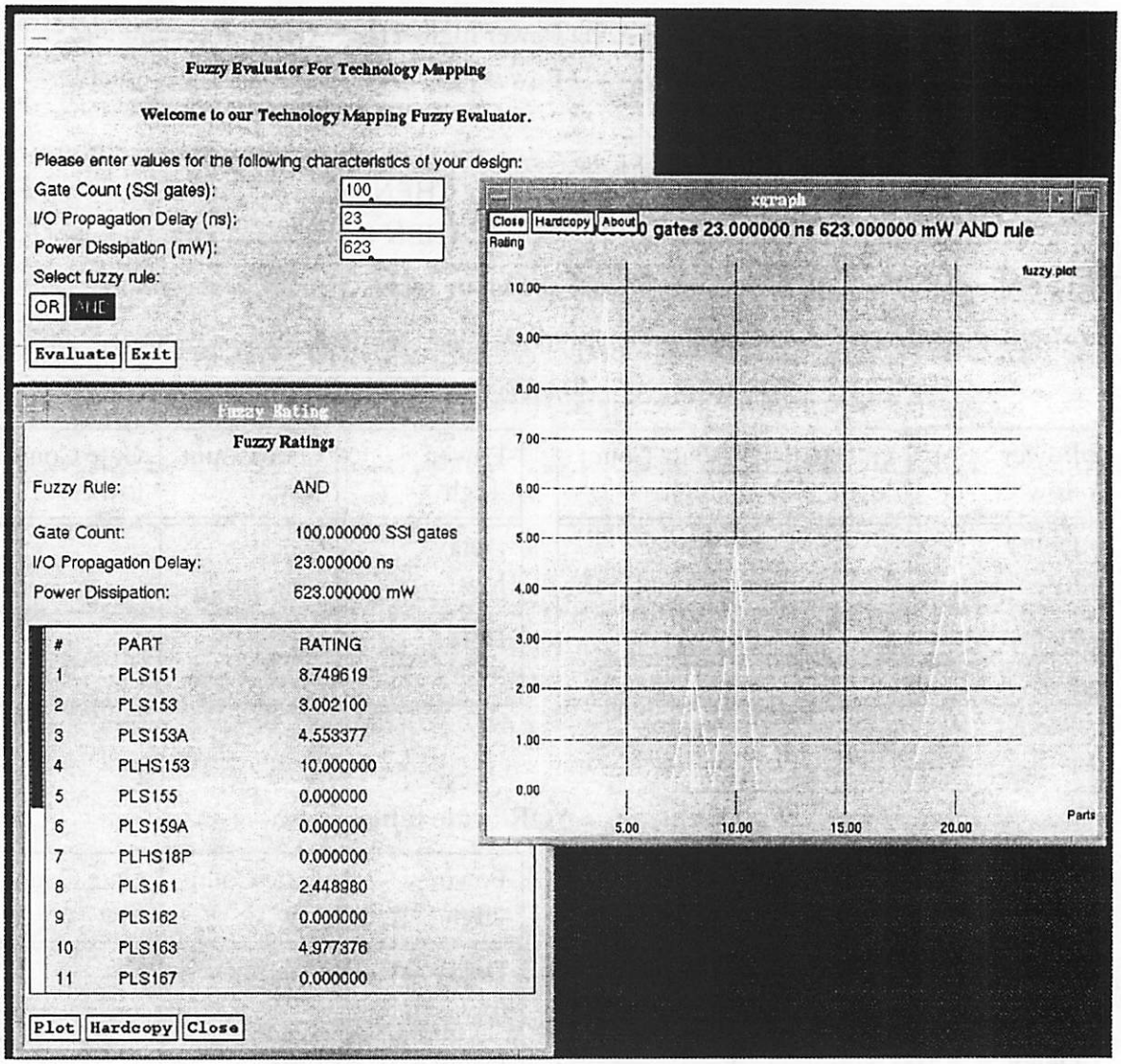


Figure 7: Screen dump of a fuzzy evaluator interface

base) and the X-Windows interface running on the UNIX operating system. The characteristics data of the FPGAs as shown in Table 4 is stored in the Ingres database. The use of the software is rather straightforward. A screen dump of the software interface is shown in Figure 7.

3.3.4 Application results

The fuzzy ratings and plots for three different designs, *A*, *B* and *C*, are shown in Figures 8, 9 and 10 (on pages 17 to 19), respectively. The designs have the following characteristics:

- Design *A*: Gate Count = 100; Delay = 23 *ns*; Power = 623 *mW*;
- Design *B*: Gate Count = 25; Delay = 80 *ns*; Power = 260 *mW*;
- Design *C*: Gate Count = 50; Delay 80 *ns*; Power = 800 *mW*.

By looking at the ratings and the plots, we see that the AND rule is more strict in the sense that FPGAs that do not satisfy a characteristic of the given design will be given a rating of 0. This will ensure that FPGAs not being able to satisfy our design criteria will be eliminated.

The OR rule appears to be less strict; and therefore it is harder for us to distinguish among the FPGAs using the OR rule. However, the advantage of using the OR rule can be seen in Figure 10, as discussed in the last paragraph of this section.

Design *A* is just a typical design and the Fuzzy Evaluator appears to have mapped the design to the various FPGAs well.

Design *B* is a low-power design as it only uses 260 milliwatts of power. There is only one FPGA among the twenty-three in our database that is capable of operating at such low power. It is the PLC473, which is a low-power CMOS chip. As one can see from Figure 9, the AND rule distinctively picks out the PLC473 chip by giving it a fuzzy rating of 10 while eliminating all other FPGAs by giving them fuzzy ratings of 0. The results of the OR rule are not as clear, but the choice is still obvious.

It may appear that the AND rule is superior to the OR rule in our technology mapping scheme. But this is not necessarily true. Design *C* is a high-power design that uses 800 milliwatts of power. Every FPGA in our database is capable of operating at powers lower than the 800 milliwatts except for one. That is the PLUS405A which is a high-power, bipolar, programmable state machine of the Mealy type. As one might except, the AND rule clearly picked out this particular FPGA and gave it a rating of 0. However, because the other characteristics of the design (gate count of 50 and delay of 80 ns) are all acceptable for FPGA implementation, all other FPGAs received ratings of 10. This is not desirable because although we have eliminated the use of the PLUS405A, we are not able to determine which of the remaining FPGA is best suited for our design. However, the OR rule makes this choice clear, as it determine that the PLUS405A is not acceptable and also to find which of the remaining FPGAs is better suited for Design *C*.

4.0 Conclusions

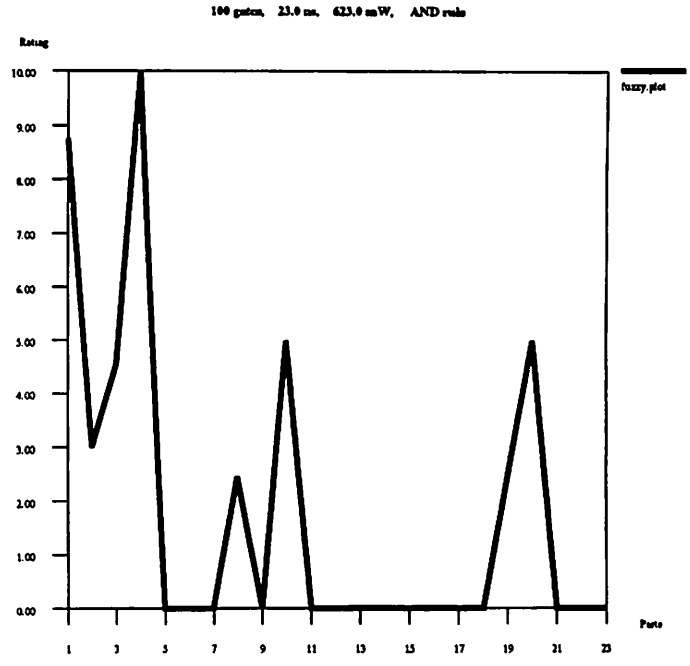
It appears that our Fuzzy Evaluator for Technology Mapping works very well even in its simplest form. Although the AND rule appears to be slightly superior to the OR rule, some combination of the two could prove to be the best. Furthermore, better characteristic fuzzy membership functions can be developed in order to improve the results.

FUZZY RATING:

Fuzzy Rule: AND

Gate Count: 100.000000 SSI gates
I/O Propagation Delay: 23.000000 ns
Power Dissipation: 623.000000 mW

#	PART	RATING
SERIES 20:		
1	PLS151	8.749619
2	PLS153	3.002100
3	PLS153A	4.553377
4	PLHS153	10.000000
5	PLS155	0.000000
6	PLS159A	0.000000
7	PLHS18P	0.000000
SERIES 24:		
8	PLS161	2.448980
9	PLS162	0.000000
10	PLS163	4.977376
11	PLS167	0.000000
12	PLS167A	0.000000
13	PLS168	0.000000
14	PLS168A	0.000000
15	PLS173	0.000000
16	PLS179	0.000000
17	PLHS473	0.000000
18	PLC473	0.000000
SERIES 28:		
19	PLS100	2.448980
20	PLS103	4.977376
21	PLS105	0.000000
22	PLS105A	0.000000
23	PLUS405	0.000000



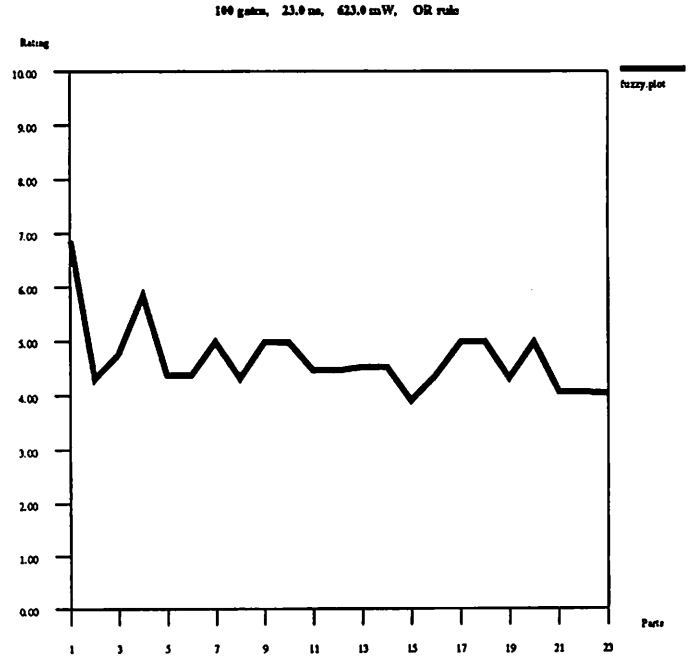
(a) **Using the "AND" rules**

FUZZY RATING:

Fuzzy Rule: OR

Gate Count: 100.000000 SSI gates
I/O Propagation Delay: 23.000000 ns
Power Dissipation: 623.000000 mW

#	PART	RATING
SERIES 20:		
1	PLS151	6.849617
2	PLS153	4.334654
3	PLS153A	4.788877
4	PLHS153	5.855855
5	PLS155	4.406176
6	PLS159A	4.406176
7	PLHS18P	5.000000
SERIES 24:		
8	PLS161	4.345550
9	PLS162	5.000000
10	PLS163	4.989980
11	PLS167	4.493927
12	PLS167A	4.493927
13	PLS168	4.551971
14	PLS168A	4.551971
15	PLS173	3.940678
16	PLS179	4.406176
17	PLHS473	5.000000
18	PLC473	5.000000
SERIES 28:		
19	PLS100	4.345550
20	PLS103	4.989980
21	PLS105	4.088670
22	PLS105A	4.088670
23	PLUS405	4.074074



(b) **Using the "OR" rules**

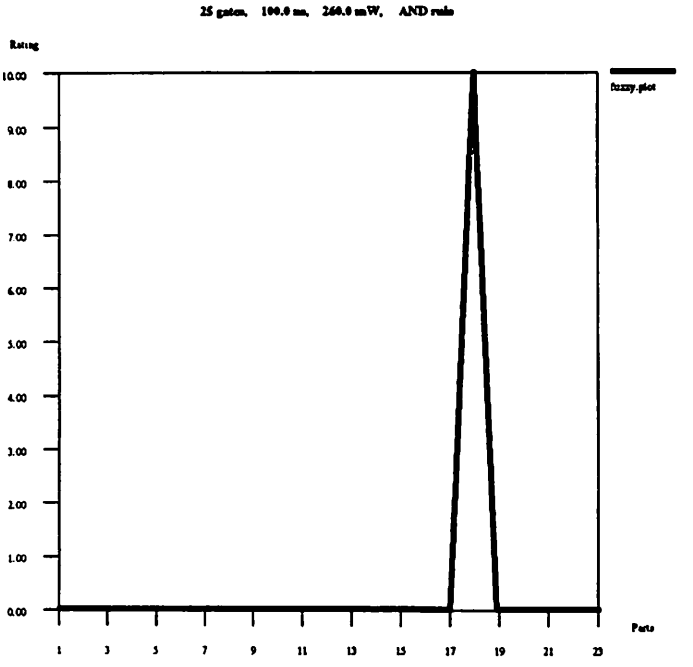
Figure 8: Fuzzy ratings for design A

FUZZY RATING:

Fuzzy Rule: AND

Gate Count: 25.000000 SSI gates
I/O Propagation Delay: 100.000000 ns
Power Dissipation: 260.000000 mW

#	PART	RATING
SERIES 20:		
1	PLS151	0.000000
2	PLS153	0.000000
3	PLS153A	0.000000
4	PLHS153	0.000000
5	PLS155	0.000000
6	PLS159A	0.000000
7	PLHS18P	0.000000
SERIES 24:		
8	PLS161	0.000000
9	PLS162	0.000000
10	PLS163	0.000000
11	PLS167	0.000000
12	PLS167A	0.000000
13	PLS168	0.000000
14	PLS168A	0.000000
15	PLS173	0.000000
16	PLS179	0.000000
17	PLHS473	0.000000
18	PLC473	10.000000
SERIES 28:		
19	PLS100	0.000000
20	PLS103	0.000000
21	PLS105	0.000000
22	PLS105A	0.000000
23	PLUS405	0.000000



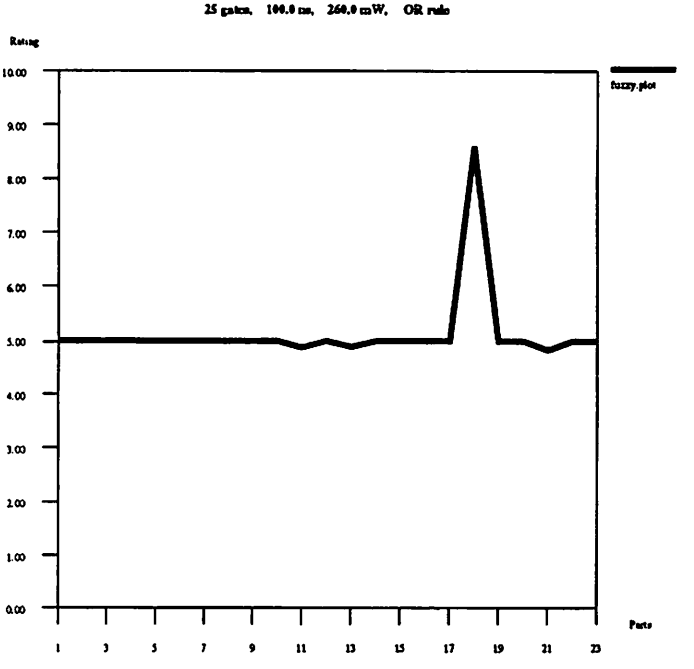
(a) **Using the "AND" rules**

FUZZY RATING:

Fuzzy Rule: OR

Gate Count: 25.000000 SSI gates
I/O Propagation Delay: 100.000000 ns
Power Dissipation: 260.000000 mW

#	PART	RATING
SERIES 20:		
1	PLS151	5.000000
2	PLS153	5.000000
3	PLS153A	5.000000
4	PLHS153	5.000000
5	PLS155	5.000000
6	PLS159A	5.000000
7	PLHS18P	5.000000
SERIES 24:		
8	PLS161	5.000000
9	PLS162	5.000000
10	PLS163	5.000000
11	PLS167	4.882408
12	PLS167A	5.000000
13	PLS168	4.895046
14	PLS168A	5.000000
15	PLS173	5.000000
16	PLS179	5.000000
17	PLHS473	5.000000
18	PLC473	8.571429
SERIES 28:		
19	PLS100	5.000000
20	PLS103	5.000000
21	PLS105	4.827586
22	PLS105A	5.000000
23	PLUS405	5.000000



(b) **Using the "OR" rules**

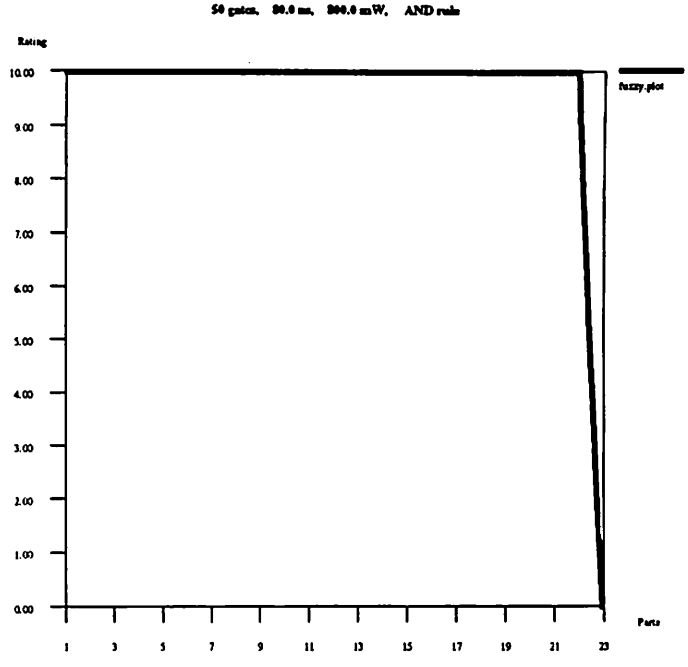
Figure 9: Fuzzy ratings for design B

FUZZY RATING:

Fuzzy Rule: AND

Gate Count: 50.000000 SSI gates
I/O Propagation Delay: 80.000000 ns
Power Dissipation: 800.000000 mW

#	PART	RATING
SERIES 20:		
1	PLS151	10.000000
2	PLS153	10.000000
3	PLS153A	10.000000
4	PLHS153	10.000000
5	PLS155	10.000000
6	PLS159A	10.000000
7	PLHS18P	10.000000
SERIES 24:		
8	PLS161	10.000000
9	PLS162	10.000000
10	PLS163	10.000000
11	PLS167	10.000000
12	PLS167A	10.000000
13	PLS168	10.000000
14	PLS168A	10.000000
15	PLS173	10.000000
16	PLS179	10.000000
17	PLHS473	10.000000
18	PLC473	10.000000
SERIES 28:		
19	PLS100	10.000000
20	PLS103	10.000000
21	PLS105	10.000000
22	PLS105A	10.000000
23	PLUS405	0.000000



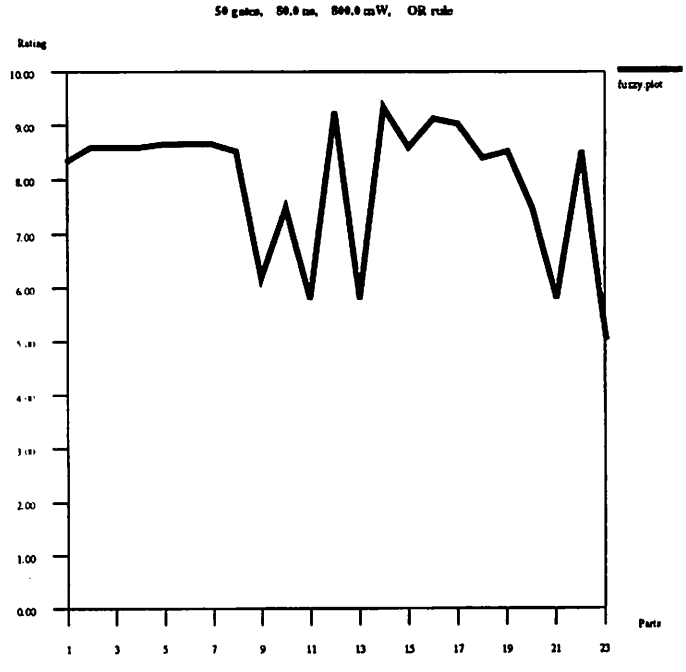
(a) **Using the "AND" rules**

FUZZY RATING:

Fuzzy Rule: OR

Gate Count: 50.000000 SSI gates
I/O Propagation Delay: 80.000000 ns
Power Dissipation: 800.000000 mW

#	PART	RATING
SERIES 20:		
1	PLS151	8.257840
2	PLS153	8.511905
3	PLS153A	8.511905
4	PLHS153	8.511905
5	PLS155	8.571429
6	PLS159A	8.571429
7	PLHS18P	8.571429
SERIES 24:		
8	PLS161	8.447205
9	PLS162	6.153846
10	PLS163	7.422680
11	PLS167	5.769231
12	PLS167A	9.158249
13	PLS168	5.769231
14	PLS168A	9.240121
15	PLS173	8.511905
16	PLS179	9.040307
17	PLHS473	8.945148
18	PLC473	8.322147
SERIES 28:		
19	PLS100	8.447205
20	PLS103	7.422680
21	PLS105	5.769231
22	PLS105A	8.447205
23	PLUS405	5.000000



(b) **Using the "OR" rules**

Figure 10: Fuzzy ratings for design C

5.0 Ideas for Future Work

As pointed out the previous sections, further work can be done in terms of developing better fuzzy rules for the Fuzzy Evaluator. Furthermore, more research must be done into each technology in order to come up with more accurate fuzzy membership functions.

One more interesting thought might be the use of our Fuzzy Evaluator with neural networks in order for the Fuzzy Evaluator to learn from experience by seeing how expert engineers map designs to technologies.

References

[1] Signetics Corporation, "1987 Signetics Programmable Logic Data Manual" (1986);

[2] L.A. Zadeh, "Fuzzy Sets", *Information and Control, Vol.8*, 338-353 (1965);

[3] L.A. Zadeh, "Knowledge Representation in Fuzzy Logic", *IEEE Trans. on Knowledge and Data Engineering, Vol. 1, No. 1, (*1989);

[4] L.A. Zadeh, "Fuzzy Logic", *CS289 Reader (Part II),* University of California at Berkeley (1991);

[5] H.J. Zimmermann, "Fuzzy Set Theory - and Its Applications", Kluwer Nijhoff Publishers (1991);

[6] H.J. Zimmermann, "Fuzzy Sets, Decision Making, and Expert Systems", Kluwer Nijhoff Publishers (1987);

[7] B. Kosko, "Neural Networks and Fuzzy Systems: a dynamical systems approach to machine intelligence", Prentice Hall (1992).

Appendix

A.1 SQL C Code for Our Software Application

```
/*
Copyright (c) 1991 Regents of the University of California

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this specific, written prior permission. The University of California makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

 $Author: hcliu and raymond $
 $Source: /export/mnt/radon1/users/spanos/hcliu/classes/ee244/project $
 $Revision: 1.0 $
 $Date: 12/13/91 $

*/

/*****************************************************************************/

#include <stdio.h>
#include <sys/types.h>

/*
 *      Include files required for all Toolkit programs
 */
#include <X11/Intrinsic.h> /* Intrinsics Definitions */
#include <X11/StringDefs.h> /* Standard Name-String definitions */

#include <X11/Shell.h>

/*
 *      Public include file for widgets we actually use in this file.
 */
#ifdef X11R3
#include <X11/Form.h>
#include <X11/Box.h>
#include <X11/Dialog.h>
#include <X11/Command.h>
#include <X11/AsciiText.h>
#include <X11/Label.h>
#include <X11/MenuButton.h>
#include <X11/Viewport.h>
#include <X11/Toggle.h>
#else /* R4 or later */
#include <X11/Xaw/Form.h>
#include <X11/Xaw/Box.h>
#include <X11/Xaw/Dialog.h>
#include <X11/Xaw/Command.h>
#include <X11/Xaw/AsciiText.h>
#include <X11/Xaw/Label.h>
#include <X11/Xaw/MenuButton.h>
#include <X11/Xaw/Viewport.h>
#include <X11/Xaw/Toggle.h>
#endif /* X11R3 */
```

```
/*
 *      Event bindings translations
 */
static char defaultTranslations[] = "#override\n\
        Ctrl<Key>F:             forward-character() \n\
        Ctrl<Key>B:             backward-character() \n\
        Ctrl<Key>D:             delete-next-character() \n\
        Ctrl<Key>K:             kill-to-end-of-line() \n\
        <Key>Right:             forward-character() \n\
        <Key>Left:              backward-character() \n\
        <Key>Delete:            delete-previous-character() \n\
         <Key>BackSpace:        delete-previous-character() \n\
        <Key>Linefeed:          beginning-of-line() \n\
        <Key>Return:            beginning-of-line() \n\
        <Key>:                  insert-char() \n\
        <Btn1Down>:             select-start() \n\
        <Btn1Motion>:           extend-adjust() \n\
        <Btn1Up>:               extend-end(PRIMARY, CUT_BUFFER0) \n\
        <Btn2Down>:             insert-selection(PRIMARY, CUT_BUFFER0) \n\
        <Btn3Down>:             extend-start() \n\
        <Btn3Motion>:           extend-adjust() \n\
        <Btn3Up>:               extend-end(PRIMARY, CUT_BUFFER0)";
XtTranslations texttranslations;

/*
 *      To handle Ingres SQL database errors.
 */
exec sql include sqlca;

/*
 *      Global variables
 */
FILE *fp, *fopen();
int i;
char buf[256], buf2[256];
float atof(), gc, d, p, rating[23];
Widget topLevel, gcText, dText, pText, orToggle, andToggle, hcDialog;

/*
 *      Global variables for use with the Ingres database
 */
exec sql begin declare section;
        char part[23][20];
exec sql end declare section;

/*****************************************************************************/

/*
 *      Fuzzy logic AND function
 */
float AND(num1, num2, num3)
float num1, num2, num3;
{
        float output;

        if ((num1 <= num2) && (num1 <= num3))
                output = num1;

        if ((num2 <= num1) && (num2 <= num3))
```

```
                output = num2;

        if ((num3 <= num1) && (num3 <= num2))
                output = num3;

        return(output);
}

/*
 *      Fuzzy logic OR function
 */
float OR(num1, num2, num3)
float num1, num2, num3;
{
        float output;

        if ((num1 >= num2) && (num1 >= num3))
                output = num1;

         if ((num2 >= num1) && (num2 >= num3))
                output = num2;

        if ((num3 >= num1) && (num3 >= num2))
                output = num3;

         return(output);
}

/*****************************************************************************/

/*
 *      Exit callback function
 */
void Exit(w, client_data, call_data)
Widget w;
caddr_t client_data, call_data;
{
        /*
         *          Disconnect from database.
         */
        exec sql disconnect;

        exit(0);
}

/*
 *      Plot callback function
 */
void Plot(w, topLevel, call_data)
Widget w, topLevel;
caddr_t call_data;
{
        /*
         *          Writing to fuzzy.plot file.
         */
        fp = fopen("fuzzy.plot", "w");

 if (strcmp(XawToggleGetCurrent(orToggle), "andToggle") == 0)
                sprintf(buf, "TitleText: %f gates %f ns %f mW AND rule\n",
```

```
                        gc, d, p);

if (strcmp(XawToggleGetCurrent(orToggle), "orToggle") == 0)
                sprintf(buf, "TitleText: %f gates %f ns %f mW OR rule\n",
                        gc, d, p);

        fprintf(fp, buf);
        fprintf(fp, "XUnitText: Parts\n");
        fprintf(fp, "YUnitText: Rating\n");

        for (i = 0; i < 23; i++)
                fprintf(fp, "%d\t%f\t%s\n", i+1, rating[i], part[i]);

        fclose(fp);

        /*
         *      Plot fuzzy.plot.
         */
        system("xgraph fuzzy.plot -bar -ly 0,10 &");
}

/*
 *      Close callback function
 */
void Close(w, downshell, call_data)
Widget w, downshell;
caddr_t call_data;
{
        XtPopdown(downshell);
}

/*
 *      Print callback function
 */
void Print(w, parent, call_data)
Widget w, parent;
caddr_t call_data;
{
        String printer;

        /*
         *      Get printer name.
         */
        printer = XawDialogGetValueString(hcDialog);

        /*
         *      Printing results to fuzzy.out output file.
         */
        fp = fopen("fuzzy.out", "w");

        fprintf(fp, "FUZZY RATING:\n\n");

if (strcmp(XawToggleGetCurrent(orToggle), "andToggle") == 0)
                sprintf(buf, "Fuzzy Rule:\t\tAND\n\n");

if (strcmp(XawToggleGetCurrent(orToggle), "orToggle") == 0)
sprintf(buf, "Fuzzy Rule:\t\tOR\n\n");

        fprintf(fp, buf);
```

```
        fprintf(fp, "Gate Count:\t\t%f SSI gates\n", gc);
        fprintf(fp, "I/O Propagation Delay:\t%f ns\n", d);
        fprintf(fp, "Power Dissipation:\t%f mW\n\n", p);

        fprintf(fp, "#\tPART\t\tRATING\n\n");

        fprintf(fp, "SERIES 20:\n");
        for (i = 0; i < 7; i++)
                fprintf(fp, "%d\t%s\t\t%f\n", i+1, part[i], rating[i]);

        fprintf(fp, "\nSERIES 24:\n");
        for (i = 7; i < 18; i++)
                fprintf(fp, "%d\t%s\t\t%f\n", i+1, part[i], rating[i]);

        fprintf(fp, "\nSERIES 28:\n");
        for (i = 18; i < 23; i++)
                fprintf(fp, "%d\t%s\t\t%f\n", i+1, part[i], rating[i]);

        fclose(fp);

        /*
         *      Print fuzzy.out file.
         */
        sprintf(buf, "lpr -P%s fuzzy.out", printer);
        system(buf);

        /*
         *      Pop down Printer shell.
         */
        XtPopdown(parent);
}

/*
 *      Hardcopy callback function
 */
void Hardcopy(w, parent, call_data)
Widget w, parent;
caddr_t call_data;
{
        Position x, y;
        Dimension width, height;
        Arg arg[2];
        Widget Printer, print, cancel;

        /*
         *      Set up widgets.
         */
        Printer = XtCreatePopupShell(
                "Printer",
                transientShellWidgetClass,
                parent,
                NULL,
                0
                );

        hcDialog = XtVaCreateManagedWidget(
                "hcDialog",
                dialogWidgetClass,
```

```
            Printer,
            XtNlabel, "Enter printer name:",
            XtNvalue, "",
            NULL
            );

    print = XtVaCreateManagedWidget(
            "print",
            commandWidgetClass,
            hcDialog,
            XtNlabel, "Print",
            NULL
            );

    cancel = XtVaCreateManagedWidget(
            "cancel",
            commandWidgetClass,
            hcDialog,
            XtNlabel, "Cancel",
            NULL
            );

    /*
     *      Callbacks
     */
    XtAddCallback(print, XtNcallback, Print, Printer);

    XtAddCallback(cancel, XtNcallback, Close, Printer);

    /*
    *       Get coordinates from parent widget.
    */
    i = 0;
    XtSetArg(arg[i], XtNwidth, &width); i++;
    XtSetArg(arg[i], XtNheight, &height); i++;
    XtGetValues(parent, arg, i);

    /*
    *       Translate coordinates from parent widget.
    */
    XtTranslateCoords(parent,
            (Position) width/2,
            (Position) height/2,
            &x, &y);

    /*
    *       Move Printer widget to position given by x, y.
    */
    i = 0;
    XtSetArg(arg[i], XtNx, x); i++;
    XtSetArg(arg[i], XtNy, y); i++;
    XtSetValues(Printer, arg, i);

    /*
    *       Pop up Printer widget.
    */
    XtPopup(Printer, XtGrabNonexclusive);
}
```

```
/*
 *      Evaluate callback function
 */
Evaluate(w, topLevel, call_data)
Widget w, topLevel;
caddr_t call_data;
{
        Arg arg;
        char *ptr;
        float gc_u, gc_a, ngc, d_u, d_a, nd, p_u, p_a, np, r[2], r_sum;
        Widget rate, rate_form, title, ruleLabel, ruleData, gcLabel,
                gcData, dLabel, dData, pLabel, pData, rateview,
                rateview_form, rateviewTitle1, rateviewTitle2,
                rateviewTitle3, rateviewData1[23], rateviewData2[23],
                rateviewData3[23], plot, hardcopy, close;

        /*
         *      Declare variables for use with the Ingres database.
         */
        exec sql begin declare section;
                char command_buf[257];
                float gc_ll, gc_ul, d_ll, d_ul, p_ll, p_ul;
        exec sql end declare section;

        /*
         *      Read characteristic values off screen.
         */
        XtSetArg(arg, XtNstring, &ptr);
        XtGetValues(gcText, &arg, 1);
        gc = atof(ptr);

        XtSetArg(arg, XtNstring, &ptr);
        XtGetValues(dText, &arg, 1);
        d = atof(ptr);

        XtSetArg(arg, XtNstring, &ptr);
        XtGetValues(pText, &arg, 1);
        p = atof(ptr);

        /*
         *      Evaluating
         */

        /*
         *      Retrieving part characteristics from database.
         */
        exec sql declare cur cursor for
                select part, gate_count_ll, gate_count_ul,
                        delay_ll, delay_ul, power_ll, power_ul
                from signetics;
        exec sql open cur;
        for (i = 0; i < 23; i++) {
                exec sql fetch cur
                        into :part[i], :gc_ll, :gc_ul, :d_ll,
                        :d_ul, :p_ll, :p_ul;
                /*
                 * Calculating belongness of each characteristic.
                 */
                ngc = gc;
```

```
                nd = d;
                np = p;

                if (gc < gc_ll)
                        ngc = gc_ll;

                if (gc > gc_ul)
                        ngc = gc_ul;

                if (d < d_ll)
                        nd = d_ll;

                if (d > d_ul)
                        nd = d_ul;

                if (p < p_ll)
                        np = p_ll;

                if (p > p_ul)
                        np = p_ul;

                gc_u = ngc/gc_ul;
                gc_a = (gc_ul-ngc)/gc_ul;

                d_u = (d_ul-nd)/(d_ul-d_ll);
                d_a = (nd-d_ll)/(d_ul-d_ll);

                p_u = (p_ul-np)/(p_ul-p_ll);
                p_a = (np-p_ll)/(p_ul-p_ll);

                /*
                 *      Using fuzzy inference to determine rating.
                 */
                if (strcmp(XawToggleGetCurrent(orToggle), "andToggle") == 0) {
                        r[0] = AND(gc_u, d_u, p_u);
                        r[1] = AND(gc_a, d_a, p_a);
                }

                if (strcmp(XawToggleGetCurrent(orToggle), "orToggle") == 0) {
                        r[0] = OR(gc_u, d_u, p_u);
                        r[1] = OR(gc_a, d_a, p_a);
}

                r_sum = r[0] + r[1];

                if (r_sum == 0)
                        r_sum = 1;

                rating[i] = (r[0]*0.00 + r[1]*10.0) / r_sum;
        }
        exec sql close cur;

        /*
         *      Set up widgets.
         */
        rate = XtCreatePopupShell(
                "Fuzzy Rating",
                applicationShellWidgetClass,
                topLevel,
```

```
            NULL,
            0
            );

rate_form = XtVaCreateManagedWidget(
            "rate_form",
            formWidgetClass,
            rate,
            XtNdefaultDistance, 5,
            NULL
            );

title = XtVaCreateManagedWidget(
            "title",
            labelWidgetClass,
            rate_form,
            XtNlabel, "Fuzzy Ratings",
            XtNwidth, 500,
            XtNborderWidth, 0,
            NULL
            );

 ruleLabel = XtVaCreateManagedWidget(
            "ruleLabel",
            labelWidgetClass,
            rate_form,
            XtNlabel, "Fuzzy Rule:",
            XtNborderWidth, 0,
            XtNwidth, 250,
            XtNfromVert, title,
            XtNvertDistance, 20,
            XtNjustify, XtJustifyLeft,
            NULL
             );

if (strcmp(XawToggleGetCurrent(orToggle), "andToggle") == 0)
            sprintf(buf, "AND");

 if (strcmp(XawToggleGetCurrent(orToggle), "orToggle") == 0)
            sprintf(buf, "OR");

ruleData = XtVaCreateManagedWidget(
            "ruleData",
            labelWidgetClass,
            rate_form,
            XtNlabel, buf,
            XtNborderWidth, 0,
            XtNfromVert, title,
            XtNvertDistance, 20,
            XtNfromHoriz, ruleLabel,
            XtNjustify, XtJustifyLeft,
            NULL
            );

gcLabel = XtVaCreateManagedWidget(
            "gcLabel",
            labelWidgetClass,
            rate_form,
            XtNlabel, "Gate Count:",
```

```
            XtNborderWidth, 0,
            XtNwidth, 250,
            XtNfromVert, ruleLabel,
            XtNvertDistance, 20,
            XtNjustify, XtJustifyLeft,
            NULL
            );

    sprintf(buf, "%f SSI gates", gc);
    gcData = XtVaCreateManagedWidget(
            "gcData",
            labelWidgetClass,
            rate_form,
            XtNlabel, buf,
            XtNborderWidth, 0,
            XtNfromVert, ruleLabel,
            XtNvertDistance, 20,
            XtNfromHoriz, ruleLabel,
            XtNjustify, XtJustifyLeft,
            NULL
            );

    dLabel = XtVaCreateManagedWidget(
            "dLabel",
            labelWidgetClass,
            rate_form,
            XtNlabel, "I/O Propagation Delay:",
            XtNborderWidth, 0,
            XtNwidth, 250,
            XtNfromVert, gcLabel,
            XtNjustify, XtJustifyLeft,
            NULL
            );

    sprintf(buf, "%f ns", d);
    dData = XtVaCreateManagedWidget(
            "dData",
            labelWidgetClass,
            rate_form,
            XtNlabel, buf,
            XtNborderWidth, 0,
            XtNfromVert, gcLabel,
            XtNfromHoriz, dLabel,
            XtNjustify, XtJustifyLeft,
            NULL
            );

    pLabel = XtVaCreateManagedWidget(
            "pLabel",
            labelWidgetClass,
            rate_form,
            XtNlabel, "Power Dissipation:",
            XtNborderWidth, 0,
            XtNwidth, 250,
            XtNfromVert, dLabel,
            XtNjustify, XtJustifyLeft,
            NULL
            );
```

```
sprintf(buf, "%f mW", p);
pData = XtVaCreateManagedWidget(
        "pData",
        labelWidgetClass,
        rate_form,
        XtNlabel, buf,
        XtNborderWidth, 0,
        XtNfromVert, dLabel,
        XtNfromHoriz, pLabel,
        XtNjustify, XtJustifyLeft,
        NULL
        );

rateview = XtVaCreateManagedWidget(
        "rateview",
        viewportWidgetClass,
        rate_form,
        XtNallowVert, TRUE,
        XtNforceBars, TRUE,
        XtNuseBottom, TRUE,
        XtNwidth, 495,
        XtNheight, 350,
        XtNfromVert, pLabel,
        XtNvertDistance, 20,
        NULL
        );

rateview_form = XtVaCreateManagedWidget(
        "rateview_form",
        formWidgetClass,
        rateview,
        XtNdefaultDistance, 5,
        NULL
        );

rateviewTitle1 = XtVaCreateManagedWidget(
        "rateviewTitle1",
        labelWidgetClass,
        rateview_form,
        XtNlabel, "#",
        XtNborderWidth, 0,
        XtNwidth, 50,
        XtNjustify, XtJustifyLeft,
        NULL
        );

rateviewTitle2 = XtVaCreateManagedWidget(
        "rateviewTitle2",
        labelWidgetClass,
        rateview_form,
        XtNlabel, "PART",
        XtNborderWidth, 0,
        XtNwidth, 150,
        XtNfromHoriz, rateviewTitle1,
        XtNjustify, XtJustifyLeft,
        NULL
        );

rateviewTitle3 = XtVaCreateManagedWidget(
```

```
            "rateviewTitle3",
            labelWidgetClass,
            rateview_form,
            XtNlabel, "RATING",
            XtNborderWidth, 0,
            XtNwidth, 200,
            XtNfromHoriz, rateviewTitle2,
            XtNjustify, XtJustifyLeft,
            NULL
             );

rateviewData1[0] = XtVaCreateManagedWidget(
            "rateviewData10",
            labelWidgetClass,
            rateview_form,
            XtNlabel, "1",
            XtNborderWidth, 0,
            XtNwidth, 50,
            XtNfromVert, rateviewTitle1,
            XtNjustify, XtJustifyLeft,
            NULL
            );

rateviewData2[0] = XtVaCreateManagedWidget(
            "rateviewData20",
            labelWidgetClass,
            rateview_form,
            XtNlabel, part[0],
            XtNborderWidth, 0,
            XtNwidth, 150,
            XtNfromVert, rateviewTitle1,
            XtNfromHoriz, rateviewData1[0],
            XtNjustify, XtJustifyLeft,
            NULL
             );

sprintf(buf, "%f", rating[0]);
rateviewData3[0] = XtVaCreateManagedWidget(
            "rateviewData30",
            labelWidgetClass,
            rateview_form,
            XtNlabel, buf,
            XtNborderWidth, 0,
            XtNwidth, 200,
            XtNfromVert, rateviewTitle1,
            XtNfromHoriz, rateviewData2[0],
            XtNjustify, XtJustifyLeft,
            NULL
            );

for (i = 1; i < 23; i++) {
            sprintf(buf, "rateviewData1%d", i);
            sprintf(buf2, "%d", i+1);
            rateviewData1[i] = XtVaCreateManagedWidget(
                        buf,
                        labelWidgetClass,
                        rateview_form,
                        XtNlabel, buf2,
                        XtNborderWidth, 0,
```

```
                    XtNwidth, 50,
                    XtNfromVert, rateviewData1[i-1],
                    XtNjustify, XtJustifyLeft,
                    NULL
                    );

        sprintf(buf, "rateviewData2%d", i);
        rateviewData2[i] = XtVaCreateManagedWidget(
                    buf,
                    labelWidgetClass,
                    rateview_form,
                    XtNlabel, part[i],
                    XtNborderWidth, 0,
                    XtNwidth, 150,
                    XtNfromVert, rateviewData1[i-1],
                    XtNfromHoriz, rateviewData1[i],
                    XtNjustify, XtJustifyLeft,
                    NULL
                    );

        sprintf(buf, "rateviewData3%d", i);
        sprintf(buf2, "%f", rating[i]);
        rateviewData1[i] = XtVaCreateManagedWidget(
                    buf,
                    labelWidgetClass,
                    rateview_form,
                    XtNlabel, buf2,
                    XtNborderWidth, 0,
                    XtNwidth, 200,
                    XtNfromVert, rateviewData1[i-1],
                    XtNfromHoriz, rateviewData2[i],
                    XtNjustify, XtJustifyLeft,
                    NULL
                     );
}

plot = XtVaCreateManagedWidget(
        "plot",
        commandWidgetClass,
        rate_form,
        XtNlabel, "Plot",
        XtNfromVert, rateview,
        XtNvertDistance, 20,
        NULL
        );

hardcopy = XtVaCreateManagedWidget(
        "hardcopy",
        commandWidgetClass,
        rate_form,
        XtNlabel, "Hardcopy",
        XtNfromVert, rateview,
        XtNvertDistance, 20,
        XtNfromHoriz, plot,
        NULL
        );

close = XtVaCreateManagedWidget(
        "close",
```

```
                commandWidgetClass,
                rate_form,
                XtNlabel, "Close",
                XtNfromVert, rateview,
                XtNvertDistance, 20,
                XtNfromHoriz, hardcopy,
                NULL
                );

        /*
         *      Callbacks
         */
        XtAddCallback(plot, XtNcallback, Plot, topLevel);

        XtAddCallback(hardcopy, XtNcallback, Hardcopy, rate);

        XtAddCallback(close, XtNcallback, Close, rate);

        /*
         *      Pop up rate application shell.
         */
        XtPopup(rate, XtGrabNone);
}

/*****************************************************************************/

/*
 *      Main function
 */
main(argc, argv)
int argc;
char *argv[];
{
        Widget fuzzy, fuzzy_form, title, welcome, instruction, gcLabel, dLabel,
                pLabel, ruleLabel, evaluate, exit;

        /*
         *      Connect to the EE244 database.
         */
        exec sql connect ee244;

        /*
         *      Set up widgets.
         */
        topLevel = XtInitialize(
                "topLevel",
                "fuzzy.app",
                NULL,
                0,
                &argc,
                argv
                );

        texttranslations = XtParseTranslationTable(defaultTranslations);

        fuzzy = XtCreatePopupShell(
                "Fuzzy",
                applicationShellWidgetClass,
                topLevel,
```

```
        NULL,
        0
        );

fuzzy_form = XtVaCreateManagedWidget(
        "fuzzy_form",
        formWidgetClass,
        fuzzy,
        NULL
        );

title = XtVaCreateManagedWidget(
        "title",
        labelWidgetClass,
        fuzzy_form,
        XtNlabel, "Fuzzy Evaluator For Technology Mapping",
        XtNwidth, 600,
        XtNborderWidth, 0,
        NULL
        );

welcome = XtVaCreateManagedWidget(
        "welcome",
        labelWidgetClass,
        fuzzy_form,
        XtNlabel, "Welcome to our Technology Mapping Fuzzy Evaluator.",
        XtNwidth, 600,
        XtNborderWidth, 0,
        XtNfromVert, title,
        XtNvertDistance, 20,
        NULL
        );

instruction = XtVaCreateManagedWidget(
        "instruction",
        labelWidgetClass,
        fuzzy_form,
        XtNlabel, "Please enter values for the following characteristics of your design:",
        XtNborderWidth, 0,
        XtNfromVert, welcome,
        XtNjustify, XtJustifyLeft,
        XtNvertDistance, 20,
        NULL
        );

gcLabel = XtVaCreateManagedWidget(
        "gcLabel",
        labelWidgetClass,
        fuzzy_form,
        XtNlabel, "Gate Count (SSI gates):",
        XtNborderWidth, 0,
        XtNwidth, 300,
        XtNfromVert, instruction,
        XtNjustify, XtJustifyLeft,
        NULL
        );

gcText = XtVaCreateManagedWidget(
        "gcText",
```

```
            asciiTextWidgetClass,
            fuzzy_form,
            XtNeditType, "edit",
            XtNfromVert, instruction,
            XtNfromHoriz, gcLabel,
            XtNtranslations, texttranslations,
            NULL
            );

dLabel = XtVaCreateManagedWidget(
            "dLabel",
            labelWidgetClass,
            fuzzy_form,
            XtNlabel, "I/O Propagation Delay (ns):",
            XtNborderWidth, 0,
            XtNwidth, 300,
            XtNfromVert, gcLabel,
            XtNjustify, XtJustifyLeft,
            NULL
            );

dText = XtVaCreateManagedWidget(
            "dText",
            asciiTextWidgetClass,
            fuzzy_form,
            XtNeditType, "edit",
            XtNfromVert, gcLabel,
            XtNfromHoriz, dLabel,
            XtNtranslations, texttranslations,
            NULL
            );

pLabel = XtVaCreateManagedWidget(
            "pLabel",
            labelWidgetClass,
            fuzzy_form,
            XtNlabel, "Power Dissipation (mW):",
            XtNborderWidth, 0,
            XtNwidth, 300,
            XtNfromVert, dLabel,
            XtNjustify, XtJustifyLeft,
            NULL
            );

pText = XtVaCreateManagedWidget(
            "pText",
            asciiTextWidgetClass,
            fuzzy_form,
            XtNeditType, "edit",
            XtNfromVert, dLabel,
            XtNfromHoriz, pLabel,
            XtNtranslations, texttranslations,
            NULL
            );

ruleLabel = XtVaCreateManagedWidget(
            "ruleLabel",
            labelWidgetClass,
            fuzzy_form,
```

```
            XtNlabel, "Select fuzzy rule:",
            XtNborderWidth, 0,
            XtNfromVert, pLabel,
            XtNjustify, XtJustifyLeft,
            NULL
            );

orToggle = XtVaCreateManagedWidget(
            "orToggle",
            toggleWidgetClass,
            fuzzy_form,
            XtNlabel, "OR",
            XtNfromVert, ruleLabel,
            NULL
            );

andToggle = XtVaCreateManagedWidget(
            "andToggle",
            toggleWidgetClass,
            fuzzy_form,
            XtNlabel, "AND",
            XtNfromVert, ruleLabel,
            XtNfromHoriz, orToggle,
            XtNradioGroup, orToggle,
            NULL
            );

evaluate = XtVaCreateManagedWidget(
            "evaluate",
            commandWidgetClass,
            fuzzy_form,
            XtNlabel, "Evaluate",
            XtNfromVert, orToggle,
            XtNvertDistance, 20,
            NULL
            );

exit = XtVaCreateManagedWidget(
            "exit",
            commandWidgetClass,
            fuzzy_form,
            XtNlabel, "Exit",
            XtNfromVert, orToggle,
            XtNvertDistance, 20,
            XtNfromHoriz, evaluate,
            NULL
            );

/*
 *          Callbacks
 */
XtAddCallback(evaluate, XtNcallback, Evaluate, topLevel);

XtAddCallback(exit, XtNcallback, Exit, 0);

/*
 *          Pop up fuzzy application shell
 */
XtPopup(fuzzy, XtGrabNone);
```

```
    /*
     *      Loop for events.
     */
    XtMainLoop();
}
```

A.2 X-Windows Resource Application Code

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Berkeley Computer-Aided Manufacturing (BCAM) System
! bcam.app - BCAM main program applications file
!
! Copyright (c) 1991 Regents of the University of California
!
! Permission to use, copy, modify, and distribute this software and its documentation for any purpose and ! without fee is hereby granted, provided that the above copyright notice appear in all copies and that both ! that copyright notice and this permission notice appear in supporting documentation, and that the name of ! the University of California not be used in advertising or publicity pertaining to distribution of the software ! without specific, written prior permission. The University of California makes no representations about the ! suitability of this software for any purpose. It is provided "as is" without express or implied warranty.
!
! $Author: hcliu and raymond $
! $Source: /export/mnt/radon1/users/spanos/hcliu/classes/ee244/project $
! $Revision: 1.0 $
! $Date: 12/13/91 $
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! Appearance resources
!
*background: white
*font: -*-helvetica-medium-r-normal--17-*-100-100-*-*-iso8859-1
!
*Form.background: gray
*Box.background: gray
*Dialog.background: gray
*Label.background: gray
*Command.background: lightgray
*Toggle.background: lightgray
*Command.font: -*-courier-bold-r-normal--17-*-100-100-*-*-iso8859-1
*Text.background: white
!
*title.font: -*-times-bold-r-normal--17-*-100-100-*-*-iso8859-1
*welcome.font: -*-times-bold-r-normal--17-*-100-100-*-*-iso8859-1
```