5-2001

Development of a simulation backplane with dynamic configurability

Lloyd Gabriel Clonts

Recommended Citation

Clonts, Lloyd Gabriel, "Development of a simulation backplane with dynamic configurability. " PhD diss., University of Tennessee, 2001.
https://trace.tennessee.edu/utk_graddiss/8479

To the Graduate Council:

I am submitting herewith a dissertation written by Lloyd Gabriel Clonts entitled "Development of a simulation backplane with dynamic configurability." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Electrical Engineering.

Donald W. Bouldin, Major Professor

We have read this dissertation and recommend its acceptance:

J. M. Rochelle, E. J. Kennedy, D. F. Newport, W. R. Hamel

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a dissertation written by Lloyd G. Clonts entitled "Development of a Simulation Backplane with Dynamic Configurability ". I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctorate of Philosophy with a major in Electrical Engineering.

Donald W. Bouldin

Dr. Donald W. Bouldin, Major Professor

We have read this dissertation and recommend its acceptance:

William B. Hamel

James M. Rochelle

Accepted for the Council:

Interim Vice Provost

and Dean of the Graduate School

Development of a Simulation Backplane with Dynamic Configurability

A Dissertation

Presented for the

Doctorate of Philosophy

Degree

The University of Tennessee, Knoxville

Lloyd G. Clonts

May 2001

ACKNOWLEDGMENTS

I would like to thank my major professor, Dr. Donald W. Bouldin for his guidance, patience, and reassurance during this dissertation. I would also like to thank the other committee members, Dr. J.M. Rochelle, Dr. E. J. Kennedy, Dr. D.F. Newport, and Dr. W.R. Hamel for their comments and encouragement. I would also express my deepest gratitude to my group leader, W.L. Bryan, at ORNL for his guidance, patience, and help in limiting the scope of the topic. Finally, I would like to thank my mother, Elaine Clonts, for her encouragement and for putting-up with my endless complaining during the completion of this degree, and to my grandparents, Donald and Barbara Ownby.

ABSTRACT

The subject of this thesis was the development a simulation backplane for coupling an electrical simulator with a mechanical finite-element simulator although the backplane is applicable to any simulator with the proper architecture. To tradeoff performance and accuracy during the analysis, a dynamic configuration option was included, where different simulators and models in a simulator can be switched in and out during the analysis. This option provided the designer with the flexibility to analyze all details of simulations from different modeling representations to optimizing the simulation performance within the same analysis. This backplane was able to transverse multiple coupling architectures to be a configuration tool for simulating hybrid environments with dynamic changes. In this work, the dynamic configurability procedure was outlined with other issues and procedures for coupling multiple simulators.

To improve upon the basic coupling process, different interface configurations were examined in different casualty-based formats. Specifically, conventional interface combinations were compared under different sensitivity calculation criteria and methods to find the interface or combination with the best iteration efficiency and convergence. The iteration efficiency was typically determined by the sensitivity calculation options while the convergence was determined by the interface combination between the simulators and by the backplane initialization sequence. The optimum convergence for any conventional interface combination was 93%. From these analyses, a dynamic-interface configuration procedure was developed based on coupling conditions and variable causality to identify the interface configuration with the best chances of convergence. A tiered dynamic interface procedure had convergence of 95% and equivalent iteration efficiency with the conventional interfaces. Finally, a flow correction method using behavioral models was examined, where a predictor and corrector process was implemented that allowed more versatility in the coupling process and improved initialization compared to the other interfaces. The flow correction process had a convergence of 93% tested over behavioral models with different accuracy constraints. A sensitivity calculation problem limited the success of the flow correction procedure and caused the iteration inefficiency to be two times larger than the other procedures.

TABLE OF CONTENTS

Chapters Page

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AIC	Absolute Iteration Comparison
ASIC	Application Specific Integrated Circuits
ASIS	Application Specific Integrated Systems
CAD	Computer Aided Design
CAE	Computer Aided Engineering
DOF	Degrees of Freedom
E2E	Electrical to Electrical simulator
E2FE	Electrical to Finite Element simulator
FE	Finite Element
FEA	Finite Element Analysis
FEM	Finite Element Method
II	Iteration Inefficiency
NA	Nodal Analysis
MEMS	Micro-Electro-Mechanical Systems
MNA	Modified Nodal Analysis
PVM	Parallel Virtual Machine
RPC	Remote Procedure Call
SB	Switch Back

Chapter 1. Introduction

Micro-Electro-Mechanical Systems (MEMS) are difficult to design and simulate due to their multidisciplinary nature. MEMS integrate different engineering disciplines where batch processing can reduce costs and enhance features [3,4]. The disciplines [1,2] used in MEMS are process technology (micromachining), electronics, mechanics, packaging, thermodynamics, etc, as shown in Figure 1-1. These different disciplines in MEMS can be designed, modeled, and simulated independently, but few environments consider all disciplines simultaneously as a single problem at the system-level using different discipline simulators. Assuming each discipline has found the optimum iteration algorithm(s) and modeling/abstraction method(s) for a particular type of design, then the problem becomes how to integrate the different discipline simulators into an efficient system-level concept.

To narrow the scope of this work, this research focused on integrating an electrical analog-type simulator and a mechanical Finite Element (FE) simulator. An electrical simulator is used for lumped-sum devices with well-defined behavior while a FE simulator is used to analyze movable devices using a known structure and material models with multiple degrees of freedom (DOF). The capabilities of the individual simulator determine how the simulator can be used to simulate MEMS, since each simulator offers different performance, accuracy, and implementation tradeoffs. The first requirement is to define the basic structure of a generic simulator and the modeling issues between the different simulators. These issues are then used to examine the different simulator architectures for integrating disciplines (not just MEMS). The different architectures are independent

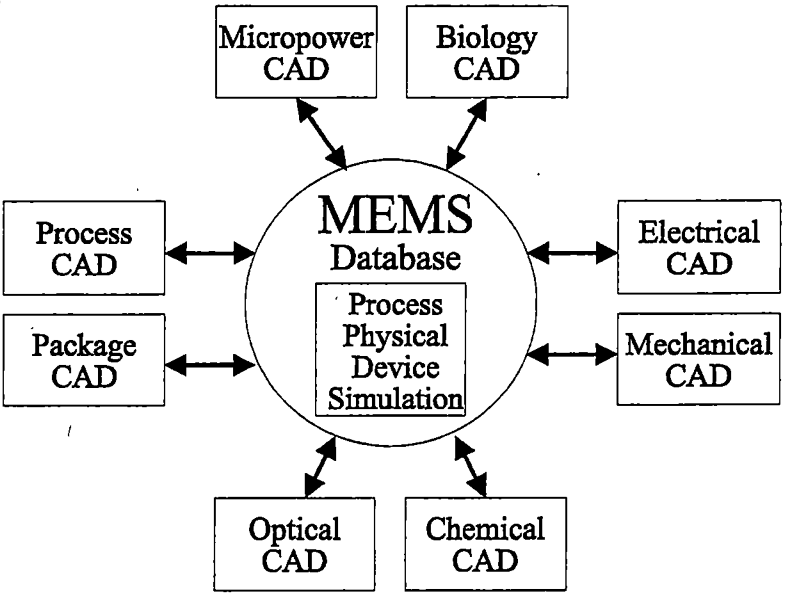


Figure 1-1. Database and simulation overview for MEMS

simulators, coupled simulators, and a unified simulator.

1.1 A Generic Simulator and the Modeling Issues

The common causes for any design failure are a lack of accurate models and insufficient simulation of the design over different conditions. These modeling issues translate into a tradeoff between defining the simulation accuracy to meet the design specifications [6,7] and performing a sufficient number of simulations to validate the design under a reasonable set of different conditions. Unfortunately, simulation accuracy and performance cannot typically be optimized simultaneously. To make tradeoffs between accuracy and performance, the designer can use different levels of abstraction (design representations or modeling techniques) or different solver algorithms. These two elements represent the fundamental aspects of a simulator, which are a solver(s), equation matrix, and a modeler as shown in Figure 1-2.

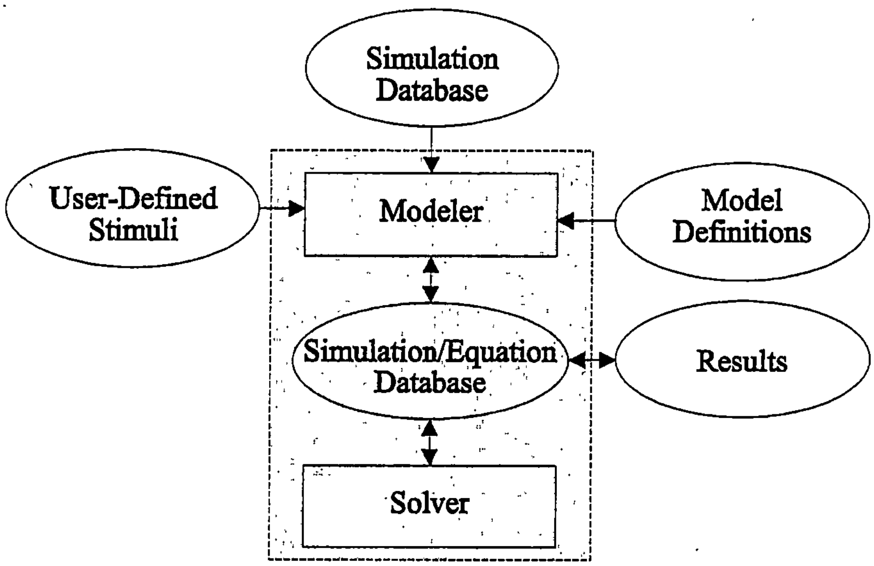


Figure 1-2. The structure of a simulator

The role of the solver is to calculate a solution from the equation matrix. Each solver can use a variety of different iteration techniques to reach convergence because certain solver algorithms or combinations of algorithms work better for specific problem types and sizes. In most cases, the solver algorithms are hidden from the designer, but the noticeable differences between simulators are improvements in performance or undesired convergence failures. Primarily, solvers have a greater contribution on simulation performance and solution convergence than on simulation accuracy, although the timestep and iteration control in the solver can influence the accuracy of the analysis.

The modeler directly determines the devices that the simulator can recognize and **translates** the devices definitions from the *different syntax* databases into the equation matrix. The modeler dynamically defines the equation matrix values for a device based on the device's physical parameters, governing behavioral equations, and the present solution.

Most simulators are built around predefined component definitions to simplify the modeling of a design, so the interfaces are very direct and optimized for solving a particular problem. With the optimized interface, the designer must only specify the essential physical parameters into the predefined models to perform an analysis. Using the predefined modeling definitions, a designer does not have to specify the equations governing a component's behavior or verify the model.

Solvers are essential to the simulation process, but the modeling can have the largest impact on both accuracy and performance. The modeling defines the level of abstraction and the different calculation modes for an element in the simulation. Different calculation modes can be accessed within the same simulator, but the level of abstraction often requires a different type of simulator (domain or discipline). Therefore, another reason to integrate simulators is to have full access to the abstraction capabilities of the different simulators and solver algorithms in some cases.

1.1.1 Levels of Abstraction

The level of abstraction can range from a physical low-level to a functional or conceptual high-level. The lower abstraction levels typically have greater accuracy and slower performance by focusing on device and structural behavior. A low-level representation requires the simulator to solve a large number of equations for each primitive element. The higher abstraction levels have faster performance and typically less accuracy by examining modules that describe behavior instead of the interaction between the multiple elements comprising the device or module. A high-level representation minimizes the number of equations to be solved or limits the number of possible solutions to a finite number of states. The higher abstraction levels are generally used at the beginning of the design cycle to make system-level tradeoffs and the lower abstraction levels are used at the end of the design cycle for verification. Unfortunately, no simple rule exists for defining the level of abstraction except to simulate as many different ways and over as many different conditions as possible [91].

In a MEMS design, the coupling mechanisms between disciplines are the critical points to be modeled accurately since the transduced energy is the sensor mechanism or actuator control. Because the coupling mechanism is between the disciplines, the perception is that the models must be at a low abstraction level to achieve high precision. For examining a particular characteristic of the coupling mechanisms, low-level analyses are the only true mechanisms to ensure that the nonlinear or undefined effects missing in higher-level models do not accumulate to create significant errors. However, these detailed analyses can be time consuming, which can require hours of simulation for microseconds of real time. When longer simulation runs are required, the low-level analyses become restrictive at the system-level and the designer is forced to use higher level models for examining different conditions.

In many situations, the higher-level less-accurate models can provide all the information needed at the system-level. However, a point can be reached where the higher-level models become counter productive due to errors. One concern with higher-level models is that these models can break down under untested conditions and lead to the wrong response. Examples of extreme failure modes are structural breaks and contacts, where the functionality of the device undergoes significant changes. Besides the extreme situations, improperly modeled device behavior can cause system instability especially if the device's transfer characteristics are a strong function of the operating region. Errant behavior in a model is very difficult to predict without a different modeling viewpoint. Another reason to switch between different abstraction levels is to provide for verification and error analysis.

1.1.2 Importance of Modeling

Most equation-based simulators can expand their modeling capabilities by modeling a device in terms of differential equations [5,64-68], so only the mathematical behavior of a component is required to create a model. With the correct models defined for a component, any generic simulator can predict the physical behavior of MEMS from the mechanical structure to the electronics and beyond. Under these considerations, the accuracy of the models is *almost* independent of the simulator, where only superior models can create accurate simulations. If all modeling is assumed equal across all simulators, then the modeling issues shift toward the dedicated solver algorithms for improving performance.

To further complicate the problems in simulating MEMS, designers must deal with fabrication variations. With variations in the fabrication process, models cannot absolutely be defined for any component or material since a tolerance is on all parameters. Process variations do jeopardize the validity of requiring highly accurate models unless simulations are done over multiple sets of model variations. Even with the different sets of models, the designer still has accuracy constraints with uncertainties.

1.2 Independent Simulators

Most designers approach MEMS simulation using independent simulators as shown in Figure 1-3. In this architecture, the designers can continue to use their native simulator(s) and design environments [15,66,95]. The independence between the design environments and simulators allows the foreign elements to be modeled in any manner possible in a given simulator. In this procedure, the behavior of each device must be characterized, modeled, and verified against the measured or simulated results from the other disciplines [15,95]. Consequently, this architecture puts significant emphasis on the accurate modeling of components from a different discipline since the different databases can be completely independent. A potential hazard is that design modifications between the different environments are not necessarily updated in the models used in other databases. For this reason, the modeling procedure can require a sequential iterative process to reach an accurate and consistent model representation.

In this architecture, the problems are constructing models that are accurate over all

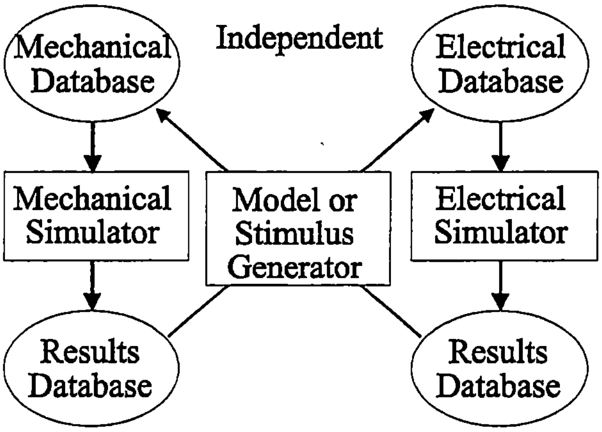


Figure 1-3. Independent Simulator Architecture

operating regions of each device and determining how to analyze effectively the devices using the system-level response when higher-accuracy models are required. Typically, the devices' behavior is modeled in a lumped-sum or macromodel format. These new or updated device models can be constructed in three different ways: automatically from the design database [8,63,85], automatically from simulation results [15,67,95], or analytically from the structure [64-66]. In the automated procedures, model generation does require translation of the results from the different simulators, so the different simulators do become loosely coupled via the data translator in the model generator. Although this iterative process can be time consuming and resource intensive, the final model can be efficient and accurate, at least over limited regions of operation.

1.3 Coupled Simulators

A single virtual simulator is implemented when different simulators are coupled as shown in Figure 1-4. With a simulation backplane, the different simulators can operate in a plug-and-simulate environment where distributed or parallel processing is a part of the process. A simulation backplane integrates simulators together through a procedural interface or inter-process protocol that links multiple simulators into a common environment and provides data-transfer and synchronization mechanisms. Designers from the different disciplines can simulate how their devices, components, or functions interact directly with other engineering disciplines using their native tools. Each designer has the benefit of examining the responses with different simulators. At the very least, the coupling of the different simulators provides the mechanism for concurrent verification of the

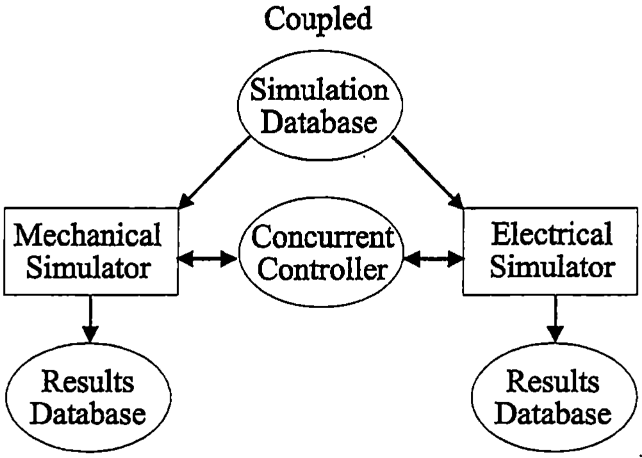


Figure 1-4. Coupled Simulator Architecture

system-level design from different modeling perspectives.

This coupled architecture brings challenges in terms of communications protocols, interfacing procedures, and design partitioning. Simulators do require an open interface with sufficient controllability as a prerequisite for this process. In particular, the design partitioning becomes a crucial part of the process. Virtually any configuration of simulators or design representations is possible with the proper partitioning algorithms. A designer can select how the designs or devices are partitioned across the different simulators or algorithms can search for optimal configuration of accuracy or performance based on certain simulation criteria.

The importance of the partitioning process is to address the performance problems of this architecture, especially the coupling of the Finite Element Analysis (FEA) with electrical analysis. The FE simulators offer very detailed time-consuming analyses [33], but repeated long simulations are certain to include redundant information from previous analyses. After several simulations, the coupling of the FEA with the electrical simulator becomes very monotonous and provides little or no additional insight into the behavior.

1.4 Unified Simulator

The unified simulator is a compact form of the coupled simulators with the same types of problems. The difference between the unified simulator and coupled architectures is that the simulation backplane becomes an algorithmic backplane, which internally links the different simulator algorithms into a single program with a common output as shown in Figure 1-5. For this implementation, a unified simulator would require the capabilities of ALL the individual discipline-specific simulators, i.e. the simulator would have unlimited capabilities: multiple levels of abstraction, multiple iteration methods, multiple analysis formats, and modeling information on all possible disciplines. At present, the mathematical-type simulators, like MATLAB [88], are the closest implementations to a unified simulator.

At a certain point, the difference between simulators becomes a translation problem,

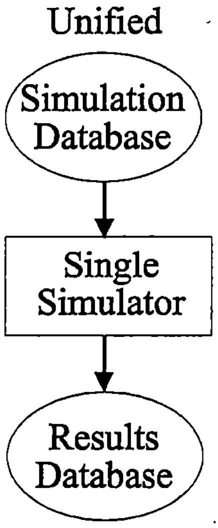


Figure 1-5. Unified Simulator Architecture

so a unified simulator can be emulated by integrating the modelers from the different disciplines. Discipline-specific modelers would translate all the various physical or conceptual representations into a common simulation or mathematical format. Once in a common database, a generic simulator could analyze the complete system of equations assuming that the size of the system does not exceed the simulator's capabilities. Because of the potential size of the problem, a distributed or parallel-processing mode becomes essential to the unified simulator architecture. This distributed homogeneous simulator does eliminate most interfacing problems that can occur between heterogeneous simulators, and load balancing becomes possible to optimize the parallel processing. The only missing element would be the dedicated solver algorithms and analysis methods for a particular problem.

1.5 Conclusions

The unified simulator architecture with parallel processing would be the optimum solution. However, the discipline modelers are already implemented in the different design and simulation environments, so the implementation choice is either the coupled or the independent architectures [67]. For linking different simulators into a larger virtual system, the coupled simulator architecture with the proper partitioning options is the most flexible approach to make any tradeoffs. However, the independent simulator architecture allows designers to continue using their existing design and simulation procedures at the cost of sacrificing *some* accuracy while maintaining a high degree of performance. The decision becomes one of performance and accuracy, where accuracy depends on how well the models characterize the behavior of the devices and materials regardless of the simulator. For this reason, only the performance aspects of the two architectures can be analyzed.

For a performance comparison, the independent architecture is required to validate or crosscheck the solution using the other discipline simulators to guarantee matching between the solutions. Otherwise, the independent architecture has a distinct advantage in performance, since the time consuming device-level analysis would be unnecessary. To calculate the simulation time for each architecture (X), four variables are defined: maximum system-level simulation time (T_S), maximum average mechanical simulation time (T_M), number of iterations (n_X^j), and number of data points (d_X^j). An average value for a variable y is defined by $\bar{y}$. Using these definitions, the total simulation times for the coupled architecture (T_C) and for the independent architecture (T_I) are

$$T_C = MAX(T_S,T_M)\cdot \sum_{j=1}^{j=d_C} n_C^j = MAX(T_S,T_M)\cdot d_C \cdot \bar{n}_C \qquad \text{Equation 1-1}$$

$$T_I = T_S \sum_{j=1}^{j=n_I} d_I^j + T_M \sum_{j=1}^{j=n_I} d_{I,r}^j \approx (T_S \cdot d_C + T_M \cdot \bar{d}_I)\cdot n_I \qquad \text{Equation 1-2}$$

In the coupled architecture, the number of iterations is varying per time point and the slowest simulator defines the simulation time because of parallel processing effects. For the independent architecture, the total simulation time is a summation of the system-level and the slowest device-level simulation time because of the sequential process. The ratio of the two simulation times is

$$\frac{T_I}{T_C} = \left(\frac{T_M \cdot \bar{d}_I + T_S \cdot d_C}{MAX(T_M,T_S)}\right)\cdot \frac{n_I}{\bar{n}_C \cdot d_C} \approx \alpha \frac{n_I \cdot \bar{d}_I}{\bar{n}_C \cdot d_C} \qquad \text{Equation 1-3}$$

For this relationship, the assumption is made that the device-level simulation time is typically slower than the system-level analysis, so the value can be lumped into the parallel processing factor (α).

Prior to the construction of accuracy models in the independent architecture, the only known relationship between the architectures is that $\overline{d}_I \leq d_C$. This condition occurs because the device-level simulation process can eliminate piecewise linear (PWL) conditions within the data sequence to reduce the number of points. By eliminating data points, the independent architecture can avoid the synchronization issues in the coupled architecture. With the potential disproportionate relationship between simulation times, this condition is assumed to eliminate the advantages due to parallel processing ($\alpha \cdot \overline{d}_I \approx d_C$), so the ratio depends primarily on the number of iterations.

For the number of iterations, the expectation is the initial value of n_I is equal to or greater than the maximum n_C^j, so $\overline{n}_C < n_I$. The justification of this consideration is that the initial model is a poor representation, which can cause divergence until the regions of device operation are parameterized. However, the number of iterations in the independent architecture will approach unity if *the accuracy specifications are met over all simulation conditions*. Unfortunately, the number of iterations in the coupled approach remains fixed because each simulation starts with no prior information. A point is reached where the independent architecture using behavioral models becomes more efficient than the coupled architecture.

1.5.1 Dissertation Goals

Based on the comparison of the two architectures, the coupled approach is more efficient than the iterative process of the independent approach ***until behavioral models reach an acceptable accuracy***. Even with a model that has acceptable accuracy, situations can still occur when models break down because of conditions outside of the characterization range. During these conditions, the only option is to switch to different abstraction levels to guarantee the operation and accuracy of the device or module. However, the cost of this modeling switch is a decrease in simulation performance, which is unfortunately the main benchmark for simulation. In general, these coupling processes typically have poor performance compared to behavioral models. Although the solutions from the coupling process can have superior accuracy to behavioral models, optimized behavioral models over a narrow range of operation can have precision equal to FEA.

To provide the maximum performance and to always meet the tolerance parameters, a hybrid approach is needed where the advantages of both architectures are used. This hybrid approach involves a dynamically configurable simulation [72] where different modeling representations are switched during the analysis to promote performance or accuracy as shown in Figure 1-6. This illustration also shows how two different model representations can have equivalent accuracy with different performance parameters. Additional information about the dynamic modeling procedures is provided in section 2.5 and section 4.4.

For maximum flexibility, simulation is done through the simulation backplane, so different simulators can be exchanged during the analysis. **The goal of this research is to**

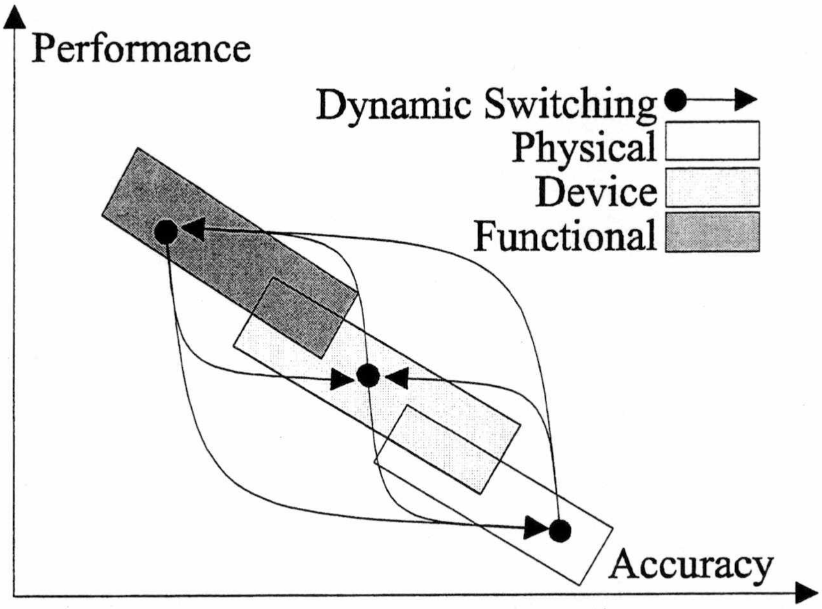


Figure 1-6. Dynamic model switching based on simulation performance and accuracy

create a coupled architecture using a simulation backplane with dynamic reconfiguration of the discipline-specific simulators for MEMS analysis. Specially, a mechanical FE simulator and an electrical simulator are coupled, and representations for the MEMS device model are dynamically switched between FE and behavioral models. For the dynamic switching process, the assumption is that the alternate behavioral models exist prior to the analysis although the coupling process could guide the domain mapping between the disciplines as suggested in section 2.5. For the backplane development, the major questions to be answered are:

- What are the general procedures for the simulation backplane?
- What specific operation(s) does the simulation backplane require to switch dynamically between models representations in different simulators?
- What calculation and iteration process is needed to couple the simulators and to achieve

convergence?

- What type of interface(s) is needed to couple the simulators?
- Does an optimum interface exist in terms of convergence and iteration efficiency to couple the simulators for the hybrid approach?
- Can the backplane implement an interfacing procedure to improve the coupling efficiency between the simulators?

1.5.2 Dissertation Outline

This research is presented in the following manner. Chapter 2 presents a literature review that describes the present MEMS specific design and simulation environments. The different coupling methods for different simulators and backplane implementations are examined. The basic requirements of a simulator are outlined for this coupling architecture and a more comprehensive overview of the dynamic modeling option is provided. An accuracy-based trigger is also examined with other considerations to aid in the behavioral mapping of a device using the dynamic modeling procedure.

Chapter 3 examines different interfacing structures with the emphasis on the configurations and modeling combinations to optimize the iteration efficiency and convergence between simulators. This section also outlines the process flow of calculation procedure from direct nodal relationship to the final error-minimization procedures with input causality identification using bond graph techniques. Specifically, various methods of using causality assignment are examined, and a flexible interface node is created to investigate the effects of dynamic interface configuration on the iteration process. A flow correction method, which overlaps FEA with a behavioral model in the electrical simulator, is also examined for improving performance between simulators.

Chapter 4 describes the implementation process for backplane process. The procedures for the different backplane modes are outlined and considerations for improving performance are described. The database structure of the backplane and local simulator interfaces is described. Additional specifications on the iteration process are defined.

Chapter 5 provides the results from the different coupling configurations and examines the relative merits of the different coupled approaches over a variety of modeling representations and design examples. The results show mechanical-to-electrical simulator coupling and electrical-to-electrical simulator coupling to demonstrate the flexibility of the backplane. The results for different coupling processes using the hybrid method are presented with examples using no coupling, static coupling, and dynamic coupling on the performance and relative accuracy of the analysis.

Chapter 6 presents the conclusions and future areas of research.

Chapter 2. Background

This literature review is presented in almost chronological order as a system-level view is taken for simulating MEMS and for crystallizing the scope of this topic. The review began by examining the different MEMS-specific environments for simulating MEMS, where FEA is typically used to examine the characteristics of mechanical devices. Based on these environments, coupling the different simulators is the most direct method of analyzing the system although creating behavioral models is the more common and simple method of analysis. This information is presented in section 2.1.

For the coupling process, partitioning the system-level design, calculating the solution across multiple simulators, interfacing and communicating between simulators, are the major issues. The partitioning process is the first and perhaps most important step in coupling the different simulators to meet a designer's perceptions and goals of modeling all aspects of a system from a specific simulator to a certain abstraction level. To use existing CAD environments for these partitioning tasks, a system-level partitioning process is required to create discipline specific databases to define this array of options. This partitioning process is described in section 2.2 with the ramifications on the simulation backplane.

After the system-level design is partitioned and post-processed, the analysis task begins and the different discipline simulators are coupled to find a solution. The focus of most research in this area is defining the interface and iteration method to optimize performance. The basic simulator iteration and interface process is examined in section 2.3. Finally, each simulator requires certain functions and capabilities to be used in the coupling process. The basic simulator requirements and an overview of the backplane control are described in section 2.4. Finally, the dynamic modeling process is presented in section 2.5.

2.1 MEMS Simulation Environments

The main design and simulation environments specifically for MEMS are CAEMEMS [47-49], MEMCAD [51-59], and SESES [60-62]. These different MEMS design-and-simulation environments have focused primarily on the mechanical aspects of MEMS using FEA and those electrical properties influencing the mechanical behavior. With FEA, designers can examine the behavior of practically any structure given the material and interaction characteristics of the components. However, these systems concentrate primarily on device-level and not system-level analysis.

To overcome this problem, Application Specific Integrated Systems (ASIS) have been proposed for MEMS [2], which like Application Specific Integrated Circuits (ASICs) would create separate tools to integrate the different disciplines. At this point, the typical approaches are coupling simulators and creating parameterized behavioral models. Several authors have already coupled an electrical simulator to a mechanical simulator for MEMS, and the results demonstrate the high accuracy and slow simulation performance relative to behavioral models. This section examines the different MEMS-specific simulation environments, different simulator coupling approaches, and MEMS behavioral modeling.

2.1.1 CAEMEMS

Computer-Aided Engineering of Micro-Electro-Mechanical Systems (CAEMEMS) is a CAE framework for MEMS, which functions as a MEMS-specific designer's workbench [47]. This MEMS framework provides a set of databases, libraries, graphical user interfaces (GUI), plotting functions, solid modelers, and utilities for the integration of MEMS-related CAD and CAE programs [48,49]. Within this framework, a user can design and optimize specific MEMS applications, and the tools provide guidance to the user. One module is the CAEMEMS-D [48] for pressure sensor design and simulation. Another module is ASEP [49] for simulation of the etching of <100>-oriented silicon using KOH. ANSYS is the finite element simulator used in CAEMEMS.

2.1.2 MEMCAD

MEMCAD system is an environment that creates the architecture to address the system level problems and the specific tool-development level, that can solve the entire coupled set of CAD problems [50]. MEMCAD consists of accurate 3-D solid model generation from CIF mask layers and process descriptions and development of material model databases [51,52,53,58]. In MEMCAD, the MEMS design process requires designers to create a 2-dimensional layout in CIF mask format. Then, the CIF file is translated to IGES format, and a solid modeler generates the 3-D shapes using process descriptions as input into a mechanical simulator [52,58]. The 3-D solid model is simulated using ABAQUS for the finite element analysis and visualization using PATRAN or I-DEAS.

With a solid representation of the mechanical structures, MEMCAD can find the deformation of the mechanical structure due to electrostatics [55,56,59]. The electrostatic simulator called COSOLVE-EM [59] is used to find charge densities across structures using ABAQUS and FASTCAP. The charge distribution creates electrical pressure loads for the mechanical simulator to determine the deformations, hysteresis effects, or contact problems for a structure [57]. As the mechanical structure is deformed, the charge density across a structure is redistributed to change the electrostatic force contributions. An iterative process is applied between the deformations and charge densities until the device reaches self-consistent values.

2.1.3 SESES/SOLIDIS

Unlike the CAEMEMS and MEMCAD environments, SESES is a closed system and not a collection of interfaced programs [60]. The SESES environment has concentrated on developing a unified software concept, which provides flexible coupling of electrical, thermal, and mechanical deformation phenomena in a uniform and consistent environment [60]. In addition, the general coupling effects between physical properties of MEMS are outlined to illustrate how the different coupling effects can influence the iteration methods for solution of the system equations. SOLIDIS is a package of programs with common database structures and algorithms that uses tailored finite-element analysis to solve self-consistent thermo-electro-mechanical problems [61,62]. This system can create a SPICE netlist representing the behavior of the mechanical device for simulation at the system-level [63].

2.1.4 Concurrent Electrical and Mechanical Analysis

Several articles describing concurrent analysis for MEMS are [32-36]. In [32], an electrical simulator ELDO is coupled to the mechanical simulator ANSYS for a piezoelement simulation. An iterative solution is found with ANSYS controlling the process. Different interface configurations based on relaxation methods are examined. In [36], the electrical simulator PSPICE is coupled to ANSYS for piezoelectric analysis of an oscillator. An external transfer program creates load vectors for ANSYS from the PSPICE analysis. In [35], an electrical VHDL simulator is linked to the mechanical simulator PROUESSE using Remote Procedure Calls (RPC). Because of the simulator types, this particular problem uses logical states instead of current and voltage.

The remaining articles define a simulation backplane to link the simulators. In [33], ANSYS and the electrical simulator SABER are integrated using Parallel Virtual Machine (PVM) and gradient methods to solve the interface problem. However, an iterative process is not used to generate a consistent result like [32]. The test cases are an acceleration sensor and thermal analysis of an electrical circuit. The lack of performance is defined as a major negative of the coupling process although the backplane significantly improves the performance compared to file-based transfers. In [34], the mechanical simulators CAPA and ANSYS are coupled to the electrical simulator KOSIM using a relaxation-based method with Newton's method. PVM is also used as the communication mechanism.

2.1.5 Behavioral Models

For system issues, behavioral models in an electrical simulator provide a designer with a faster and simple approach to designing and simulating MEMS [5,64-67,85]. Electrical devices have also been simulated in a FE simulator [68]. In most situations, the designers develop the mathematical behavior of a device based on analytical simplifications and assumptions [5,64-65,85], where the behavior can be verified using FEA. The most practical approach is to apply stimuli to the FEA and then create behavioral models for the electrical simulators [67].

Reduced-order models for the Finite Element Method (FEM) [14] and lumped model equivalents [8,9,85] of the fabricated device responses can produce the same accuracy as the FEA at least in certain operational regions. The concern in this work is that device models become invalid under certain conditions since higher-level models are by definition less accurate than lower-level physical representations. In most cases, the problem is not the behavioral model per se, but errors resulting from system-level analysis using faulty stimuli definitions outside the model limits. This situation becomes possible as different discipline engineers become involved in a system when the limits of the design can be reached or exceeded. Minor inaccuracies in the behavioral model do become irrelevant beyond a certain point once the designer has tested the major conditions, but failure modes are still possible in the unverified situations.

2.2 System-level Partitioning

Most partitioning processes break a system into small modules starting with a system-level representation or top-level design as shown in Figure 2-1(a). For each design, the partitioning process can potentially choose from a variety of behavioral models or structural representations for each instance of a design as shown in Figure 2-1(b). Virtually any configuration of simulators and modeling options is possible with a flexible-partitioning program [17,21] and the proper design and model viewpoints [69]. The degree of partitioning across multiple simulation databases depends on the capabilities of each

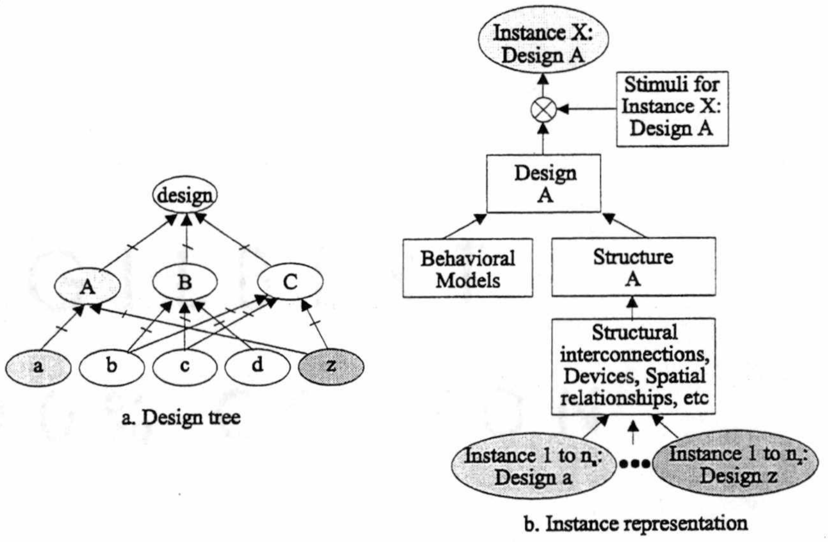

design
A
B
C
a
b
c
d
z
a. Design tree
Instance X: Design A
Stimuli for Instance X: Design A
Design A
Behavioral Models
Structure A
Structural interconnections, Devices, Spatial relationships, etc
Instance 1 to n_a: Design a
Instance 1 to n_z: Design z
b. Instance representation

Figure 2-1. A complete design database

simulator and the partitioning options.

The first task for all partitioning algorithms is to flatten the design hierarchy by expanding all instances of a design until the desired modeling representation or the primitive elements are reached. Starting with the top level of the design, flattening algorithms take the present instance names at each hierarchical level and use them to name all sub-instances, nodes, devices, etc. From the flattened database, the partitioning process generates a unique simulation database for each simulator in the analysis. The simulation databases contain the interface configurations between the simulators, the communication mechanisms, and the object naming conventions to link simulators across the backplane.

Most design environments and CAD frameworks provide partitioning and translation mechanisms to create a simulation database for a specific simulator(s) from a design database(s) [70,71]. In this research, the most notable partitioning problem is the lack of a common design database prior to generation of a physical database for the fabrication process. This work assumes that multiple design environments are used, so no single database will contain **all** design and model representations. Without a common or central database, an automated or interactive partitioning process is not possible with all the different modeling options simultaneously. Different simulators can still be coupled using a common naming reference through the backplane, but only single instances of a different discipline design can be simulated without manual creation of a design database for each instance. One environment is expected to contain sufficient information to construct a system-level database. With a system-level database, the major problems in the partitioning process are the discipline refinement process and an instance's discipline-specific stimulus.

The discipline refinement process generates the simulator configurations and performs all translation functions between the different databases. Examples of the refinement process are the meshing operations for the FEA and separation of analog and digital circuitry for mixed-mode simulation. Two implementations for the refinement process are a single partitioning program for all disciplines and different post-partitioning processes from within the different design environments. In both cases, a system-level partitioning process is still required to recognize and to separate the different disciplines from a central database. However, the single partitioning program requires enormous flexibility and scope to implement any known partitioning algorithm. The multiple discipline partitioning option follows the basic premise of the research that the discipline-specific tasks are best performed by the native tools. The partitioning process also defines the environments doing the post-processing on the different design or simulation databases [70]. Consequently, each instance can be post-processed differently based on performance or accuracy constraints. The only requirement in the discipline-specific processes is that all post-processing operations must preserve the links to the simulation backplane.

For just coupling the different simulators, access to the stimuli in a discipline is not a requirement. However, the hybrid approach requires the stimuli to be synchronized between the different simulators, or the solutions are not correlated as all distributed variables are lumped into a single composite representation to prevent complex interactions. The stimuli are also required if a behavioral model is generated using data from another simulator.

2.3 Interfacing and Iterating

For defining the interfacing and iteration process, this research examined mixed-mode simulation in electrical simulators [16-22,41-45,47], mixed device and circuit simulators [28-31], mixed level simulators [44,46], parallel processing [23-26,80-84], and simulator algorithms [10-13,37,40]. In mixed-mode simulation, analog and digital types of electrical analyses are combined. Analog simulation uses the fundamental equations (Kirchoff's Laws) of circuit analysis to solve voltage and current relationships between devices. Digital simulation abstracts the circuit analysis of the voltage or current levels into discrete logical states. Mixed-mode electrical simulation is a translation process between two different abstraction domains, which did not directly apply to this research. However, the architectures of the mixed-mode simulators and synchronization of the different simulators are relevant as described in section 2.3.1 and section 2.3.3.

Mixed circuit-device simulation implements the same basic process that is required in this research. Most iteration schemes between simulators with nonlinear relationships use some variation of the Newton Raphson method to linearize the given variables at the present solution [31,46]. The focus of most coupling schemes is the interface between the solver algorithms to maximize performance [29,30]. In [29], the full, two-level, and modified two-level Newton methods are compared. The full Newton method solves all unknown variables simultaneously as a single problem. The two-level method does a standard linearization at the device and circuit level, where the device is linearized at one level and the circuits are linearized at a second level using the device representation. The modified two-level method applies a linear predictor step at the device-level in the standard two-level method to improve the runtime performance and to maintain the same iteration efficiency. A Norton equivalent circuit is used at the circuit level for the device, and voltage sources are applied at the device-level. In [30], different node tearing algorithms using a full Newton method are examined by adding different resistance boundary conditions to the interface. The addition of series resistance to the bordered block diagonal matrix did improve the runtime performance and iteration efficiency of the coupling process in three dimensions compared to an interface without the resistance. The mixed-level simulators [44,46] have similar characteristics to the mixed device-circuit simulator, except that state variable transfer functions in S and Z domains are solved with the circuit analysis.

The basic simulation methods are typically variations and combinations of different iteration techniques to improve performance without sacrificing accuracy. To enhance performance, the common implementation practices are bypassing unnecessary model evaluations [10,22,74] and partitioning the matrix into loosely coupled blocks [10,11]. Model evaluations are done during solution checks and the sensitivity calculations within the blocks. With the proper partitioning of the system into blocks, simulators can suspend internal iteration processing once all shared and internal variables have converged for a block. The coupling process becomes event-driven or selectively traced [10,11], so blocks remain idle whenever possible.

In the iteration process for this research, the system-level partitioning process has already divided the complete system into disciplines. However, the blocks based on discipline partitioning are not necessarily efficient from a simulation standpoint. Iteration methods can require certain matrix conditions to guarantee convergence, so matrix re-assembling or matrix overlapping [27] may be required to provide the necessary conditions for convergence. Another technique to improve performance is to limit the number of iterations without guaranteeing full convergence (e.g. timing simulation). The solution process is described in section 2.3.2 and the convergence requirements for an interface are outlined in section 2.3.4.

Parallel processing has the goal of fully utilizing all computer resources by load balancing and eliminating sequential steps from a simulation process [23-26,80-84]. Although the iteration process for coupling simulators has qualities of parallelism, load balancing cannot be achieved intentionally. Any improvement in performance due to parallel processing is just a by-product of the discipline partitioning process. Furthermore, mechanical FEA is computationally intensive compared to electrical analysis, so the process is typically dominated by the mechanical simulation time unless the mechanical simulator has a parallel processing option.

2.3.1 Simulator Architectures

The mixed-mode simulators provided the most information about the simulator architectures, which are very distinctly defined because of the mapping process between continuous and discrete states. The three architectures for coupling mixed-mode simulators are the core, glue, and unified approaches [16,21,40]. These architectures are the bases for the independent, coupled, and unified architectures in Chapter 1. Figure 2-2 illustrates these different simulation architectures for mixed-mode simulation.

In the core architecture, the analog or digital component is given capabilities in another simulation domain [16,21,40]. One single simulator does all information conversion

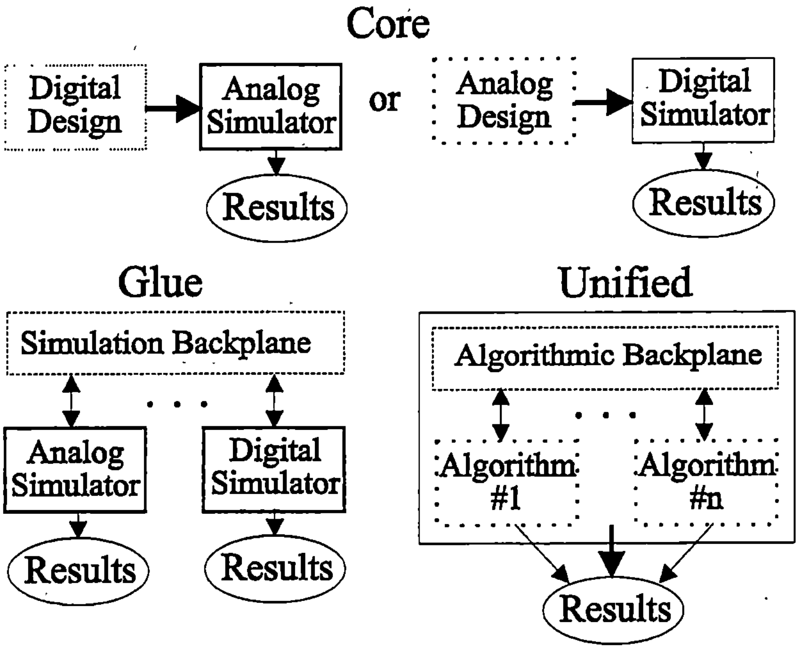


Figure 2-2. Three different mixed-mode architectures

between the two domains by simulating a component at the level of the given simulator. In the analog simulator, digital cells are modeled with analog representations while analog cell's responses are emulated with digital representations in the digital simulator.

Unified simulators make no distinctions between analog and digital circuits due to an internal algorithmic backplane, which provides policies that govern time synchronization, signal mapping, etc. [17-22,40,44]. A single, integrated environment has algorithms to solve any specific level of abstraction. This type of simulator is very flexible, and new algorithms can be compiled into the backplane as needed. Most simulator development is moving towards algorithmic backplane architectures because of their flexibility and expandability. This flexibility allows the unified simulators to be used as glue simulators, so a unified simulator should be the first type of simulator considered in any simulation process.

The glue approach has separate simulators communicating and working together using a procedural interface or inter-process protocol [41-43,45]. This protocol links multiple simulators into a common environment and provides data-transfer and synchronization mechanisms, which collectively are called a simulation backplane. A simulation backplane is a distributed version of the algorithmic backplane. Potentially, any design can be solved regardless of the size or complexity since new simulators with compatible interface protocols can be added to the backplane as needed. Multiple simulators can be coupled using a simulation backplane for distributed and parallel processing with the proper partitioning process.

2.3.2 Solution Process

All solution processes are required to solve a system of M equations where functions of a vector ($x \in \Re^M$) at a given time are equal to zero, or

$$f_j\left(x,t\right)=0 \quad \textit{for all } j \in M \qquad \text{Equation 2-1}$$

Because of the pre-partitioning operations, these equations are spread across multiple simulators, but the basic solution process does not change. System-level iteration still performs standard nonlinear solver procedures, except model evaluations become solution sequences of a simulator. The two solver techniques considered in this research for the coupling process are the Newton-Raphson and relaxation methods. The Newton method is most commonly used in the interfacing process because of tightly coupled relationships between simulators.

2.3.2.1 Newton-Raphson Method

For the simulation of the nonlinear system of equations, the Newton-Raphson (Newton) method or linearization process is typically applied where

$$x^{i+1}=x^i-\left[\frac{\partial f^i}{\partial x^i}\right]^{-1} f^i \qquad \text{Equation 2-2}$$

$$\left[\frac{\partial f}{\partial x}\right]=\begin{bmatrix}\frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N}\\ \vdots & \ddots & \vdots\\ \frac{\partial f_N}{\partial x_1} & \cdots & \frac{\partial f_N}{\partial x_N}\end{bmatrix} \qquad \text{Equation 2-3}$$

The sensitivity matrix $\left[\frac{\partial f}{\partial x}\right]$ is also called gradient matrix or Jacobi matrix. For an unknown N terminal device, this procedure requires the simulator to perform at least N model evaluations to generate the N^2 gradients depending upon the differential formula. The gradient value is calculated as

$$\frac{\partial f_k}{\partial x_j} = \frac{f_k(x^i + \Delta x_j^i) - f_k(x^i)}{\Delta x_j} \textit{ for all } k \in N \qquad \text{Equation 2-4}$$

This linearization procedure applies a variable delta to a device's terminal with all other terminals held constant. Then, the functional offsets of the other terminals are measured and used to define the sensitivity matrix. Other differential formulas are also possibilities (see [94, pp.149-155]), but additional evaluation sequences are required if a more complex formula is used. All variables with no direct relationship to a device have a sensitivity of zero. Most systems of equations are sparse with blocks of tightly coupled relationships.

Because of the computational intensity of this process, the full gradient calculations are done very selectively. In many cases, a modified Newton method is used where

$$x^{i+1} = x^i - \left[\frac{\partial f^0}{\partial x^0}\right]^{-1} f^i \qquad \text{Equation 2-5}$$

The full gradient calculation is performed only once, since one calculation can provide a reasonable approximation [10,78] that is mathematically accurate enough to be iteration efficient. The tradeoff is between model evaluations for the sensitivity calculations or for finding new solutions. For example, N+1 model simulation sequences can test N+1 new solutions or test a single solution with the calculation of a new sensitivity matrix. The choice is to do model evaluations for new solutions except during initialization. At initialization, the simulators do not have a valid sensitivity matrix definition because of the unknown solution vector. Several iterations can be required before the solutions reach sufficiently close conditions [78,79] to calculate a valid sensitivity matrix.

The Newton method is the main iteration technique in the mixed device-circuit simulators because of the tight coupling between simulators. As outlined in [31], the Newton matrix for the mixed device-circuit simulator is defined by

$$\begin{bmatrix} R_0 & 0 & \cdots & 0 & T_0 \\ 0 & R_1 & \cdots & 0 & T_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & R_m & T_m \\ S_0 & S_1 & \cdots & S_m & Y \end{bmatrix} \begin{bmatrix} \delta x_1 \\ \delta x_2 \\ \vdots \\ \delta x_m \\ \delta z \end{bmatrix} = \begin{bmatrix} -F_1 \\ -F_2 \\ \vdots \\ -F_m \\ -F_n \end{bmatrix} \quad \begin{matrix} R_m = \dfrac{\partial F_m}{\partial x_m} & S_m = \dfrac{\partial F_n}{\partial x_m} \\ T_m = \dfrac{\partial F_m}{\partial z} & Y = \dfrac{\partial F_n}{\partial z} \end{matrix} \qquad \text{Equation 2-6}$$

The variable x represents the internal device variables and terminal currents, the variable z represents the voltage and currents between modules, and the variable F represents the time discretization of the function that should be zero [31]. Within this research, access to the internal variables of the device or module is possible, but only the shared power data (z) for a terminal is exchanged between simulators with the relevant sensitivity information for the variables. The internal variables are eliminated from the system-level representation. To maximize performance, the individual sensitivity matrixes are calculated in parallel by each simulator with respect to the shared nodes. These sensitivity matrixes are broadcast or transferred [28] to all simulators for defining the system-level interface matrix. This calculation process is outlined in section 3.4 for this simulation backplane.

2.3.2.2 Relaxation Methods

A general overview of different relaxation methods can be found in [10,11,78]. Most relaxation techniques use some variation of Newton's method as a linearization step to solve the nonlinear equations. The relaxation-based algorithms are well suited for parallel processing [86], since solutions can be done per variable and a complete sensitivity matrix does not have to be calculated [10]. However, relaxation methods are guaranteed to converge only if the system-level matrix has a diagonal dominance. To define diagonal dominance, the system-level in a linear format is defined by

$$Ax = b \quad A \in \Re^{MxM} \quad x, b \in \Re^{M} \qquad \text{Equation 2-7}$$

For diagonal dominance, the absolute value of the diagonal elements of the system matrix are larger than the summation of the absolute value of non-diagonal elements, or

$$\left|A_{j,j}\right| \geq \sum^{M} \left|A_{j,k}\right| \; k \neq j \qquad \text{Equation 2-8}$$

For the system-level partitioning in this research, diagonal dominance is a condition that cannot be assumed.

As an example, two configurations for a sensor application are shown in Figure 2-3. The worst case situation is Figure 2-3(a), where the external device is defining the feedback path for the amplifier. Any variations on V_A around V_{MID} can make V_{OUT} oscillate between the output limits of the power supplies. With large gain, a solution may never be found especially if the error tolerances for the variables are too large. In this situation, the simulator with the amplifier requires the completion of the feedback path. Figure 2-3(b) is a more stable example, since the feedback sets the virtual node at V_A to be V_{MID}. However, the wrong interface model (a voltage source) at the virtual node can still cause divergence problems. These situations show that poor partitioning of the system-level across multiple simulators and poor interface definitions can cause problems

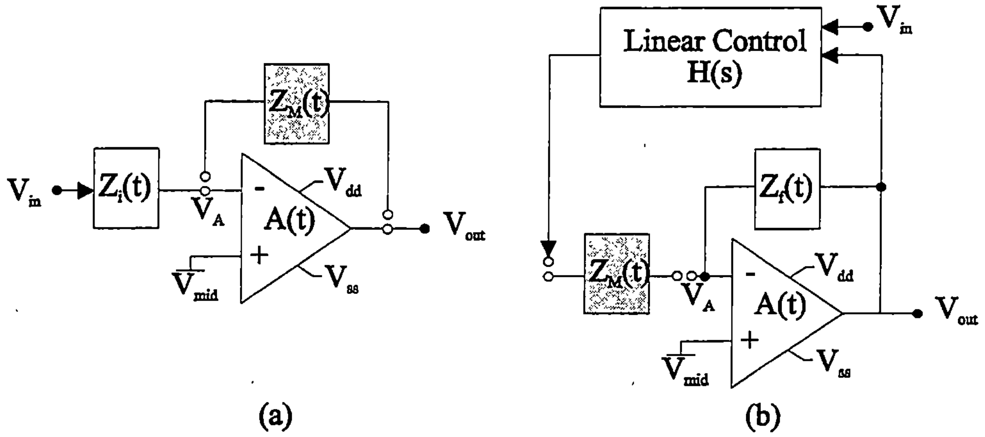


Figure 2-3. Two extreme feedback problems

in a relaxation process. One method to overcome the poor partitioning problem is the use of overlapping matrixes [27], which is technique used in waveform relaxation [12].

2.3.3 Synchronization

The two most common synchronization methods for simulation are the lockstep and rollback methods [40-42,73-75]. The rollback scheme takes an optimistic approach to time step control, where all simulators progress at their individual rates. When a variable conflict occurs between simulators, each involved simulator rollbacks all timing information to the conflicting time point, and then re-starts simulation using the new information. Because of this characteristic, the rollback scheme is a memory intense process since sufficient information must be stored to return to the time point.

The lockstep method is a pessimistic approach that sets the global time-step as the minimum timestep of any simulator. By sacrificing simulation performance, this method is very efficient in terms of memory utilization, since only the present time delta of the simulator is changed. Due to the complexity of the different simulators involved, the lockstep method is more appropriate for this research. As a requirement, each simulator must have a one-step rollback where the backplane controls the simulator's present timestep.

A variety of hybrid methods is also possible as outlined in [73-75], but these methods can cause locking problems. In simulators that partition a matrix into smaller blocks, a typical synchronization method is to allow the different blocks to proceed at different rates. The potential problem with these methods is that different time deltas can be too large and the same data points are iterated to calculate multiple intermediate solutions. The iteration process can invalidate previous data points [20], but these hybrid synchronization methods can improve the simulation performance of the analysis if piecewise linear (PWL) conditions exist.

2.3.4 Convergence

The three main convergence parameters are a deadlock limit, an absolute tolerance, and a relative tolerance. For variable convergence, the convergence criterion is based on an absolute tolerance parameter (δ_A) and a relative tolerance parameter (δ_R). An absolute tolerance requires that the absolute difference between the present solution (X_P) and the next or expected solution (X_N) values must be less than absolute parameter, or

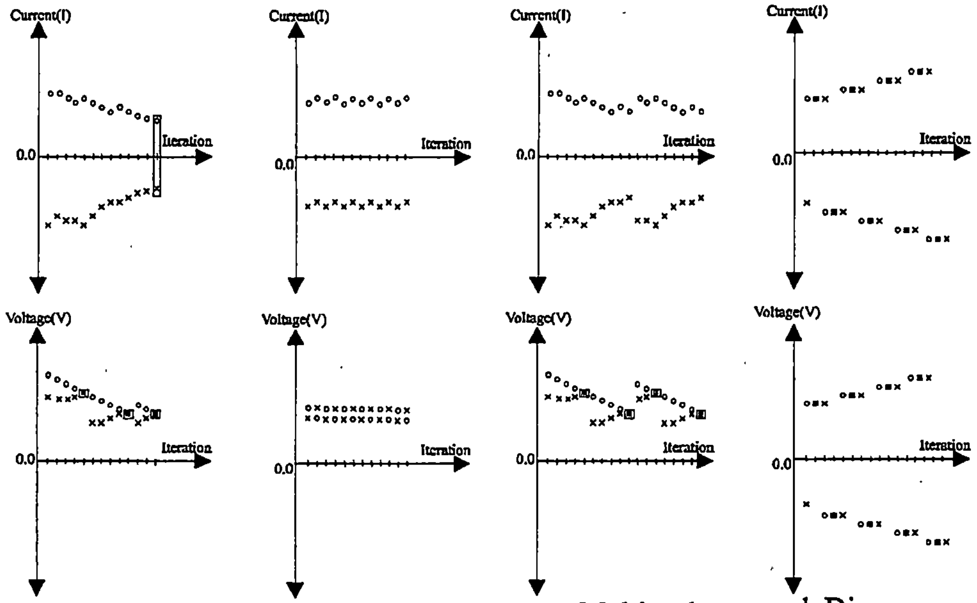
$$|X_N - X_P| \le \delta_A \qquad \text{Equation 2-9}$$

A relative tolerance requires that the absolute difference between the values is less than a parameter times the maximum variable plus an absolute tolerance [29], or

$$|X_N - X_P| \le \delta_R \cdot MAX(X_N, X_P) + \delta_{AR} \qquad \text{Equation 2-10}$$

The absolute relative tolerance ($\delta_{AR} \ll \delta_A$) is used during the relative comparison to ensure that a comparison is not operating on integration noise or a signal too close to zero. The relative tolerance procedures are the most flexible.

In a calculation process, a solution can iterate towards convergence, non-convergence, or divergence as shown in Figure 2-4. Divergence is the solution sequence with an oscillating geometric growth while the non-convergence applies to a solution that never reaches a stable point nor diverges. Most non-convergence problems are symptomatic of feedback loops, model discontinuities, tolerance problems, poor sensitivity calculations, or poor calculation methods. Instead of detecting non-convergence, most simulators implement a deadlock limit. A deadlock or iteration limit specifies the maximum number of solution iterations (n_L) that can occur on a single data point before the simulator is halted.



a. Convergence. b. EVEN/ODD type non-convergence. c. Multicycle type non-convergence. d. Divergence.

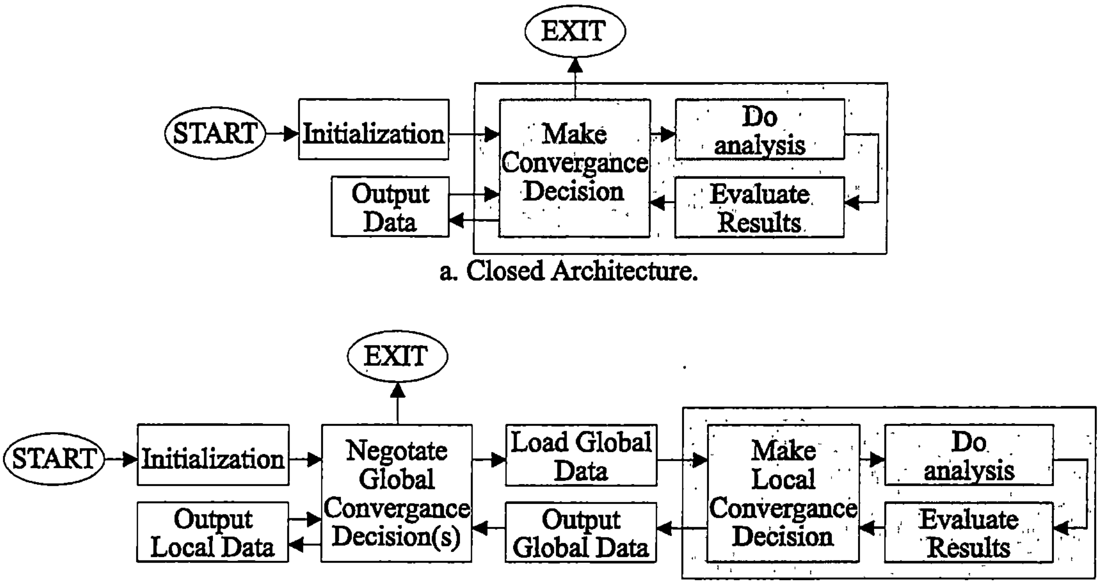
Figure 2-4. Comparison of convergence and non-convergence types

All simulators have a deadlock limit to prevent the endless iteration for a single solution.

2.4 Communications and Controls

The communication mechanisms are secondary issues compared to the simulator requirements, internal database structures, and control mechanisms, since the communications mechanisms are an implementation issue. In most cases, performance degradations due to the backplane operation are assumed insignificant compared to the iteration time of a simulator. With a FE simulator in the analysis, this performance condition can easily be verified, but the coupling process is prohibitive for "small" problems. The goal for a backplane is flexibility and expandability to allow data exchange between any simulators through common procedures [39]. Using a simulation backplane, the simulators become computation engines controlled by the backplane interface. The main operations of the simulation interface are accessing internal variables, modifying internal matrix values, translating information between the backplane and the simulator, and executing simulator operations consistently with the other simulators.

All simulators in this coupled architecture require an iteration capability that can be controlled by an external program or internal interface code. Simulators with an open-architecture can make the implementation simpler than simulators that need modification at a structural level to incorporate the backplane interface. The basic difference between an open and closed architecture is shown in Figure 2-5, where the shaded areas are common between the architectures. With an open interface, the translation and command operations can be incorporated directly into the simulator. In the worst case situation, the iteration



a. Closed Architecture.

b. Open Architecture.

Figure 2-5. Closed and distributed architectures for a simulator

process can be implemented by a middle-ware program as a sequence that starts and stops a simulator by saving and loading previous solutions with new data parameters. Most control functions depend upon the type of system-level architecture in terms of a central controller or a distributed controller [38]. A central controller defines what each simulator does while distributed controllers have each simulator perform the same procedures. This process is outlined in Chapter 4 for this simulation backplane.

2.5 Dynamic Modeling Process

The dynamic modeling process is described in [72], and a brief summary of this work for mixed-mode simulation is presented here for the simulator coupling process. In dynamic simulation, the four major parts are the representation control, model definitions, state control, and timing control. Representation control defines the situations and conditions where the different models and the interface configurations are switched (connected or disconnected) into the analysis based on trigger conditions (automatic mode), static selection (constant mode), or interactive selection. The static control mode does the standard simulator coupling process and the interactive mode waits for the designer's request to toggle between models. In the automatic mode, a trigger can be any event or condition in the analysis. The representation control using simulator coupling is examined in section 2.5.2 where a region-based modeling approach using an accuracy trigger is proposed.

The various model representations are typically part of the simulation database prior to the analysis as in [72], but the model generation is possible during the analysis. Models only need to be valid prior to the switching process. Considerations for different model definitions are examined in section 2.5.3 in terms of the region-based modeling. The timing control involves the rollback of the simulators and the recovery operation from a simulation failure while state control guarantees that both models have equivalent states at the transition point. In this work, these two control mechanisms are grouped together where timing control is used to implement the state control as described in section 2.5.1. The general sequence for the dynamic modeling is the following procedure after the detection of a switching condition:

1.Stop the analysis after the present iteration.

2.Save the state information of the present model(s) and shutdown or turnoff the simulator(s) with the present model.

3.Start the new simulator(s) with the new modeling definition(s) at N steps prior to present time. Apply the previous N solutions and stimuli directly to the new models and load any state information for the old model if relevant.

4.Link the new simulator(s) into the backplane using the appropriate name, variable, and objects references.

5.Restart the analysis.

Unlike [72], this procedure does not rollback the analysis of the simulators since only a single step rollback is permitted. Consequently, the models must be reliable and relatively consistent until the transition point occurs.

2.5.1 State and Timing Control

Without a doubt, state control is the most critical consideration for an accurate transition between the models because the internal state information of the old model does not necessarily correlate to a new model's states. In the simplest case, all models can be completely defined by the present terminal variables and the applied stimuli in a lumped-sum format. However, devices like an oscillator or devices with hysteresis effects can be multi-valued, so finding the equivalent state can be difficult based only on the present solution of the device. To ensure state control, all models can share certain internal variables to prevent state conflicts as part of the modeling method. Adding required states to the procedure will limit the number of modeling methods that can be used [90].

In this process, the timing control becomes responsible for the state control by seeking the closest equivalent state from the available information. The state equivalency and model compatibility are not rigorously checked to permit switches between any model representations, so the model transitions can introduce errors into the analysis. With a sufficient number of time steps in Step 3, the timing control is assumed to synchronize the dynamic response between the new and old models. The problem becomes determining the number of steps to reach synchronization or providing mechanisms to ensure state control.

2.5.2 Representation Control

Like [72], the trigger can be any event, and for this research, a certain accuracy of the behavioral model is considered as a trigger. The model representation changes only when the simplified behavioral model matches the response from the more accurate model within a specified error tolerance. Additional constraints on this condition are that the trigger conditions must persist for a certain sequential number of iterations and analysis times. Of course, the simulation time of this other model in the other simulator has to be less than the simulation time of the other simulators to improve performance. Otherwise, there is no reason to change representations.

To recognize the trigger, the behavioral model simulators can be in concurrent operation with the most accurate model, but the information is not included in the iteration process. A central controller is required to make the dynamic modeling decisions. Once the behavioral model simulator has met the given tolerance parameters, the FEA is shutdown and the behavioral simulator assumes a normal status to replace the FEA. These various conditions guarantee state control between the representations without the N step rollback.

The problem in this procedure is defining how to guarantee state control in the reverse direction and when to switch back to the FEA. As a simple definition, a model is defined as valid only over the known test conditions and not necessarily over the complete variable space. As shown in Figure 2-6, each region of a device operation can have different models $g(\theta_x)$ that are inaccurate outside the given region. Because of the region definitions, the models have a trigger condition to change between representations. If a region has been tested and the results have an error measurement below the given tolerances, then the model simulator can continue the analysis into the other regions. Otherwise, the region is assumed untested, and the reverse trigger condition occurs once the variable solution crosses a gray area between the regions. Then, the most accurate modeling representation is switched into the analysis and synchronized with an N-step roll forward or a state recovery or both. Continuity across the boundaries is also critical.

This region-based modeling approach complicates the modeling and switching

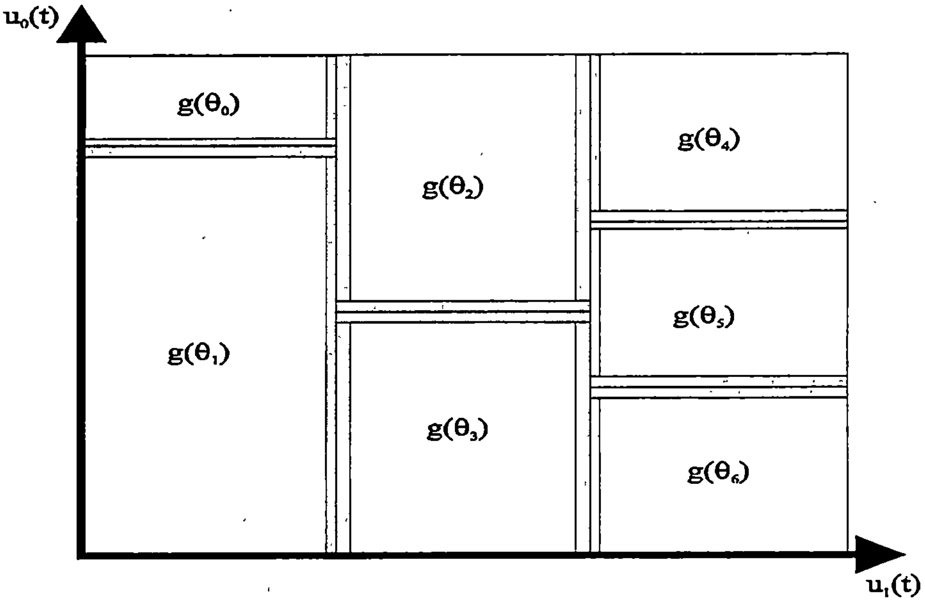


Figure 2-6. Region map of a device behavior in two variables

process. In a fully integrated environment, an array of different models with various degrees of accuracy and performance are available for this dynamic modeling process. This condition assumes that information is available about the accuracy and performance of each model in a simulator with all the region mappings. The advantages of this region-based process are a known accuracy, the potential to simplify the model definitions in a region, and complete region verification for eliminating re-simulation. A region check-off and verification procedure is integrated into the modeling process.

2.5.3 Model Definitions

With region-based models, the model generation process can become an integral part of the approach. This consideration is beyond the scope of this work, but different options are examined. Model definitions can be generated automatically using FEA for a specific simulation environment, but the problem is defining the stimuli to test consistently all operational regions without user interaction. To implement the model definition process, the three modeling methods using the backplane are a coupled solution, independent analysis, and hybrid analysis.

The coupled solution uses the concurrent results from the analysis to construct a model, but several flaws are apparent in the process. An incomplete analysis and lack of model integrity over the complete operational region are potential problems. The distribution of the solution data can skew the model without data filtering, so the model can be inaccurate in regions with few test points. The region mapped by the coupled simulation cannot be absolutely “checked off” from the total variable space without complete verification over all conditions.

In the independent analysis, a complete region mapping of the device's behavior is possible where a predefined stimuli sequence is applied to more accurate but time-consuming model to create a reduced order model [67]. The process guarantees the model integrity, since region mapping is done over all possible stimuli conditions. A single model representation is possible [89] where a common error statistic governs the model to eliminate the need for an accuracy trigger. However, generating the stimuli over all operational regions and conditions can be extremely difficult and lengthy process over all potential responses, and a poor mapping may result.

In the hybrid analysis, the coupling process assists the independent modeling process by defining the variable ranges to prevent the mapping of all possible regions. This consideration is a variation of the piecewise linear modeling, where each region is assumed linear in one variable and the stimuli can be applied one-at-a-time especially if the regions are sufficiently small. The problems with the region mapping process are defining the size of the regions and the modeling method. The one significant advantage of this hybrid approach is that the model generation process can be implemented in parallel with the coupling process where each new region found in the coupled analysis spawns a model generation process. Once the region mapping is completed, the models can then be used dynamically to trigger the representation control. This process can potentially minimize the amount of FEA needed on the devices while guaranteeing a high-degree of accuracy. If necessary, the different region-based models can be combined [89] into a single model instead of testing a single stimuli definition to cover all conditions.

2.6 Summary

This chapter has outlined several MEMS design, modeling, and simulation environments. In addition, the four main aspects of a general coupling process have been described with an emphasis on the iteration process and solution procedures. Finally, the dynamic modeling process has been presented for the hybrid coupling process for MEMS. A variation of the dynamic modeling process is outlined to ensure certain accuracy constraints and potentially to optimize the simulation performance and accuracy via automated model generation.

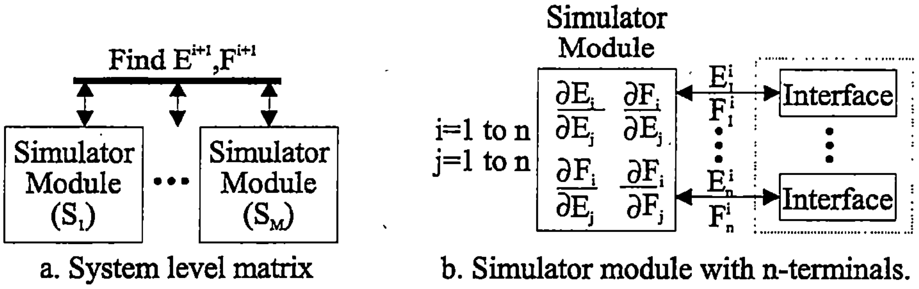
Chapter 3. Causality-based Interfaces

In a coupling process, multiple simulators are linked via nodes that exchange EFFORT (E) and FLOW (F) variables as shown in Figure 3-1(a). In the electrical domain, this relationship corresponds to Kirchoff's current law and voltage equality. Since other disciplines have similar relationships, a solution process was defined to include any simulator where power was exchanged based on generic EFFORT (E) and FLOW (F) variable definition, which is the bases of bond graph analysis techniques [91]. The two rules for n_T elements in a node required that

$$\sum_{j=1}^{j=n_T} I_j = 0 \qquad or \qquad \sum_{j=1}^{j=n_T} F_j = 0 \qquad \text{Equation 3-1}$$

$$V_1 = V_2 = V_3 = \cdots = V_{n_T} \quad or \quad E_1 = E_2 = E_3 = \cdots = E_{n_T} \qquad \text{Equation 3-2}$$

In this process, all simulators were black boxes, and the coupling process ultimately required the sensitivity information from the interface variables of each simulator as shown in Figure 3-1(b). The backplane was required to solve the coupling relationship based on the simulator's interface, which did determine what sensitivity parameters could be calculated and how the backplane interacted with a simulator.



a. System level matrix

b. Simulator module with n-terminals.

Figure 3-1. System-level simulation matrix representation

Most simulators defined policies that forced the user to define the variable causality (input or output definition of the variables in the node). Within the backplane, causality was not necessary known except if an interface constrained either the EFFORT or FLOW variable. Of course, the interface could violate the variable causality required by the design [87]. Usually, the partitioning of the design eliminated this problem, but this research did not have this information prior to simulation. The initial consideration was to implement dynamic interface configuration where **the goal was to find the interface configurations that optimized the iteration efficiency and convergence of the coupling process.** The known requirement of this goal was a flexible interface definition as described in section 3.1. Causality assignment in section 3.2 was the procedure considered for implementing a “smart interface” to maximize coupling efficiency.

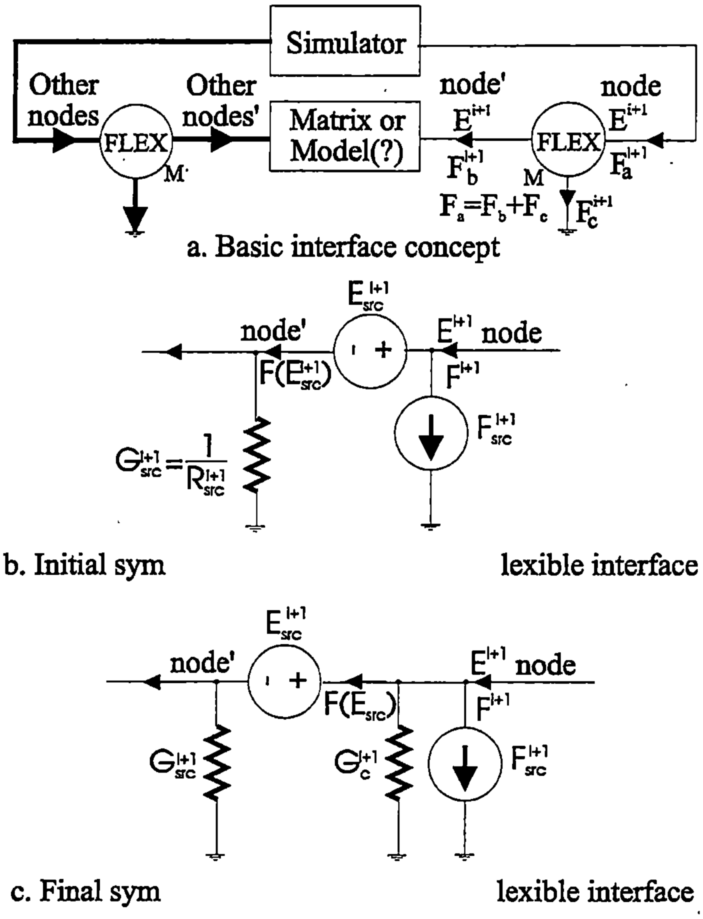
This research began by examining interfaces that directly solve Equation 3-1 and 3-2 as described in section 3.3, where simple voltage and current source definitions were used directly to implement the two rules in a sequential process. Unfortunately, these simple element definitions adversely affected the iteration efficiency and caused divergence if the wrong type of source was used [32,87]. The premise of this work became that causality assignment with interface reconfiguration could improve the iteration process by identifying the particular element controlling the node variables. For example, the interface definition did have a small impact on the performance of device-level analysis [30]. The direct approach did eliminate the sensitivity parameter calculations, which was a computationally expensive part of the coupling process, but the approach was unsuccessful.

After the failure of the direct calculation interface, the causality assignment process was applied to the conventional interfaces using more formal calculation procedures as described section 3.4. Conventional interfaces required the calculation of the sensitivity parameters and determination of the proper input sensitivity variables. To enhance the iteration process, a flow variable correction method was examined in section 3.5. This particular method overlapped the different simulators using behavioral models for the external devices. One simulator with the behavioral model assumed the role of a system-level simulator with a complete system-level definition while the other simulators became device or module simulators. The behavioral model operated as a predictor function [90] that was corrected by the external device simulator. This configuration was better suited to the dynamic modeling approach, since the corrector function could be changed or eliminated for the coupling process. Various versions of the correction process were considered to eliminate the sensitivity calculations and to improve performance.

3.1 Interface Definition

For implementing the causality tests and sensitivity calculations, an interface configuration was required with sufficient flexibility to define an effort or flow. This element had to monitor the flow into any device model for the correction process or an equivalent matrix created by the backplane for a simulator. Initially, this work attempted to define an interface that had all characteristics of both an effort or flow source, so any aspect of power can be absolutely controlled. However, the ability to control both variables implied that the process controlled power, which was not possible without eliminating the iteration process with the simulator. The basic coupling concept is shown in Figure 3-2(a)

The initial symbolic representation of the flexible interface concept was a combination of a Thevinin and Norton equivalent interfaces from the electrical domain as shown in Figure 3-2(b). The backplane calculations defined the conduction G_{SRC}^{i+1}, effort E_{SRC}^{i+1}, and flow F_{SRC}^{i+1}. The simulator returned a solution effort E^{i+1} from the flow source and the partial flow $F(E_{SRC}^{i+1})$ through the effort source. Eventually, the flexible interface was changed to a Norton equivalent representation with a Thevinin equivalent



a. Basic interface concept

b. Initial sym lexible interface

c. Final sym lexible interface

Figure 3-2. Concept and structures for the flexible interface element

monitor as shown in Figure 3-2(c). This representation was used to indicate that the backplane calculated the flow through the conduction element for the Norton interface. Considerations were also made to use the conduction element G_C for calculating sensitivity parameters of internal models, but no practical usage for this element was found.

3.2 Causality Detection

Causality assignment is defined as the process of assigning inputs and outputs to subsystems and possibly in an arbitrary manner [91, pp.143-154]. This particular statement applies to bond graphs, but the same principle can be applied to simulator coupling especially if one simulator defined the response to another simulator. Initially, the causality assignment process was conceived as the flow diagram in Figure 3-3. The strategy for this process was to identify the dominant element in a node as the CAUSE element, and the other elements followed the operation of the CAUSE element. Specifically, the CAUSE element defined the EFFORT stimulus to the EFFECT elements in sequential manner. The CAUSE element was a FLOW-based configuration while the EFFECT element was an EFFORT-controlled configuration. This convention was maintained throughout the research.

The GENERIC element was the default interface configuration in this process until the CAUSE element was identified. Four transition states were also included between the three basic elements to ensure that the conditions for CAUSE and EFFECT persisted for at least two iterations. The most difficult task in this process was defining the conditions for the state changes in Figure 3-3. At this point, the only available information for the

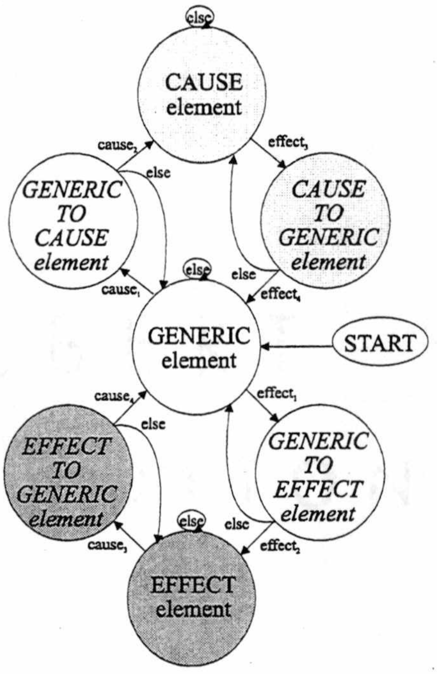


Figure 3-3. State diagram for causality detection

detection process was the element's power and sensitivity contribution to the node.

3.2.1 Absolute Power Criterion

The absolute power criterion was not a very robust detection mechanism because power in a node sums to zero. The absolute power criterion compared an element's *absolute* power contribution to the *absolute* power of the other node elements by using a power ratio (α_{PW}) and a state-based power multiplier (α_S). The power ratio was the primary parameter that defined the absolute power ratio to make a CAUSE or EFFECT decision. The state-based power multiplier was a secondary variable that changed the range of absolute power ratio to ease the transition process between element types. When an element's absolute power was $\alpha_S\ \alpha_{PW}$ times larger than the maximum absolute power of an external element, the element followed the CAUSE paths in Figure 3-3, or

$$|E_1 \cdot F_1| \geq \alpha_S \cdot \alpha_{PW} \cdot MAX(|E_j \cdot F_j|) \quad \forall \quad j = 2 \ldots n_T \qquad \text{Equation 3-3}$$

When the element's absolute power was $\alpha_s \alpha_{PW}$ times smaller than the maximum absolute power of an external element, the element followed the EFFECT path in Figure 3-3, or

$$|E_1 \cdot F_1| \leq \alpha_S \cdot \alpha_{PW} \cdot MAX(|E_j \cdot F_j|) \quad \forall \quad j = 2 \ldots n_T \qquad \text{Equation 3-4}$$

In most cases, the criteria only identified elements with minimum power contributions to a node and the criterion was only stable when two elements were coupled because of the potential overlap in the detection criteria. In this procedure, a CAUSE element was not always the maximum power element and the EFFECT element was not always the minimum power element in the node.

Because of the potential overlap between CAUSE and EFFECT, three rules were required to make the causality detection rules consistent. Based on bond graphs [91], the three rules were:

1. Only one CAUSE element can exist per node, and other elements must be an EFFECT.
2. Without a CAUSE element, two GENERIC elements are required in a node.
3. A node cannot contain just EFFECT elements.

These three rules prevented elements from improperly transitioning into a state where the iteration function stopped or the solution diverged. The violation of any rule forced the two elements contributing the maximum absolute power to the node to become GENERIC elements. By choosing the two elements with the largest absolute powers, the dominant elements in the node were identified.

In most situations, the absolute power criterion was completely unacceptable. However, the calculation procedure in section 3.3 was able to identify causality using the absolute power criteria when peaks or inflection points occurred in the response. An inflection point in the time response represented a change in the transient behavior of an element where the time derivative of the element's effort was reversed from the previous transient conditions. The element causing the inflection point had to supply the power to change the transient behavior of the other element(s), so the element's power increased. The additional power forced the other element(s) through the previous solution where the integration process encountered a local minimum and the power of the other element(s) decreased. Thus, causality was detected, but the process was not very reliable. The combination of the power ratio, signal magnitude, and time-step determined if an inflection point could be detected. In many situations, the iteration sequence corrected the power difference between the elements before the criteria recognized a peak in the response.

3.2.2 Sensitivity Criterion

The sensitivity criterion was similar to the absolute power criteria. For this criterion, the absolute internal conduction parameter of an element was compared to the absolute conduction from the other elements by using a state-based ratio (α_R). When an element's conduction was α_R times larger than the maximum internal conduction of other simulators, the element began to follow the CAUSE paths in Figure 3-3, or

$$\left|G_1^i\right| > \alpha_R \cdot MAX\left(\left|G_j^i\right|\right) \quad G_j^i = \left.\frac{\partial F}{\partial E}\right|_j^i \quad j = 2 \ldots n_T \qquad \text{Equation 3-5}$$

When the element's conduction was α_R times smaller than the maximum conduction of other simulators, the element began to follow the EFFECT path in Figure 3-3, or

$$\left|G_1^i\right| < \frac{1}{\alpha_R} \cdot MAX\left(\left|G_j^i\right|\right) \quad j = 2 \ldots n_T \qquad \text{Equation 3-6}$$

In this criterion, the CAUSE element was always the dominant element in the node with the largest sensitivity parameter. The two causality definitions did not overlap and only one CAUSE assignment was made, so no additional rules were required to prevent assignment problems.

3.3 Direct Calculation Interface

The direct calculation interface attempted to solve Equation 3-1 and 3-2 directly using only the present flow and effort values, but these interfaces had divergence and non-convergence problems [32]. Since the interfaces were not very reliable, an ***attempt*** was made to improve the process. The experiment started with the problem in Figure 3-4 using two SPICE simulators. The possibility of using causality assignment was realized after analyzing this problem using both combinations of current source and voltage source with the direct relationships.

In the experiment, the VOLTAGE$_{C1}$-CURRENT$_{C2}$ configuration found the correct initialization point for the node, but the transient analysis diverged when the stimulus V$_{in}$ started changing at time t$_1$. The CURRENT$_{C1}$-VOLTAGE$_{C2}$ configuration found the wrong initialization point, but the dynamic solution converged and the result had the same

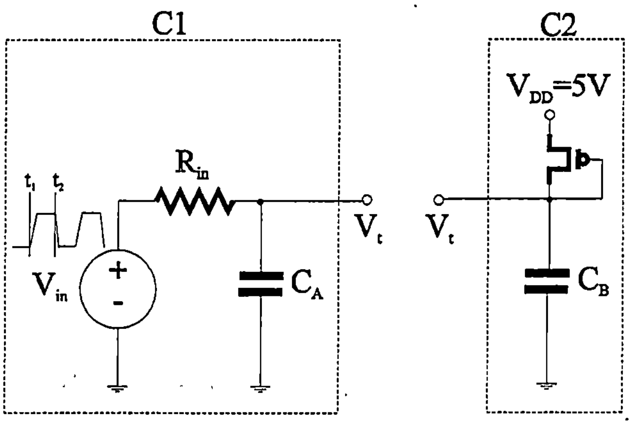


Figure 3-4. Example circuit to test causality rationale

waveform characteristics as the complete system solution obtained from SPICE. One configuration was statically correct and the other solution was dynamically correct. Logically, an intelligent combination of the two results produced the correct solution. Based on this experiment, causality was considered a dynamic and not a static property. The initial premise was that a *pure current (FLOW) and VOLTAGE (EFFORT) configuration using direct calculations was possible for a node, if the elements in the node are* ***correctly coordinated and configured at the proper times.*** A better explanation was that the interface procedure had an error, but this observation became the driving rationale behind this research.

In this example, variation of the resistance value and the transistor's width and length did cause divergence in the coupling process. The simple source definitions did not have the capability to find a stable point since the iteration process was one-dimensional. A stable bias point was discovered to be a requirement for a direct configuration. To overcome this problem, a GENERIC element was defined as outlined in section 3.3.1. The initial GENERIC definition converged to a solution, but the rate of convergence was slow (7 to 11 iterations per step) and different speedup conditions were tested to improve convergence.

During the testing of the GENERIC node, the speedup considerations improved the convergence rate, but a "sticky point" was found with slow convergence. Within this sticky point, the iteration process forced the power of one element to approach zero, and the realization was made that the absolute power criteria could be applied for transiting to the pure CAUSE and EFFECT arrangements. This causality identification did improve the overall rate of convergence, and the interface process appeared viable based on this example. Different test situations were later examined, and the process only converged when a node had two elements with NO FEEDBACK. Certain situations also caused divergence when causality was incorrectly identified. To account for all the various exceptions and divergence possibilities, this procedure had to implement additional rules, which eventually eliminated the causality detection. As described in section 3.3.2, this interface was a terrible approach to the coupling problem.

3.3.1 Averaging Interpolation Interface

The calculation process for the GENERIC element was required to define the source values as variations of the present iteration values ONLY. The variable deltas were calculated by simple averaging. All elements were moved toward a common average voltage value and a common current difference was applied across the different elements. The resistance was defined as a ratio of the node variations. In the electrical domain, the node elements were defined by

$$I_{SRC}^{i+1} = I_1^i + \Delta I_1^i \quad \Delta I_1^i = \frac{1}{n_T}\sum_{j=1}^{n_T} I_j^i \qquad \text{Equation 3-7}$$

$$V_{SRC}^{i+1} = \frac{1}{n_T}\sum_{j=1}^{n_T} V_j^i = V_1^i + \Delta V_1^i \quad \Delta V_1^i = \frac{1}{n_T}\sum_{j=1}^{n_T} V_j^i - V_1^i \qquad \text{Equation 3-8}$$

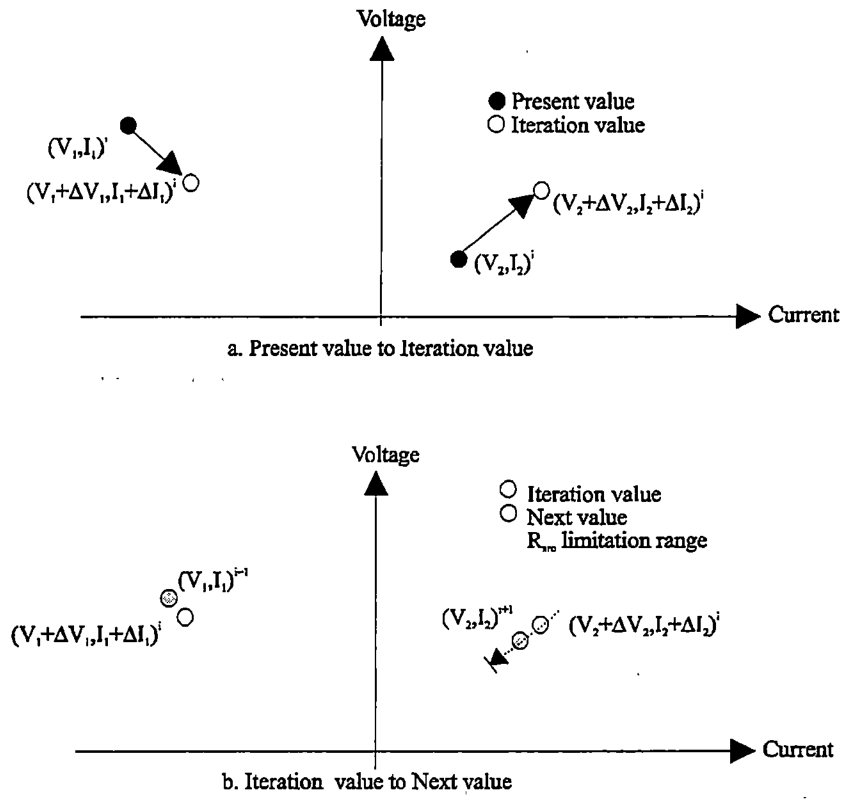
$$R_{SRC}^{i+1} = R_{VAR} = \frac{\Delta V_1^i}{\Delta I_1^i} \qquad \text{Equation 3-9}$$

The calculation process was a form of the secant method and a crude approximation of the sensitivity parameters in a conventional interface, where

$$\begin{gathered} I_1^{i+1} = I_{SRC}^i + I(V_{SRC}^{i+1}) = I_1^i + \Delta I_1^i + \frac{V_1^{i+1} - \left(V_1^i + \Delta V_1^1\right)}{R_{SRC}^i} \\ \frac{I_1^{i+1} - I_1^i}{\Delta I_1^i} = \frac{V_1^{i+1} - V_1^i}{\Delta V_1^i} \end{gathered} \qquad \text{Equation 3-10}$$

Because of the averaging, this procedure stepped toward the solution. The variation terms changed the power to a simulator element as shown in Figure 3-5(a). In the simulator, the voltage and current calculation for the element returned toward the previous solution with a rate that depended on the dominance of the element as shown in Figure 3-5(b).

This resistance calculation was a very poor approximation of the normal linearization process and "sticky" points occurred where the convergence rate slowed or stopped. The process was very sensitive to whether one variable delta converged faster than the other delta. As ΔI_1^i approached zero, the resistance R_{SRC}^{i+1} became very large, and the voltage variation from the voltage source had no effect on the response. When the ΔV_1^i approached zero, a small resistance value was created where the voltage source dominated the response and absorbed the current variation. The options for eliminating this problem were applying one variable at a time, redefining the variation terms, or redefining the resistance definition. The choice was made to redefine the resistance.


Voltage
● Present value
○ Iteration value
$(V_1,I_1)^i$
$(V_1+\Delta V_1,I_1+\Delta I_1)^i$
$(V_2+\Delta V_2,I_2+\Delta I_2)^i$
$(V_2,I_2)^i$
Current
Voltage
○ Iteration value
○ Next value
R_{ro} limitation range
$(V_1,I_1)^{i-1}$
$(V_1+\Delta V_1,I_1+\Delta I_1)^i$
$(V_2,I_2)^{i+1}$
$(V_2+\Delta V_2,I_2+\Delta I_2)^i$
Current

a. Present value to Iteration value

b. Iteration value to Next value

Figure 3-5. Iteration process for the GENERIC element

Two additional resistance values were defined: differential resistance R_{DIFF} and an absolute resistance R_{ABS}. The differential resistance was a dynamic resistance calculated based on a time derivative approximation where the resistance was the slope between the present iteration value and the previous event point. The differential resistance was defined as

$$R_{DIFF} = \begin{cases} \dfrac{V_1^i(t_{now}) - V_1\,(t_{prev})}{I_1^i(t_{now}) - I_1\,(t_{prev})} = \dfrac{\Delta V_{DIFF}}{\Delta I_{DIFF}} & \text{if } \Delta I_{DIFF} \neq 0 \\ R_{MAX} & \text{otherwise} \end{cases} \qquad \text{Equation 3-11}$$

The absolute resistance was an equivalent load resistance from the node to ground where

$$R_{ABS} = \begin{cases} \dfrac{\left|V_{SRC}^{i+1}\right|}{\sum_{j=1}^{j=n_T}\left|I_j^i\right|} & if \sum_{j=1}^{j=n_T}\left|I_j^i\right| \neq 0 \\ R_{MAX} & otherwise \end{cases} \qquad \text{Equation 3-12}$$

In this process, the decision was also made to use the smallest resistance value as R_{SRC}^{i+1} to guarantee that the voltage was converging faster than the current since voltage is common between all elements of a node. However, the process still required R_{SRC}^{i+1} to be redefined in certain situations as outlined in Table 3-1.

During the testing of this procedure, another sticky point was discovered at the peaks in the voltage response. After analyzing the cause of this slow convergence, the peak detection criterion described in section 3.2.1 was realized and the effect was traced to the minimum resistance criterion using the differential resistance R_{DIFF}. At the peaks, the differential resistance was typically smaller than the normal and absolute resistance values, which became large due to the divergence of the responses. In this situation, the averaging process pulled the different responses back toward the previous solution, while each element attempted to continue the response in the same direction due to the integration process of the simulator. Slowly, the CAUSE element supplied the power necessary to change the response directions of the other elements.

Table 3-1. Resistance R_{SRC}^{i+1} definition under different conditions

Conditions	Current Convergence	Otherwise
Voltage Convergence	R_{SRC}^{i}	$\max\left(R_{DIFF}, R_{ABS}, R_{VAR}\right)$
Otherwise	$\min\left(R_{DIFF}, R_{ABS}, R_{VAR}\right)$	$\min\left(R_{DIFF}, R_{ABS}, R_{VAR}\right)$

3.3.2 Causality Recognized

Using peak detection, the pure CAUSE and EFFECT elements were incorporated into the approach to improve the iteration efficiency of the GENERIC element. This combination was briefly successful and the iteration efficiency was improved for the given example. The parameter values for the test case defined α_{PW} with a value of 10 to guarantee that the response of the node is at a peak. Smaller values of the power ratio caused the elements to oscillate between the different states. In addition, the values for α_S are 0.5 in the transition states compared to 1.0 in the causal states. Unfortunately, the same procedures failed on other examples. Several serious interfacing problems were discovered, and this process was later proven to diverge under certain situations.

A third simulator demonstrated interface inconsistencies due to the overlap regions in the causality criteria, and the overlap rules in section 3.2.1 had to be created to correct these problems. Other test cases showed the divergence nature of incorrectly defining causality, and a divergence detection mechanism became necessary. Fortunately, divergence has a distinct characteristic where the solution grows to unrealistic values in an alternating sequence. Whenever divergence was detected, an internal flag was set in each node to prevent the detection of causality and all elements were forced to a GENERIC state. With all the different conditions, the interface had become too complex, and the calculation procedures were not consistent between the different element types i.e. a common mathematical calculation procedure was lacking. As additional simulators were added to a node, this procedure was unable to find a solution. At this point, the direct calculation approach was abandoned and a more conventional approach was considered.

This procedure had attacked the problem based on individual nodes, which was not sufficient for a system with feedback. As an example of the causality divergence problems, a simple state feedback loop in Figure 3-6 was examined to show divergence potential and feedback gain requirements with causality recognition. The two gain blocks had a large output conduction value and small input conduction value, so causality was easily determined. The Laplace gains were simplified to a constant gain for both the initialization (DC bias) and transient analysis. The initial conditions were assumed zero. The basic iteration process for this example was defined by

$$\begin{aligned} x^i &= G_1(z + y^{i-1}) \\ y^i &= G_2 x^{i-1} \end{aligned} \qquad \text{Equation 3-13}$$

The expansion of the iteration process revealed that

$$\begin{aligned} x^i &= z\left(G_1 + \frac{1-(G_1G_2)^{i/2-1}}{1-G_1G_2}\right) \quad i > 2 \\ y^i &= G_2 z\left(G_1 + \frac{1-(G_1G_2)^{i/2-1}}{1-G_1G_2}\right) \quad i > 2 \end{aligned} \qquad \text{Equation 3-14}$$

This iteration process converged if and only if $|G_1G_2| < 1$ or if $z = 0$. These requirements were the most critical during the initialization procedure because the interface solutions

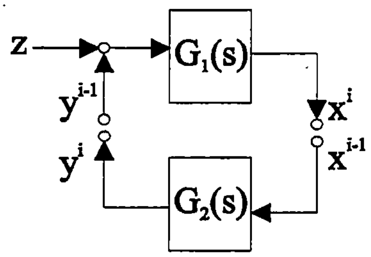


Figure 3-6. Simple feedback system to examine the iteration process

were not stable. For the transient analysis, a "sufficiently small" timestep was required for the effective loop gain to meet this criterion. Since these conditions cannot be guaranteed, this calculation procedure cannot be considered for the general coupling process.

3.4 Conventional Interfaces

Most conventional interface definitions were based on nodal or modified nodal analysis [31,77], which implied Norton equivalent interfaces and representations. Initially, the Norton equivalent interface for each simulator was solved as shown in Figure 3-7 for n simulators. To define the equivalent circuit, the sensitivity parameters (Equation 3-15) were used as the conduction definition for an element. The Thevinin or Norton equivalent circuit [29] for a particular simulator (j) at the present iteration (i) was defined by

$$G_{EQ}\Big|_j^i = \frac{\partial F}{\partial E}\Big|_j^i = \left(R_{EQ}\Big|_j^i\right)^{-1} \qquad \text{Equation 3-15}$$

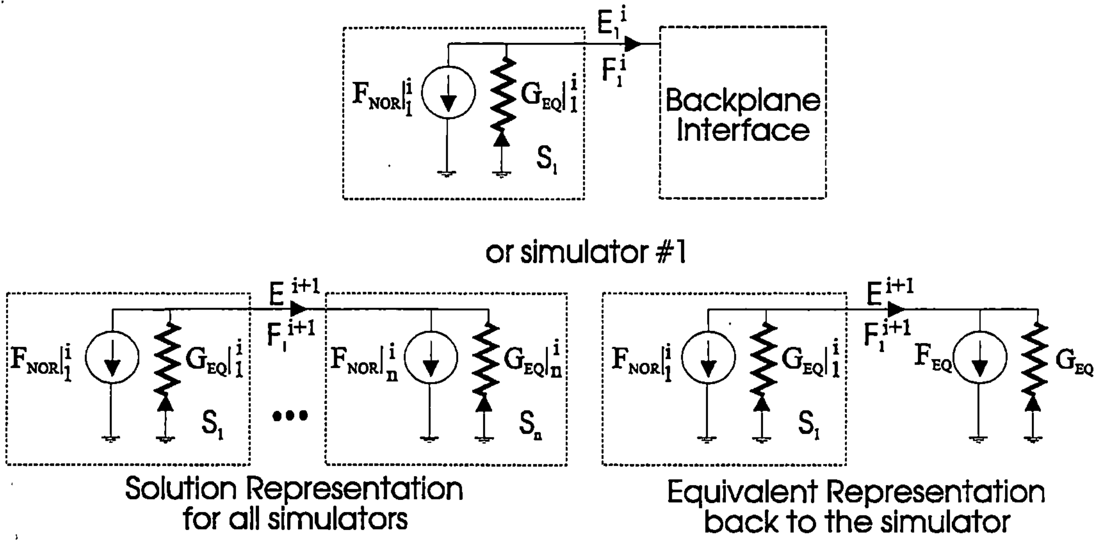


Figure 3-7. Different equivalent representations for a simulator or simulator node

$$E_{TH}\Big|_j^i = E_j^i - R_{EQ}\Big|_j^i \cdot F_j^i$$
$$F_{NOR}\Big|_j^i = G_{EQ}\Big|_j^i \cdot E_j^i - F_j^i$$

Equation 3-16

The solution for the effort and flow values (CALC subscripts) was defined as

$$\begin{aligned} E^{i+1}\Big|_{CALC} &= \left(\sum_{j=1}^{j=n_T} G_{EQ}\Big|_j^i\right)^{-1} \cdot \sum_{j=1}^{j=n_T} F_{NOR}\Big|_j^i \\ &= \left(\sum_{j=1}^{j=n_T} G_{EQ}\Big|_j^i\right)^{-1} \cdot \sum_{j=1}^{j=n_T} \left(G_{EQ}\Big|_j^i \cdot E_j^i - F_j^i\right) \end{aligned}$$

Equation 3-17

$$\begin{aligned} F_j^{i+1}\Big|_{CALC} &= - F_{NOR}\Big|_j^i + G_{EQ}\Big|_j^i \cdot E^{i+1}\Big|_{CALC} \\ &= F_j^i + G_{EQ}\Big|_j^i \cdot \left(E^{i+1}\Big|_{CALC} - E_j^i\right) \end{aligned}$$

Equation 3-18

To define a Thevinin or Norton equivalent circuit for the interface, the following equivalent relationships was required:

$$G_{EQ}^i = \sum_{j=2}^{j=n_T} G_{EQ}\Big|_j^i = \left(R_{EQ}^i\right)^{-1}$$

Equation 3-19

$$F_{EQ}^i = \sum_{j=2}^{j=n_T} F_{NOR}\Big|_j^i = \sum_{j=2}^{j=n_T} \left(G_{EQ}\Big|_j^i \cdot E_j^i - F_j^i\right)$$

Equation 3-20

In these equations, the local simulator was always referenced as 1. These equations applied to both matrix operations and singular nodes.

This solution process was not very effective when the conduction parameters became extremely large and dominated the solution as rounding errors in the EFFORT calculations often resulted in large errors in the flow variables. The flow variables were added to the solution process and Newton-type iteration method was also defined. The new solution process was

$$\begin{bmatrix} E^{i+1} \\ F_1^{i+1} \\ \vdots \\ F_n^{i+1} \end{bmatrix}_{CALC} = \begin{bmatrix} E^{i} \\ F_1^{i} \\ \vdots \\ F_n^{i} \end{bmatrix}_{CALC} - \begin{bmatrix} 0 & p_i & \cdots & p_n \\ G_{EQ}\big|_1^i & 1 & \cdots & 0 \\ \vdots & 0 & \ddots & 0 \\ G_{EQ}\big|_n^i & 0 & \cdots & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ \Delta F_1 \\ \vdots \\ \Delta F_n \end{bmatrix} \qquad \text{Equation 3-21}$$

$$\Delta F_X = G_{EQ}\big|_j^i \cdot \left(E^i\big|_{CALC} - E_X^i\right) - \left(F^i_X\big|_{CALC} - F_X^i\right) \qquad \text{Equation 3-22}$$

The interconnection matrix (p_x) was a unity-based matrix that defined which flow variables summed to zero for a particular effort to implement Equation 3-1.

This iteration process was considered for the example in Figure 3-6, but the proper sensitivity parameters had been not calculated and effort gain parameters were required. Normally, the Newton-Raphson method was applied over all variables at the system-level if sufficient functions were available that were equal to zero, but the only known equation between simulators was Equation 3-1. To overcome this problem, causality assignment was required to define the inputs and outputs variables from a simulator, since nodal analysis had treated voltage as the input and current as the output. By changing causality, the procedure calculated a different set of sensitivity parameters. The sensitivity functions (system of equations) defined the relationship between inputs and outputs for a given simulator where

$$S_{Output}(\Delta Input) = \sum_j \frac{\partial Output}{\partial Input_j} \Delta Input_j - \Delta Output = 0 \qquad \text{Equation 3-23}$$

Using these sensitivity relationships, the backplane had the functions necessary to couple the simulators. The problem was deciding which variables were inputs and what sensitivity parameters had to be calculated. The input variable detection process is described in

Appendix-B.

As causality changed, the backplane had to vary the sensitivity parameters and solution matrix. An example of the final matrix representation was Equation 3-24, which shows the sensitivity information for the both EFFORT and FLOW output variables and how the interconnectivity information was used. In Equation 3-24, each interface uniquely defined one row of the solution matrix where n flow variables and one effort variable was defined for each node (domain interface). The row for the effort variable defined the flow interconnectivity relationship while the row for each FLOW variable used the simulator function calculated in Equation 3-23.

$$\begin{bmatrix} E_1^{i+1} \\ F_{1,1}^{i+1} \\ \\ E_n^{i+1} \\ F_{n,1}^{i+1} \\ \\ F_{n,a}^{i+1} \end{bmatrix} = \begin{bmatrix} E_1^{i} \\ F_{1,1}^{i} \\ \\ E_n^{i} \\ F_{n,1}^{i} \\ \\ F_{n,a}^{i} \end{bmatrix} - \begin{bmatrix} 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \frac{\partial F_{1,1}}{\partial E_1} & 0 & \cdots & \frac{\partial F_{1,1}}{\partial E_n} & \frac{\partial F_{1,1}}{\partial F_{n,1}} & \cdots & \frac{\partial F_{1,1}}{\partial F_{n,a}} \\ & & \ddots & & & & \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 1 \\ \frac{\partial E_n}{\partial E_1} & 0 & \cdots & \frac{\partial E_n}{\partial E_n} & \frac{\partial E_n}{\partial F_{n,1}} & \cdots & \frac{\partial E_n}{\partial F_{n,1}} \\ & & \cdots & & & \cdots & \\ \frac{\partial F_{n,a}}{\partial E_1} & 0 & \cdots & \frac{\partial F_{n,a}}{\partial E_n} & \frac{\partial F_{n,a}}{\partial F_{n,1}} & \cdots & \frac{\partial F_{n,a}}{\partial F_{n,a}} \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ S_{F_{1,1}^i} \\ \\ 0 \\ S_{E_n^i} \\ \\ S_{F_{n,a}^i} \end{bmatrix} \qquad \text{Equation 3-24}$$

$$S_Y = \sum_j \frac{\partial Y}{\partial X_j}(X_{j,CALC} - X_{j,SIM}) - (Y_{CALC} - Y_{SIM}) \quad X \in Inputs \qquad \text{Equation 3-25}$$

This procedure was error minimization between the calculated variables and the simulator values as defined in Equation 3-25. Equation 3-25 defined the error functions to be minimized by Equation 3-24.

The final issue for this procedure was the calculation of the equivalent sensitivity matrix at the system-level. In situations where only nodal analysis was used, Equation 3-19 was sufficient to calculate the sensitivity information. However, a variation of Equation 3-

24 was needed account for the EFFORT output conditions. For this equivalent calculation, the basic solution matrix was modified to isolate effort variations by forcing a known delta value into each effort variable independently. For this procedure, all local simulator contributions were eliminated from the solution matrix. The rows of the EFFORT variables were set to one for each EFFORT variable while the local simulator contributions were redefined as the interconnect relationships. The solution process for the sensitivity parameters for a particular EFFORT variable used Equation 3-26.

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ & & \ddots & & & & \\ 0 & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 1 \\ & & \cdots & & & \cdots & \\ \frac{\partial F_{n,a}}{\partial E_1} & \frac{\partial F_{n,a}}{\partial F_{1,1}} & \cdots & \frac{\partial F_{n,a}}{\partial E_n} & \frac{\partial F_{n,a}}{\partial F_{n,1}} & \cdots & \frac{\partial F_{n,a}}{\partial F_{n,a}} \end{bmatrix}^{-1} \begin{bmatrix} \Delta E_1 \\ \Delta F_{1,1} \\ \\ \Delta E_{nl} \\ \Delta F_{n,1} \\ \\ \Delta F_{n,a} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \\ 0 \\ 0 \\ \\ 0 \end{bmatrix} \qquad \text{Equation 3-26}$$

$$\frac{\partial F_x}{\partial E_y} = \frac{\Delta F_x}{\Delta E_y} \qquad \text{Equation 3-27}$$

The equivalent sensitivity parameters were calculated using Equation 3-27 for each effort variable y to a local flow variable x that required the information. Certain interfaces did not require equivalent sensitivity information. In addition, the equivalent matrix was calculated only when an external simulator's sensitivity information changed. A tutorial of this calculation procedure is presented in Appendix E.

With the calculations defined, the different interfaces available in the backplane are briefly discussed. These interfaces were defined for applications in the electrical domain; so the typical electrical names were used. Appendix-B defined interfaces component values, emulation modes, and interfacing methodology. The dynamical interfacing process was also revised for the conventional interfaces and a "smarter" procedure was developed.

3.4.1 Flexible Interface

The flexible interface was a simple combination of the Thevinin and Norton equivalent representations where the basic interface was used in a Norton format. This interface forced the simulator to calculate the feedback flow variable when a conduction matrix or behavioral model was integrated into the simulator. This interface emulated all interface types for the causality assignment process and was the most generic interface available. The Flexible interface was also the only structure with a system configuration, where the simulator loaded an external matrix to complete all feedback paths. The system configuration was essentially a matrix version of a Norton implementation.

3.4.2 Norton and Thevinin Equivalents Interfaces

Both the Norton and Thevinin configurations were classified as GENERIC elements although the Norton was more appropriately a FLOW-based interface and the Thevinin was an EFFORT-based interface. The Norton equivalent configuration was the conventional interface structure for an electrical application [31] since most electrical matrixes used in nodal analysis are defined in terms of conduction parameters and current sources. Both equivalent configurations allowed variations in effort and flow variables, so any sensitivity parameters could be calculated.

3.4.3 Voltage and Current Source Interfaces

Both the voltage and current source interfaces restricted the iteration process to one variable. Consequently, certain sensitivity parameters were not calculated and these interfaces completely defined the variable causality. A current source was usable only with a simulator that output a voltage or had a resistive path to ground. Without a decoupled relationship to other nodes in the simulator, a pure current source was unable to calculate conduction parameters between different terminals. Voltage sources were the typical interface to device simulators because the system-level defined the voltage to the device. A voltage source was typically used for nodal analysis. Voltage sources had interaction problems with direct connections to virtual nodes and other voltage sources.

3.4.4 Direct Emulation

The direct configurations were residues of the direct calculation procedures, but were not real interfaces. Instead, these configurations were calculation modes within an interface where an EFFORTEQ (Equation 3-2) and FLOWSUM (Equation 3-1) were defined. Using these configurations, the FLOWSUM element defined an effort (E_{CAUSE}) to the EFFORTEQ elements in a sequential manner. The EFFORTEQ elements applied the effort and responded with a new flow, which was summed by the FLOWSUM element to calculate a new effort on the next iteration. These elements were very ineffective unless the dominance of the CAUSE element was a certainty and all restrictions outlined in section 3.3.2 were guaranteed.

3.4.5 Dynamic Interface Configuration

The initial approach for the dynamic interface procedure used the flowchart from section 3.2, but the process had flaws (see section 3.4.5.1). Eventually a new tiered process was developed (see section 3.4.5.2).

3.4.5.1 Basic Causality Modifications

The dynamic interfacing process from section 3.2 had several errors when different combinations of conventional ***configurations*** were coupled. The problems were the result of poor usage of the system configuration and poor initialization. Several basic modifications were required:

1. The detection of the FLOW input for an interface required the interface to be reconfigured immediately as a CAUSE and other interface(s) in the node became an EFFECT(s).
2. To use a system configuration, the GENERIC State in Figure 3-3 had to be a system configuration for proper initialization. Because of modification #1, both the GENERIC and CAUSE States had to use a system configuration.
3. At initialization, sensitivity parameters were checked to determine if the interface exceeded the normal sensitivity rule by more than a factor of 100. If this condition was met, then the interface was immediately changed to a CAUSE or EFFORT configuration to improve convergence.

The FLOW input detection also placed constraints on the Causality State of other interfaces when a system configuration interacted with other interface types. With a system configuration and a FLOW input interface detected in a simulator, the EFFORT input interfaces in the local simulator interfaces had to examine EFFORT-to-EFFORT parameters for gain conditions greater than one to the FLOW input interface. If the gain condition was detected, then the interface was required to remain in a system mode to complete the feedback paths. The poor handling of the system interaction problems was the final reason that a new state diagram was developed.

3.4.5.2 New Causality State Diagram

The flexibility of defining different interface configurations for the three different states was unnecessary with the proper rules. A new state process was developed as a tiered system with fixed interface types as shown in Figure 3-8, and only three configurations of the flexible interface were used (highest to lowest priority): system, Norton (FSRC_SRC), and Thevinin (ESRC_SRC). This process used the same causality rules as the first method, but additional considerations were given for gain situations. This procedure also started in the system configuration to enhance the detection of EFFORT gain. The definitions for entering the different states were better defined in this procedure.

The two system states were SYS_LOCK and SYS_GENERIC. The SYS_LOCK

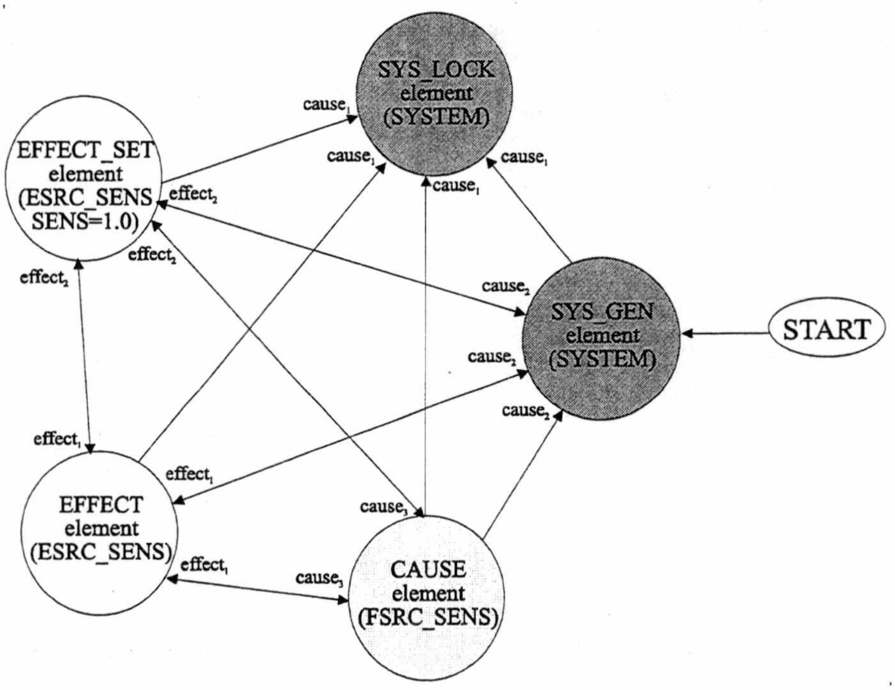


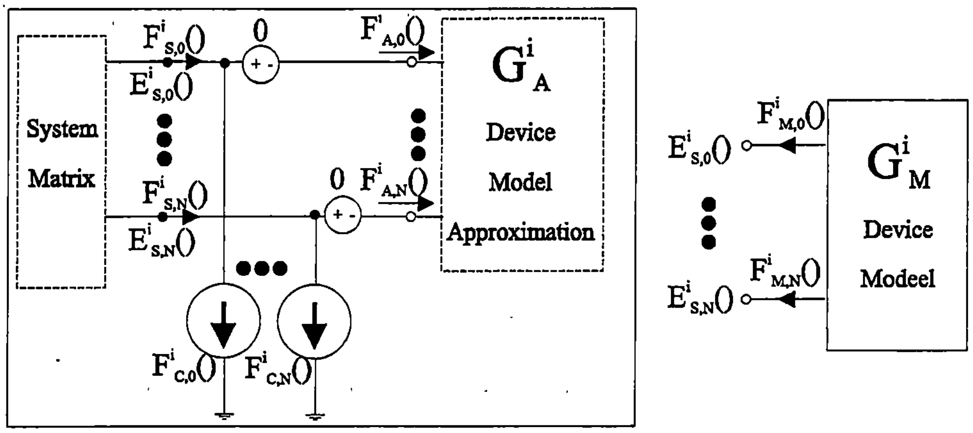
Figure 3-8. New causality state diagram

State indicated that a simulator had an EFFORT gain above an internal limit (50.0). Once this condition had been detected, the interfaces (the FLOW input interface and the interface that had the gain condition) remained in this state. In the initial method, the procedure allowed the interface to enter the EFFECT State if the effect condition was detected, which could cause failures in the coupling process. The SYS_GENERIC State indicated that a FLOW variable had been detected.

In this process, a very clear distinction was made between the CAUSE and SYS_GENERIC States, which were identical in the original procedure. The CAUSE State could only be reached by entering one of the EFFECT States, which indicated singular coupling in the simulator to the interface. The CAUSE State was ONLY included in this procedure to indicate that singular coupling had been detected. The EFFECT_SET State was a variation of the EFFECT State where the sensitivity component was set to 1.0. In this case, the equivalent sensitivity component was sufficiently large to force the interface to truly track the backplane solution. Some hard limiting was also implemented because the ESRC_SENS interface did fail due to rounding calculation errors where the sensitivity values were small ($<10^{-6}$) and the simulator had to resolve a large ESRC value.

3.5 Flow Correction Interface

Using the correction method, flow sources were added to the device model(s) in the system-level simulator and effort sources became the interfaces for device-level simulators as shown in Figure 3-9. The effort sources in the flexible interface of the system-level simulator were needed to monitor the flows into the behavioral models since the model



System-level Device-level

Figure 3-9. Flow correction interface

definitions were not necessarily known to the backplane. The flow correction method was a variation of regression analysis where unknown parameters were modeled in terms of known variables [76,90]. In this configuration, the flow sources represented the influence of any applied stimuli, nonlinear effects, or undefined characteristics of a second model not in the present model. The behavioral model became a first-order corrector in an ARMAX model format [90] where a nonlinear corrector was applied externally for guaranteeing certain accuracy in the analysis [93]. The advantages of this method were the simplified interfaces and multiple corrector definitions that are possible for the dynamic modeling approach.

In this procedure, the system-level simulator can do more internal iterations using the behavioral model. A more accurate solution was found that was always "reasonably close" to the true solution to enhance convergence before an external iteration occurred. This enhancement was intended to eliminate the poor initialization conditions that can occur with the conventional interfaces where the "sufficiently close" requirement of most solvers could be violated. In addition, the system-level simulator also had the potential to partition the design more efficiently to improve performance because of a complete system definition.

As a negative feature, this process implemented causality assignment where the system-level simulator defined the effort to the device-level simulator(s). This process was sequential and was potentially slower than the other interfaces unless improvements occur in the iteration efficiency between the simulators. A 2x factor of improvement in efficiency of the sequential process guaranteed equivalent performance to a parallel process. Fortunately, this process was implemented in a parallel fashion after an initialization sequence since the basic calculation process was a variation of nodal analysis.

Initially, the correction method was considered for situations where the mechanical FE simulator was always coupled to an electrical simulation as a form of waveform relaxation. As just a coupling process, the regression method offered a simple means of repeating analysis without using FEA and without generating a new behavioral model. The flow source values emulated the previous response in an event-driven piecewise linear (PWL) mode without re-analyzing the device behavior in the external simulator. The system-level analysis potentially achieved the desired accuracy without performance degradation. In this process, a designer constructed stimuli waveforms in terms of the flow source values from different coupled analyses to test the system-level behavior in a meaningful manner using the flow equivalent mechanical stimuli. Of course, these analyses did become inaccurate as the sensor, stimuli, or electronics was modified, so this emulation process was not as useful as a regressed model. Still, the process was simple and fast after the initial coupling analysis was completed, and most simulators were able to use directly the information in this flow waveform format.

Finally, the flow correction method offered a corrector that can be any external source. By varying the corrector source, the dynamic modeling approach was effectively changing model definitions compared to the basic model in the system-level simulator. Of course, one option was not to define an external corrector with the internal model, so this approach simplified to the independent approach. As a second option, the corrector source could be a combination of multiple modeling sources to create an N-level corrector [90]. A higher performance model would perform multiple coarse corrections with the system-level simulator before the more complex model was used as a fine corrector. This consideration was made to limit the number of FE iterations for improving performance between the simulators, but was not implemented.

The calculation procedures for the flow correction process are defined in section 3.5.1 and the convergence of the process is examined in section 3.5.2. Several variations of calculation procedure are examined to improve performance by eliminating the sensitivity matrix calculations. However, these variations have limitations depending upon the accuracy of the behavioral model. Several modifications of this algorithm are presented in section 3.5.3. Additional modeling considerations are considered in section 3.5.4.

3.5.1 Calculation Procedures

In this process, the device was an N-terminal module with a system-level model represented by a conduction matrix $G_A^i \in \Re^{NxN}$. The system-level analysis generated an effort response $E_S^i \in \Re^N$ and a flow response $F_S^i \in \Re^N$ for each external device in the system. A correction flow $F_C^i \in \Re^N$ was applied to each model for correcting the error between the different model representations. The total system-level response was defined by

$$G_A^i \cdot E_S^i + F_C^i = F_A^i + F_C^i = F_S^i \qquad \text{Equation 3-28}$$

$$G_M^i \cdot E_S^i = -F_M^i \qquad \text{Equation 3-29}$$

The device-level simulator sequentially applied the system-level effort response and generated a flow response $F_M^i \in \Re^N$ from an equivalent conduction matrix $G_M^i \in \Re^{NxN}$.

This procedure used the nodal representation in Figure 3-10 where a sensitivity matrix was available for the system-level, internal device, and external devices to construct a Norton equivalent circuit for nodal analysis. This information was used to find the next system-level effort E_S^{i+1} and flow F_S^{i+1}. A second calculation step adjusted the solution to eliminate the contributions from the internal device model. The new system-level flow and effort values from section 3.4 were defined as

$$E_S^{i+1} = E_S^i - \left(G_{EQ}\Big|_S^i + G_{EQ}\Big|_M^i \right)^{-1} \left(F_M^i + F_S^i\right) \qquad \text{Equation 3-30}$$

$$F_S^{i+1} = F_S^i + G_{EQ}\Big|_S^i \left(E_S^{i+1} - E_S^i\right) \qquad \text{Equation 3-31}$$

The new system-level effort and flow values were used to derive a new correction flow

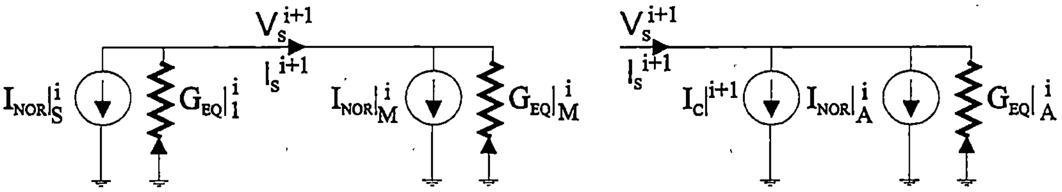


Figure 3-10. Nodal representation for the flow correction approach

where

$$\begin{aligned} F_C^{i+1} &= F_S^{i+1} - F_{NOR}\big|_A^i + G_{EQ}\big|_A^i \cdot F_S^{i+1} \\ &= F_S^{i+1} + F_A^i + G_{EQ}\big|_A^i \cdot \left(E_S^{i+1} - E_S^i\right) \\ &= F_C^i + \left(F_S^{i+1} - F_S^i\right) + G_{EQ}\big|_A^i \cdot \left(E_S^{i+1} - E_S^i\right) \end{aligned} \qquad \text{Equation 3-32}$$

This equation was further expanded to derive a flow-based relationship for testing different constraints on the solution process where

$$\begin{aligned} F_C^{i+1} &= F_C^i + \left(F_S^{i+1} - F_S^i\right) + G_{EQ}\big|_A^i \cdot \left(E_S^{i+1} - E_S^i\right) \\ &= F_C^i + \left(G_{EQ}\big|_S^i + G_{EQ}\big|_A^i\right)\left(E_S^{i+1} - E_S^i\right) \\ &= F_C^i - \left(G_{EQ}\big|_S^i + G_{EQ}\big|_A^i\right)\left(G_{EQ}\big|_S^i + G_{EQ}\big|_M^i\right)^{-1}\left(F_S^i + F_M^i\right) \\ &= F_C^i - \left(\frac{\partial F_S^i}{\partial E_S^i} + \frac{\partial F_A^i}{\partial E_S^i}\right)\left(\frac{\partial F_S^i}{\partial E_S^i} + \frac{\partial F_M^i}{\partial E_S^i}\right)^{-1}\left(F_S^i + F_M^i\right) \end{aligned} \qquad \text{Equation 3-33}$$

The iteration process was defined as a scaling matrix $\beta^i \in \Re^{NxN}$ times the flow summation between the device-level and system-level, or

$$\begin{aligned} \beta^i &= \left(\frac{\partial F_S^i}{\partial E_S^i} + \frac{\partial F_A^i}{\partial E_S^i}\right)\left(\frac{\partial F_S^i}{\partial E_S^i} + \frac{\partial F_M^i}{\partial E_S^i}\right)^{-1} & \beta^0(t_n = 0) &= \textit{identity matrix} \\ F_C^{i+1} &= F_C^i - \beta^i\left(F_M^i + F_S^i\right) & F_C^0(t_n = 0) &= 0 \end{aligned} \qquad \text{Equation 3-34}$$

Several variations of the scaling matrix are considered in the convergence analysis to eliminate the complexity of calculating three sensitivity matrixes.

This procedure was initially a sequential process to get a good initialization solution, but parallel version was easily implemented. The parallel version followed the procedures from section 3.4 except that an initialization sequence was required to apply the initial effort values to the device-level simulator. All new corrector values were defined using Equation 3-32, where the simulator-coupling matrix solved the new flow and effort variable values.

The predictor interfaces did require additional sensitivity parameters and constraints on the sensitivity function matrix (Equation 3-23). Each predictor device created both system and device sensitivity parameters, since the internal model sensitivity had to be calculated. The system parameters interacted ONLY with other system variables, and device parameters only interacted with other parameters from the same device. These relationships were defined as follows:

$$\sum_j \left(\Delta E_j\right)\cdot \left.\frac{\partial F_i}{\partial E_j}\right|_{DEV} = \Delta F_i\big|_{DEV}$$

$$\sum_j \left(\Delta input\right)\cdot \left.\frac{\partial output}{\partial input}\right|_{SYSTEM} = \Delta output\big|_{SYSTEM} \qquad \text{Equation 3-35}$$

The device sensitivity parameters were ALWAYS the FLOW over EFFORT parameters, while the system variables were identified using the input causality rules.

3.5.2 Convergence Analysis

From a flow perspective, convergence between the system-level simulator and the mechanical simulators occurred if and only if the device flow response becomes equal to the negative value of the system-level flow response. The expansion of the correction flow matrix showed that

$$\begin{aligned} F_C^{i+1} &= F_C^i + \beta^i\left(F_M^i + F_S^i\right) \\ &= F_C^{i-1} + \beta^i\left(F_M^i + F_S^i\right) + \beta^{i-1}\left(F_M^{i-1} + F_S^{i-1}\right) = \sum_{j=0}^{j=i} \beta^j \cdot (F_M^j + F_S^j) \end{aligned} \qquad \text{Equation 3-36}$$

Since the scaling matrix β^i was assumed nonzero, the flow sum had to approach zero for the process to convergence. This relationship was considered for detecting divergence in the correction process and for switching between different scaling options.

The effort iteration process showed how the different conduction matrixes interact to cause divergence problems. The recursive relationship for the system-level effort vector was defined as

$$\begin{aligned} E_S^i &= \left(G_A^i\right)^{-1}\left(F_S^i - F_C^i\right) = \left(G_A^i\right)^{-1}\left(F_S^i - F_C^{i-1} + \beta^{i-1}\left(F_M^{i-1} + F_S^{i-1}\right)\right) \\ &= \left(G_A^i\right)^{-1}\left(F_S^i - \left(F_S^{i-1} - F_A^{i-1}\right) + \beta^{i-1}\left(-G_M^{i-1}E_S^{i-1} + F_S^{i-1}\right)\right) \\ &= \left(G_A^i\right)^{-1}F_S^i + \left(G_A^i\right)^{-1}\left(\beta^{i-1} - 1\right)F_S^{i-1} + \left(G_A^i\right)^{-1}\left(G_A^{i-1} - \beta^{i-1}G_M^{i-1}\right)E_S^{i-1} \end{aligned} \qquad \text{Equation 3-37}$$

After repeated substitutions, this iteration process was equivalent to

$$\begin{aligned} E_S^i &= \left(G_A^i\right)^{-1}F_S^i + \left(G_A^i\right)^{-1}\beta^{i-1}\left(1 - G_M^{i-1}\left(G_A^{i-1}\right)^{-1}\right)F_S^{i-1} \\ &+ \left(G_A^i\right)^{-1}\sum_{j=0}^{j=i-1}\left[\prod_{k=j}^{k=i-1}\left(1 - \beta^k G_M^k\left(G_A^k\right)^{-1}\right)\right]\cdot\beta^j\left(1 - G_M^j\left(G_A^j\right)^{-1}\right)F_S^j \end{aligned} \qquad \text{Equation 3-38}$$

The necessary condition for convergence for effort was that all elements in the product term approached zero as the number of iterations increased, or

$$\prod_{k=j}^{k=i-1}\left(1 - \beta^k G_M^k\left(G_A^k\right)^{-1}\right) \to 0 \quad as\ i \to \infty \qquad \text{Equation 3-39}$$

This relationship defined the rate of convergence for the process. The convergence of this iteration process was examined based on three different scaling matrix definitions: an identity matrix, a strictly diagonal matrix, and the full definition.

3.5.2.1 Identity Matrix

As the simplest procedure, an identity-scaling matrix ($\beta' = 1$) eliminated all sensitivity calculations. For this scaling matrix to work, all models had to be relatively accurate, so corrections were done under small-scale conditions. Since the accuracy of the system-level device model was not absolutely specified, divergence became a possibility as the modeling errors increased. To examine this iteration process, the system-level flow was assumed constant while the conduction matrixes were also assumed constant and diagonal. With the diagonal matrix assumption, the non-diagonal elements in the matrix expansion process were ignored and the matrix was solved per diagonal element. A single effort value (e_S^i) with a flow (f_S) was defined by

$$\begin{aligned} e_S^i &= \frac{1}{g_A}\left[e_S + \sum_{j=0}^{j=i-1}\left[\prod_{k=j}^{k=i-1}\left(1-\frac{g_M}{g_A}\right)\right]f_S\right] = \frac{f_S}{g_A}\left[1+\sum_{j=0}^{j=i-1}\left(1-\frac{g_M}{g_A}\right)^{i-j}\right] \\ &= \frac{f_S}{g_A}\left[1+\left(1-\frac{g_M}{g_A}\right)^{i}\sum_{j=0}^{j=i-1}\left(1-\frac{g_M}{g_A}\right)^{-j}\right] = \frac{f_S}{g_M}\left[1-\left(1-\frac{g_M}{g_A}\right)^{i}\right] \end{aligned} \qquad \text{Equation 3-40}$$

The convergence requirement was that

$$0 < \frac{g_M}{g_A} < 2 \qquad g_A > 0.5 \cdot g_M \qquad \text{Equation 3-41}$$

Under the given conditions, the system-level effort and the mechanical flow converged to

$$e_S^\infty = \frac{f_S}{g_M} \qquad f_M^\infty = -g_M e_S^\infty = -f_S \qquad \text{Equation 3-42}$$

This simple process did meet the current convergence requirements, but the rate of convergence was poor unless the models were almost identical. As an example, a minimum number of iterations to satisfy a relative tolerance (ε_r) were calculated based on the maximum scaling ratio where $\alpha_{MAX} = (g_M / g_A)_{MAX}$. The minimum number of iterations n_{MIN} for relative convergence was approximately

$$(1-\alpha_{MAX})^{n_{MIN}} < \varepsilon_r$$

$$i > n_{MIN} = \begin{cases} \dfrac{\ln \varepsilon_r}{\ln|1-\alpha_{MAX}|} & 0 < \alpha_{MAX} < 2; \alpha_{MAX} \neq 1 \\ 1 & \alpha_{MAX} = 1 \\ \infty & otherwise \end{cases} \qquad \text{Equation 3-43}$$

Nonlinear device behaviors and non-diagonal matrix coupling effects were certain to increase the number of iterations for convergence. Consequently, an identity-scaling matrix was very impractical as the model accuracy decreased.

3.5.2.2 Diagonal Scaling matrix

Like the identity matrix, the diagonal scaling matrix definition ignored all coupling effects across the device's terminals. A strictly diagonal dominance assumption [83, pp.91] was made with the purpose of avoiding sensitivity matrix calculation, so all cross terminal component contributions were lumped together in the diagonal elements. A secant approximation provided the partial derivative relationship where

$$\frac{\partial F^i}{\partial E^i} \approx \frac{F^i - F^{i-1}}{E^i - E^{i-1}} = \frac{\Delta F^i}{\Delta E^i} \qquad \text{Equation 3-44}$$

If all effort variations for the nodes were identical, then the effort variations could be eliminated if the effort differences were not zero. Under this condition, the correction flow became a function of only the flow variations, where

$$\beta^{i}_{j,j} = \begin{cases} \left(\dfrac{\Delta F^{i}_{A,j} + \Delta F^{i}_{S,j}}{\Delta F^{i}_{M,j} + \Delta F^{i}_{S,j}} \right) & \left| \Delta E^{i} \right| > \varepsilon \\ 1 & otherwise \end{cases}$$ Equation 3-45

$$\beta^{i}_{j,k} = 0; j \neq k$$

Using the lumped linearization step, the consideration was that sensitivity information became a relative approximation ($G_R\big|_X^i$) times a diagonal scaling matrix (α_X^i) where

$$G_{EQ}\big|_X^i = \alpha_X^i \cdot G_R\big|_X^i$$ Equation 3-46

If the relative approximation was assumed equal in all situations and the scaling matrix was calculated in the same format for each representation, then the relative approximations cancelled and the scaling matrixes did provide sufficient information in certain cases to emulate the full sensitivity calculations. The convergence rate from this approach converged faster than the identity-scaling matrix. If the conduction matrixes combined to create a strictly diagonal result, then this process converged absolutely. Otherwise, divergence was still possible since the contributions from the cross-coupling effects accumulated to exceed the convergence conditions. Of course, the chances of divergence increased with an increase in the number of device terminals. However, this process had not calculated any gradients outside of the normal iteration process.

3.5.2.3 Full Definition

The full definition of the flow correction method had to calculate three sensitivity matrixes although the system-level simulator generated two sensitivity matrixes simultaneously. The positive feature of the full definition was that one model was completely adjusted in terms of another model representation to eliminate any errors between the two representations. Effectively, this approach negated the effects of the internal behavioral model. In a sequential format, this basic sequential process had low performance characteristics because the full sensitivity matrixes were still required. In a parallel format, the full procedure should have equivalent or better performance compared to the standard process IF THE SENSITIVITY MATRIX FOR THE INTERNAL BEHAVIORAL WAS ACCURATELY CALCULATED. Calculating the sensitivity parameters of the internal behavioral model was difficult in certain situations.

3.5.3 Algorithm Modifications

During the testing of the flow correction process, a significant problem was encountered where a large flow correction value caused the simulator to reduce its timestep. If a correction value was too large, the LTE calculation in the simulator had to compensate for the error by reducing the time step. Part of the problem was that the correction process used multiplication to find the flow correction values for the PREDICTOR interface, so any error in the sensitivity calculations caused the flow correction values to diverge. In certain cases, the large flow correction values forced the internal model into an unnatural state. The problem was the most severe at an inflection or peak point in the analysis of the internal model.

Because of the timestep reduction, the increase in the number of analysis points made this process very iteration inefficient compared to the convention interfaces. To make this interface more feasible, the problem had to be eliminated, and a minimization rule was used to limit divergence of the flow correction values. For this rule, two different correction variables ($F_{C,A}^{i+1}$ and $F_{C,B}^{i+1}$) were calculated from Equation 3-32 where

$$F_{C,A}^{i+1} = F_C^i + \Delta F_{CALC}^i + G_{present} \cdot \Delta E_{CALC}^i \qquad \text{Equation 3-47}$$

$$F_{C,A}^{i+1} = F_C^i + \Delta F_{CALC}^i + G_{previous} \cdot \Delta E_{CALC}^i \qquad \text{Equation 3-48}$$

The normal calculation for the correction process is Equation 3-47 and the checking function is Equation 3-48, where the normal function used the present sensitivity matrix and the other function used the previous sensitivity matrix values. The assumption for this rule was that only one set of sensitivity parameters was invalid at any point. The limiting feature compared these two calculations and restricted the final correction value (F_C^{i+1}) as follows:

$$\begin{array}{l} if \quad \left|F_{C,A}^{i+1}\right| > \alpha \cdot \left|F_{C,B}^{i+1}\right| \quad then \quad F_C^{i+1} = F_{C,B}^{i+1} \\ else \quad F_C^{i+1} = F_{C,A}^{i+1} \end{array} \qquad \text{Equation 3-49}$$

The ratio α was the user-definable parameter CORRSENSRATIO under the DEVDEF command (see Appendix-A). The optimum value was found quickly to be between two and ten, but very little experimentation was done beyond the default value of five.

This limiting procedure did identify the problem with the correction process as the proper calculation of the sensitivity parameters of the internal model. In the future, a more robust approach is required to eliminate the parameter divergence since certain examples still had a large number of transient points where the basic assumption was violated. Different sensitivity calculation methods did improve the iteration efficiency of certain examples.

3.5.4 Modeling Issues

The model definitions at the system-level required sufficient information about the mechanical components to define the first-order coupling mechanisms between the device terminals. Otherwise, the correction flows did not properly adjust the effort variables like the FE device model, and the process began correcting large-scale variations compared to small-scale variations. The behavioral model in this method should be at least as accurate as a device's structural representation like a RLC network (linear representation). More precise models offered faster convergence rates for the analysis and less iteration with the external simulator. The consideration was that the external models were always more complex and accurate than the internal model. Otherwise, the simulation process did not require the coupled simulator. Finally, this method required that the behavioral model did not need modified nodal analysis (MNA) since the process was based completely on nodal analysis. The primary concern was inductive elements with no series resistance where different effort sources could be shorted.

3.6 Summary

This Chapter has outlined three different coupling methods: a non-conventional, a conventional, and a flow correction. Each method attempted to use causality assignment to improve the coupling process. For the first two methods, a causality-detection scheme was considered where the interface model was self-configuring across all simulators to maximize iteration efficiency and to define an interface structure where "one fits all" for black box representations. The first method had several fundamental problems that make the process completely unstable under most conditions. The second method was based on conventional techniques and the mathematical foundation was consistent for all elements. Using conventional interfaces, causality assignment was used to switch between different calculation processes to improve convergence and iteration efficiency.

The flow correction method implemented an overlapping approach where behavioral models completed all feedback paths in the analysis. All simulators had a defined role as system-level or device-level simulator, where the system-level simulator(s) controlled the device-level simulators in a sequential format. With an initialization sequence, the correction approach was made into a parallel process and was solved with the standard interfaces. In terms of performance, the flow correction method potentially eliminated the sensitivity matrix calculations required between simulators if the ratio of the real and behavioral conduction matrixes were sufficiently diagonal. The advantage of the flow correction method was in the hybrid approach where different external modeling representations (correctors) were or were not applied to the default behavioral model.

Chapter 4. Backplane Implementation

In this chapter, the implementation of the backplane is described. The backplane routines provided a common framework (as middle ware) to synchronize and to convert information between the simulators as shown in Figure 4-1. This implementation made all simulators into simulation engines where the backplane synchronized the engines. In this process, the backplane became the simulator while the simulators became model evaluators. All backplane operations were classified into the four functional categories: calculations, communications, control, and database. Sections 4.1 to 4.4 describe the implementation aspects of these categories. The control parameters were the major focus point to implement the dynamic modeling process and to optimize the coupling efficiency between simulators. The simulator selection process is described in section 4.5.

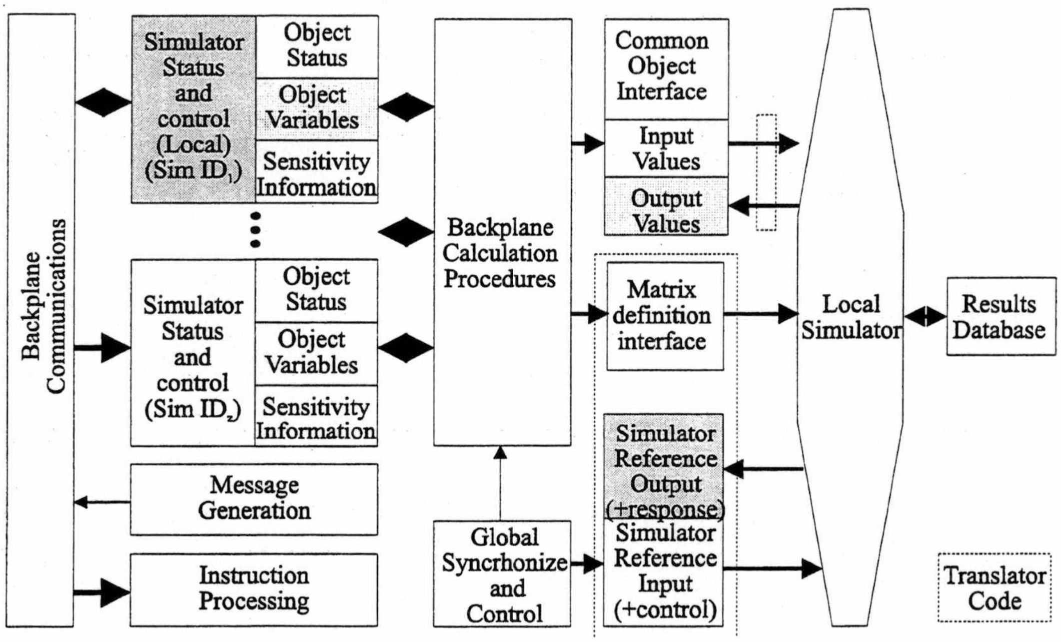


Figure 4-1. Interface between the backplane and simulator

The command set for the backplane operation was developed to maximize flexibility and expandability. All data within the backplane was sent in ASCII form to be human readable for debugging and script writing purposes. A short synopsis of the different database and command keywords are presented in Table 4-1. The complete definitions and parameter variables for the backplane are presenting in Appendix-A, which also outlines the modifications required to incorporate the backplane into a simulator.

4.1 Communications

The communication mechanism was considered a physical problem that was not a major issue in this work. Of coarse, the fastest communication mechanism was preferable to the slowest implementation. A modular development of the backplane architecture allowed

Table 4-1. Keywords for the simulation backplane

Keyword Codes	Description
DEVDEF	Define a device within a simulator.
GLOBPRM	Define global parameters common to all backplane elements.
GRADDATA	Define object gradient data from a simulator.
ID	Define the simulator ID during the initialization process.
INCLUDE	Include a file as a subcommand file.
MESSAGE	Messages (warnings, errors, etc...) sent to the backplane controller.
MODELDEF	Define a multi-facetted model for the dynamic switching process.
OBJDATA	Define object variable data of a simulator.
OBJPRM	Interface controls and configurations for an object.
REQSTAT	Request the status of a simulator or its internal objects, parameters, etc.
SIMCMD	Send a command directly to a simulator.
SIMPRM	Define simulator control instructions and parameter definitions.
START	All simulators are to start processing data based on their MODE parameter.
STOP	All simulators are to stop and await further instructions.
TOLERANCE	Define the convergence tolerance for variables of root objects.
TRIGDEF	Define the trigger conditions for switching between models.

the substitution of different transfer mechanisms like SOCKETS [96], PVM [100], or even file based transfers. The backplane communications mechanism was chosen as SOCKETS, but any communications mechanism could be used. The architecture of the communication process was more important for optimizing performance by eliminating sequential delays.

The communications architectures considered in this work were peer-to-peer and client-server with a central controller module (BKPLCON). The controller was present to make decisions for the dynamic switching process and to be a communications server. Simulators always communicate with a controller module and NEVER directly with another simulator, so only the controller had to handle errors and the addition of new simulators during the analysis. In a peer-to-peer configuration, each simulator implemented the backplane calculation procedures independently to reduce sequential operations. In the client-server mode, the server received all information, performed all system-level calculations, and defined the interface values back to each simulator. Sensitivity calculations were performed by the local simulator, but other calculations were the responsibility of the server.

Both implementations had approximately the same communications overheads. The client-server model had additional sequential processing steps because of the equivalent matrix generation required by certain interfaces. However, the client-server model had a district advantage in a single processor environment because of redundancy in the calculations. In a multiprocessor environment, the major performance issue was the calculation of the equivalent matrix with the peer-to-peer architecture having the advantage. This implementation used the peer-to-peer architecture model because of the potential elimination of sequential processing steps.

4.2 Database Structure

The primary database definition for the backplane was BKPL_DEF. This structure contained all simulator information as shown in Figure 4-2. Each simulator had to create this variable internally for interaction with the backplane as described in Appendix-A. The parameter definitions in the different structures were the control mechanisms for all aspects of the backplane. For this reason, the common parameter structure within the backplane was easily expanded with a variety of IO formats.

The backplane structure BKPL_INTF under the object definition of a simulator was the master node translator where both the backplane and simulator read and wrote

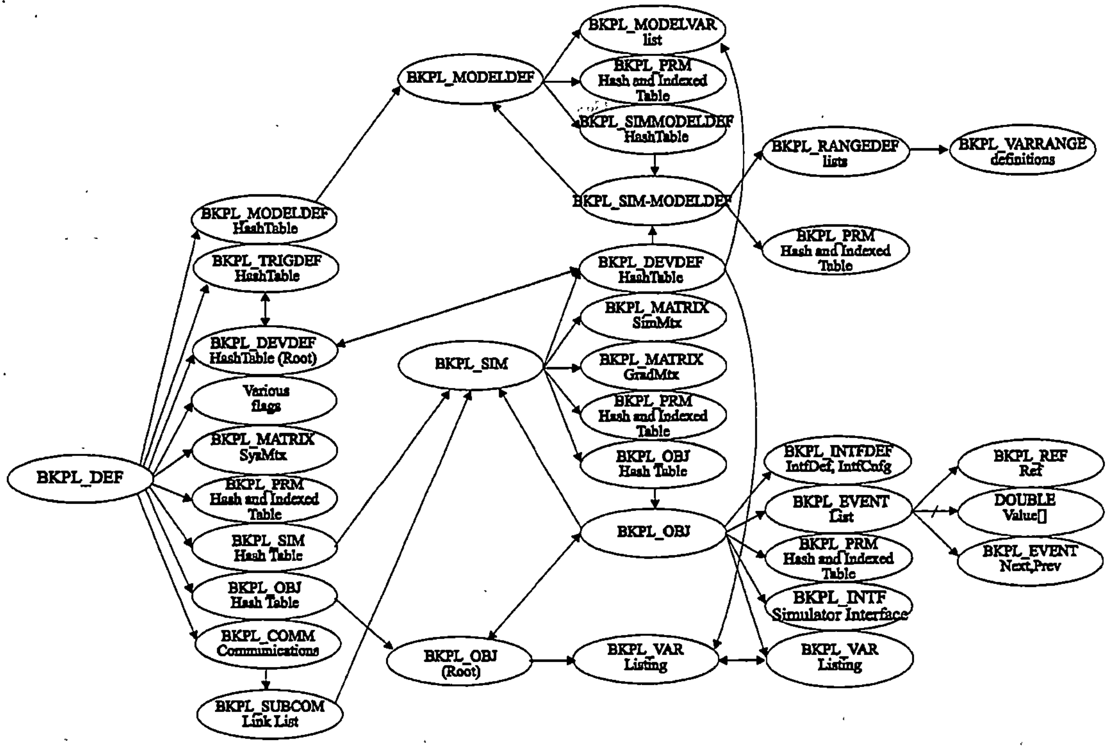


Figure 4-2. Overview of the backplane database structure

information. All data access was done through this interface structure (See Appendix-B). Each interface corresponded to a backplane object with an interface definition and configuration. The interface definition was the primitive interface type defined by the simulator. The interface configuration defined how a solution was applied to the interface.

In this coupling process, only shared data and monitor points from a simulator were transmitted over the backplane while all other solution information was stored and maintained by the local simulator. The backplane did not attempt to generate a common database since data retrieval was a function performed by the local simulator or other data translators as part of the design and simulation framework [57]. However, this data retrieval procedure could cause a problem in the dynamic modeling approach where data from a previous session could be overwritten if a simulator restart occurred. For this reason, deactivating the simulator into the OFF mode prevented the overwriting of data compared to a simulator shutdown with a restart.

4.3 Control Structure

The two control aspects of the backplane were the backplane controller (control module) and the individual simulator control. The control module interacted with the user interface and defined the operations that the other simulators performed. At the beginning of an analysis, the simulators performed the normal initialization process where a backplane initialization sequence found a unique name or ID for the simulator and created the backplane structure within the simulator. The backplane process in a simulator is shown in Figure 4-3. Once all simulators were operational, the controller module defined the objects,

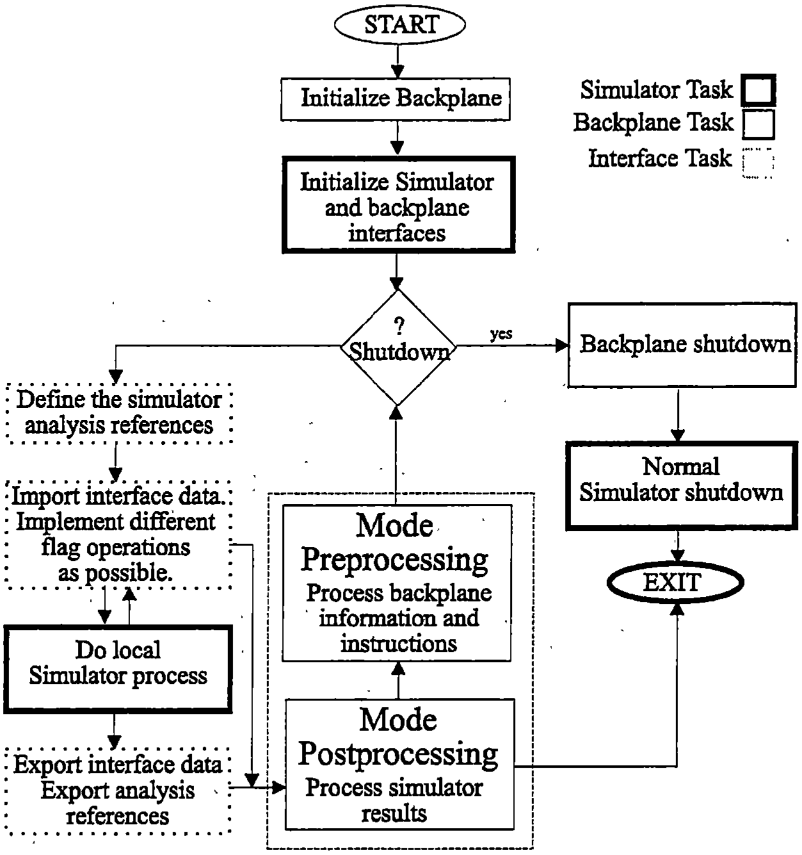

START
Initialize Backplane
Simulator Task
Backplane Task
Interface Task
Initialize Simulator and backplane interfaces
? Shutdown
yes
Backplane shutdown
Define the simulator analysis references
Import interface data. Implement different flag operations as possible.
Normal Simulator shutdown
Mode Preprocessing
Process backplane information and instructions
EXIT
Do local Simulator process
Mode Postprocessing
Process simulator results
Export interface data Export analysis references

Figure 4-3. Simulator control overview flowchart

the initial modes, tolerances, interface configurations, etc for the simulators and sub-controllers. The major aspects of the control structure were the simulator mode, operational status control, and the triggering situations.

4.3.1 Simulator Modes

The most important control parameter for a simulator was MODE under SIMPRM, which defined the processing procedure for the simulator. The MODE parameter for a simulator had eleven different states of operation as shown in Table 4-2. A simulator's mode was controlled via the START and STOP commands. This section describes the

Table 4-2. Backplane modes for a simulator

Classification	Mode Names
Non-calculation, non-interactive modes	IDLE or OFF ITERROGATE and OVERRIDE SHUTDOWN
Non-iterative modes	LOAD and SAVE INDEPENDENT SYNCHRONIZE
Iterative modes	ITERATE and SENSCALC TRACK UPTODATE

different operational modes.

4.3.1.1 Non-calculation, Non-iterative Modes

Most modes in this category were a variation of IDLE or OFF. An IDLE mode simulator received information from other backplane elements and responded to new instructions from the control module. After the successful completion of an operation or a STOP command, all backplane elements were forced into the IDLE mode except for the INTERROGATE mode. The OFF mode indicated that a simulator was to remain inactive with no contributions to the solution matrix although the IDLE mode had the same definition. The SYNCHRONIZE mode did make a distinction between the two modes.

The SHUTDOWN mode began the sequence for a simulator to perform an orderly shutdown of its internal databases prior to the backplane forcing a shutdown. The simulator had one opportunity to perform a shutdown after the shutdown flag was set to the simulator. Otherwise, the next occurrence of the main backplane procedure forced the simulator to exit or halt.

The INTERROGATE mode broadcasted the local simulator's object variables, the object configurations, and simulator parameters to the central controller for diagnosis and configuration. This mode provided all backplane elements with knowledge of the objects and variables involved in the simulator including any previous responses or event lists. This information was used for synchronization and simulation initialization. A backplane element entered the OVERRIDE mode when the INTERROGATE mode was completed. The OVERRIDE mode was another variation of the IDLE mode where the READONLY parameters in the backplane could be overridden.

4.3.1.2 Non-iterative Modes

The SYNCHRONIZE mode was a non-iterative version of the ITERATE mode where a simulator applied variable information to synchronize with the operation of the other simulators. This operation was a roll-forward for simulators that were being spawned or restarted from the OFF mode. Additional synchronization modes were the SAVE and LOAD operations for capturing and returning to a previous calculation point as discussed in section 2.5 for rollback or quick initialization. However, these functions were simulator operations and not backplane functions, so the backplane only initiated the operations. For manual synchronization processes, the OFF mode distinguished simulators that had previously been active (now in an IDLE model) from simulators that were OFF in the previous analysis operation. The backplane automatically synchronized simulators that changed from an OFF to an ITERATE mode during an analyses.

The INDEPENDENT mode detached the simulator from the backplane. By detaching the simulator from the backplane, the simulator returned to an independent operational mode where most backplane overheads were removed. Once a simulator was in the INDEPENDENT mode, the simulator was not able to reconnect to the backplane. The simulator did interact with the backplane for defining interface values that were potentially stored in a data event list.

4.3.1.3 Iterative Modes

The ITERATE mode was the primary iteration process where all simulators were iteration locked and data synchronized. The other iterative modes followed the iteration locking rules of the ITERATE mode simulators. The three iteration levels for the ITERATE mode simulators were:

1. Iteration done.
2. Reference done.
3. Simulator-stopped.

The lowest priority was the iteration done setting, where the simulator had completed the present iteration for the given reference point without convergence. The reference done level indicated that the simulator had found a converged solution for the given reference point. When all simulators reach the done level, the process continued to the next reference point on the next iteration. The simulator-stopped level indicated that the simulator had reached a stop or error condition. Simulators remained in a stop state until all simulators reached an equivalent state. The stopped level had the highest priority. Although the rules were common, the TRACK and UPTODATE modes applied the rules at different levels.

The SENSCALC mode was a temporary mode in the ITERATE mode for generating the sensitivity parameter matrix of a simulator required by the calculation process.

The TRACK mode was a variation of the ITERATE mode where simulators were locked at the reference done level, so the TRACK mode simulators were not involved in the standard calculation procedure. The TRACK mode allowed a simulator to track the final solution of other simulators by lagging the operation of the ITERATE mode simulators. The simulators in the ITERATE mode had to wait for TRACK mode simulators to finish their analysis of the previous reference point. Specifically, the TRACK mode was a concurrent verification tool of other simulators used with an accuracy trigger.

The UPTODATE mode was a combination of the ITERATE mode and SYNCHRONIZE mode. The purpose of this mode was to eliminate part of the sequential delays for a simulator in an OFF mode to be resynchronized with simulators in an ITERATE mode. The procedure waited for the simulators in the ITERATE mode to exceed the retained event limit (number of reference points that an object can maintain during an analysis). When the minimum reference point in all event lists were beyond the present reference point of the UPTODATE mode simulator, the simulator synchronized with the other simulators to the last valid solution point. An UPTODATE simulator only attempted to remain up-to-date with the analysis of the ITERATE simulators. A simulator in the UPTODATE mode was not locked with the simulators in the ITERATE mode

4.3.2 Simulator Status Control

The three synchronization parameters in the backplane were REFPREV (stable or previous iteration point), REF (present iteration point), and REFDELTA (next iteration delta value for the REF point) as shown in Figure 4-4. These reference parameters corresponded to common variables in the analysis such as time, frequency, etc. This work focused on transient analyses where time was the primary variable. The reference structure for all synchronization data within the backplane was expressed as a hexadecimal number of discrete intervals and not as a floating-point number [92,pp. 21]. Without using hexadecimal representations, the different backplane elements lost synchronization due to timestamp mismatches caused by error accumulation and rounding of imprecise floating-point numbers.

Several flags were defined for the simulator from the backplane to control various aspects of the analysis as described in Table 4-3. The one essential flag was the shutdown indicator that allowed the simulator to shutdown in a normal fashion and to cleanly close all related databases. All other flags were optional and depended on the capabilities of the

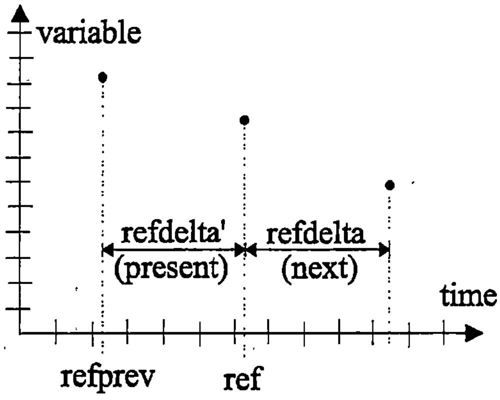


Figure 4-4. Major variable reference (time) exchange between a simulator and backplane

Table 4-3. Flags generated from the backplane to the simulator.

Flag (**)	Description
SHUTDOWN	Indictor to shutdown the simulator in an orderly fashion.
OPERATION	Indictor that a backplane interface was in the analysis.
SOLV_SAVE	Indictor to save the present iteration data as the solution. Without this flag, the simulator stored interim iteration data.
LOAD_FILE	Indictor to load external file that contains simulator solution and state information (BACKUP/restore function).
LOAD_STATE	Indictor to load a simulator's backup solution and states from memory (BACKUP/restore function).
NEW_TIME	Indictor that the REF variable has changed.
ROLLFORWARD	Indictor to a simulator that the backplane will be synchronizing the simulator to new data in the future. If possible, the simulation should move to the new reference point, and restarted the analysis assuming initialization conditions applied.
SAVE_FILE	Indictor to save a simulator's solution and state information into an external file (BACKUP/save function).
SAVE_STATE	Indictor to save a simulator's solution and state information into memory (BACKUP/save function).
SENS_CALC	Indictor that sensitivity calculations are being performed. The simulator's local model evaluation process can be skipped if possible.
SENS_SSAVE	Indictor of the beginning of sensitivity calculations. The simulator should save the present solution values to restore the values at the end of the sensitivity process.
SENS_SLOAD	Indictor of the ending of sensitivity calculations. The simulator can load the stored solution values, so an extra iteration is not required to return the solution back to the previous value.

** The prefix BKPL_ is applied to all flags to avoid naming conflicts with the simulator.

simulator, but the efficiency of the coupling process could be improved by using these flags. The sensitivity control flags had the greatest potential to improve the performance of the overall analysis by eliminating calculation steps in a simulator (See section 4.4.5).

4.3.3 Model or Simulator Switch Conditions

This simulation backplane had five different triggering conditions for model switching as defined in Table 4-4. All the different trigger conditions can be used

Table 4-4. The different trigger conditions available in the backplane

Trigger	Description
Accuracy	This trigger uses simulators in a TRACK mode to maintain a specified accuracy compared with the reference device model. The trigger is re-activated after occurring.
Immediate	Immediately switch between the active device definition and a new device model. The trigger is de-activated after occurring.
Range	This trigger uses the ranging information in the model definitions to optimize performance, variables ranges, and accuracy. The trigger is re-activated after occurring.
Reference	At a specific reference point, switch to a certain device model. The trigger is de-activated after occurring.
Variable	This trigger uses variable conditions to change between different model representations. The trigger is re-activated after occurring.

concurrently, although the accuracy and range triggers were designed to be autonomous. Most trigger conditions waited for the convergence of the present iteration sequence before implementing the model switching process to eliminate triggering loops. The specific rules for model selection using the different trigger condition are outlined in Appendix-A. A general description of the dynamic modeling process was defined in section 2.5.

The dynamic modeling process within the backplane had several levels of decision making. The trigger conditions defined the device status, the device status defined the object status, and the object status was used to define the simulation mode. Once the simulator mode was defined, the object status was redefined based on the simulator status. The rules for the object and simulator operation were very simple. If any object in a simulator was active, the simulator had to be in an ITERATE mode. Otherwise, the backplane assigned the simulator mode based on the DEACTIVE_MODE parameter. For a simulator in the

ITERATE mode, the object status was either ON or TRACK. The TRACK mode required the object to track the operation of the active interfaces in a node. An object status was set to OFF only if the simulator was in an OFF mode. The other simulator modes required the simulator's objects to be in a TRACK mode.

4.4 Calculation Control Structure

The calculation procedures were outlined in Chapter 3, where the basic iteration process was defined by Equation 3-24 and Equation 3-25. The scheduling and calculation of the sensitivity information was most critical task in this coupling process, since all aspects of the calculation process depended on the sensitivity information. The exact sensitivity functions were not required to find a solution in the iteration process, but better approximations had faster convergence [12, 92: pp. 72-81]. Any improvement in convergence enhanced performance and iteration efficiency by eliminating calculations. Thus, a variety of scheduling and sensitivity calculation options was provided by the backplane, but full optimization of the backplane interaction with the simulator was left as a future development issue. The specifics of the matrix building, convergence criteria, sensitivity calculation controls, and basic object modes are defined in this section. The interface initialization procedures did impact the calculation process as discussed in Appendix-B.

4.4.1 Matrix Building

The matrix building process was a very straightforward procedure. The backplane rebuilds each matrix on a START command and rechecks all device, object, variable, and trigger lists. During the building process, four matrixes were constructed or updated: perturbation, sensitivity (GRADMTX), interconnection (SYSMTX), and equivalent (SIMMTX). The perturbation matrix was a simulator specific matrix that was used to generate a simulator's sensitivity matrix. The sensitivity matrix was exported to all simulators, and was used to generate the interconnection matrix. The interconnection matrix was used to calculate new solutions and to generate the equivalent sensitivity matrix for the local simulator.

All matrixes were fixed in terms of variables in the matrix although a simulator's local perturbation matrix had dynamic characteristics because of the input-causality detection process. Only the perturbation matrix used the inactive and active variable capabilities of the backplane matrix structure where the inactive variables were not used in the solver process and were zeroed. Delta information (acquired during the SENSCALC mode) was defined for both active and inactive parameters since causality was determined after the perturbation method was applied to the simulator. The causality detection process essentially determined which variables were active and inactive. Finally, the perturbation matrix was selectively built based on an interface's characteristics defined in the backplane. All matrixes were solved using a pivoting LU decomposition method [86].

4.4.2 Convergence Control

Because of the variety of interface structures and procedures, the backplane initially implemented all convergence rules based on the EFFORT and FLOW rules for all variables from each simulator. Specifically, the basic convergence rules were:

$$\begin{aligned}&|E_{MAX} - E_{MIN}| < \varepsilon(MAX\{|E_{MAX}|,|E_{MIN}|\})\\&\left|\sum F_i\right| < \varepsilon(F_{MAX})\end{aligned}\qquad \text{Equation 4-1}$$

The $\varepsilon()$ function was the error calculation function of the backplane defined in Equation 2-10. After several tests, the realization was made that the basic rules had a serious flaw. The calculated solution from Equation 3-24 converged, and thus, the variable values for the simulators reached a consistent solution. However, Equation 4-1 was not satisfied and the iteration process did not converge. This approach differed from the conventional procedure of testing the convergence of each variable of the calculation solution with scale value $(\alpha = 1)$ where

$$\alpha \cdot |x_{i+1} - x_i| < \varepsilon(x_i) \qquad \text{Equation 4-2}$$

To eliminate the flaw, the conventional convergence rule was incorporated into the convergence process using the same tolerances parameters with tighter scale value $(\alpha = 10)$. When all variables satisfied Equation 4-2 given a sufficient number of iterations in the present sequence, the backplane declared the solution valid and continued to the next iteration point. Certain interfaces did override the secondary convergence rule when the interface variables were not part of the calculated solution. Equation 4-2 was checked ONLY after the initial convergence of all simulators.

Another problem during the solution process was solution oscillations. A calculation sequence was converging if the following condition was satisfied:

$$\Delta_1 = \sum (x_{i+1} - x_i)^2 < \Delta_2 = \sum (x_i - x_{i-1})^2 \qquad \text{Equation 4-3}$$

When this condition was violated, a relaxation factor (α) of 0.5 was added to Equation 3-

24 to limit divergence, or

$$x_{i+1} = x_i - \alpha \left(\frac{\partial F}{\partial x} \right)^{-1} F \qquad \text{Equation 4-4}$$

After the divergence condition had been detected a certain number of times, the relaxation factor was always applied to the calculation process until the next iteration sequence began.

Using Equation 4-3 and 4-4, the iteration sequence could still have a slow rate of convergence. The slow convergence rate was identified by

$$(\Delta_1 < \Delta_2)\ and\ (\Delta_1 > 0.9 \cdot \Delta_2) \qquad \text{Equation 4-5}$$

On this condition, the final solution was weighted with the present and previous values to generate a new solution [102] where

$$x_{i+1}' = \sum_{j=0}^{j=N} \alpha_j \cdot x_{i-j+1} \quad N = 2, \alpha_0 = 0.5, \alpha_1 = 0.4, \alpha_2 = 0.1 \qquad \text{Equation 4-6}$$

The combination of the simulator sensitivity parameters and variable oscillations was one cause of the slow convergence problems.

4.4.3 Sensitivity Delta Definitions

The calculation choices for the partial differential equation in the iteration process were the Newton or secant based approaches although various methods can be used [92]. For a comparison, Newton's method had a quadratic convergence rate (m=2) where a secant's convergence rate was lower at (m=1.618) [94, pp. 26-32]. The difference between the two approximations was the size of the delta where the Newton method had a smaller delta than secant method. The secant method became equivalent to the Newton approximation as the variable delta approached a small value. The size of the delta was critical as the applied deltas could become too small and sensitivity information encountered rounding problems [94, pp. 155-157]. Large deltas had the problem of masking essential information. The sensitivity tolerance parameters were individually definable via the TOLERANCE command while the object parameter GRADCALC defined the delta's magnitude for the sensitivity process.

The backplane process applied a single delta to calculate the sensitivity information, so the direction of the delta had an impact on the coupling process. Minor variations in the delta direction often meant the difference between convergence and non-convergence. Several different options to define the direction of the applied delta were available via the object parameter GRADSIGN (See Appendix-A). Usually, the default option applied the delta to force the simulator's variable value toward the calculated value, since a common direction toward the "true" solution should maximize convergence. In addition, the sensitivity information affected the solution, which determined how a simulator adjusted its timestep. The performance of the coupling process was also improved by minimizing the number of time steps to reach the REFSTOP point.

4.4.4 Sensitivity Scheduling and Mode Definitions

The scheduling of the sensitivity calculation was controlled by the object parameter GRADTIME (See Appendix-A). This parameter had a direct relationship on the number of calculations between the simulator and backplane. Prior to the first converged solution, sensitivity information was calculated per-iteration regardless of the GRADTIME definition to implement Newton's method to the fullest degree possible. Because of this diversity in the sensitivity scheduling, the dynamic configuration process for an interface from section 3.1 was performed at the reference-done level in the iteration sequence and on the detection of a new input variable causality.

The GRADMODE parameter for each object defined how to calculate the sensitivity information. The three options were IDENTITY or NONE, DIAGONAL, and FULL. The IDENTITY option was intended for the correction process, and this option disabled the calculation process for non-correction modes. The DIAGONAL option implemented the sensitivity procedures based on decoupled or very-strong diagonal conditions. In the DIAGONAL mode, all sensitivity information was calculated under the assumption that all gain and coupling effects between the different nodes variables in a simulator were negligible and set to zero. This method specifically used the secant approach using the present and previous iteration values while the delta(s) was sufficiently large. No sensitivity calculation modes (SENSCALC) were required. The FULL option implemented the perturbation method where a unique delta was applied to a single interface and all related delta information was gathered. Then, the backplane solved for all sensitivity parameters simultaneously using Equation 3-23 after input causality was determined.

Another critical option for the sensitivity process was how the backplane handled a simulator rollback during a sensitivity calculation. The global parameter SENS_DREF controlled this decision. Initially, the rollback problems were not a significant issue, especially with small delta values and simple problems. As the problem complexity increased, any error in the sensitivity calculations resulted in sensitivity parameter errors and causality identification problems. Furthermore, any divergence in the iteration process was often sufficient to cause complete divergence depending on the sensitivity scheduling option.

4.4.5 Enhancement Sensitivity Calculation Options

Another important consideration for the coupling process was whether the model evaluation in a simulator was necessary, since most simulators generated and solved a linear representation. If this linear matrix was available, the sensitivity calculation was approximately equivalent to varying the source values and solving for the delta effects on other variables without additional model evaluations. For a linear system approximation in a simulator where $A \cdot x = b$, this process was defined by

$$\begin{aligned} A \cdot (x + \Delta x) &= (b + \Delta b) \\ A \cdot \Delta x &= \Delta b \\ \Delta x &= A^{-1} \cdot \Delta b \end{aligned} \qquad \text{Equation 4-7}$$

Since this simulator capability would enhance performance, the sensitivity calculation process from the backplane provided a flag to the simulator to indicate sensitivity calculations. The model evaluation process can be skipped in the simulator when possible. This flag also allowed the sensitivity process to be skipped if the local simulator had a better mechanism of calculating the sensitivity information for the backplane.

4.5 Simulator Selection

In this research, the basic premise from the first chapter was to use the discipline specific tools and simulators for MEMS analysis. With the proper backplane structure, the coupling of the simulators was a straightforward procedure that was simplified with the selection of an open simulator. A more important choice for the coupling process was using commercial or public domain software. Both types of software have advantages and disadvantages with tradeoffs that were very clear.

With commercial simulators, the designer had customer support and fully documented modeling capabilities with efficiently defined graphical user interfaces for visualization of the results. These tools were typically more polished and provided better feedback on errors. In addition, a wider range of algorithm selection was usually available to improve the convergence properties of an analysis. For the disadvantages, the commercial simulators had to be purchased and some yearly maintenance fee was required for the latest version. In some cases, the internal algorithms were accessible only through the basic simulator framework, so these simulators did not always have the flexibility needed to incorporate the backplane elements for sufficient control of the simulator.

Public domain simulators had greater flexibility for the designer to make changes since the source code was typically available. The initial costs of the product were much less, but the users eventually paid for product in time and effort because of the lack of documentation and "hidden" features in the code. The degree of testing for the code was usually less than commercial software. Furthermore, the user became responsible for maintenance and for debugging the software unless a large number of users were supporting the development of the basic program and any auxiliary software.

The choice between the two environments became a tradeoff between time and money. A sufficiently large budget provided the designers with a framework of tools and capabilities if the users knew all the tool requirements. Otherwise, resources were squandered on tools that do not truly address the present and future needs of the users. With the public domain software, the user experienced a limited view of the basic techniques and only wasted time and not as much money in the process. For graduate students, more time was available than money, so this work began the development with public domain software to have maximum control and flexibility. Specially, the electrical simulator was chosen as Spice3f4 [97] and the FE simulator was constructed using the deal-II FE toolkit [98]. In both selections, the optimum simulator choices were not made.

4.5.1 Electrical Simulator

The Spice program was a core simulator with only analog capabilities, and future expansion of this simulator's capabilities would require significant development. Spice was only chosen in this work because most public-domain electrical simulators use Spice has a comparison benchmark. The code was easily accessible. In retrospect, a simulator with an algorithm backplane should have been used. Still, the program sufficed for this research. One timestep implementation problem was found with the breakpoint definitions in Spice and a modeling discontinuity problem was discovered.

One major implementation issue occurred with the interfacing of the backplane to SPICE (version 3f4) because of the transient breakpoints created by SPICE. At a breakpoint, SPICE algorithm reduced the next timestep delta of the analysis. This interaction with other simulators caused problems especially when the reference point of the analysis was rolled back before the breakpoint of another simulator. Because of the breakpoints, the iteration process reduced timestep more than necessary unless the breakpoint algorithm was performed only when the SOLV_SAVE flag was active (ON) and GRAD_CALC was inactive (OFF). Without these conditions, the interaction between two simulators was very dependent on the sensitivity options with results ranging from divergence to long iteration sequences per reference point with a small timestep. After the conditions were applied to the breakpoint algorithm, the coupling process become consistent and less dependent on sensitivity options.

However, SPICE did have one implementation problem that was not a result of the backplane and simulator interaction. The problem was in the interaction of a voltage source with internal models, specifically transistor models. When the voltage was connected directly the transistor models, the current response had noticeable errors, which may have been a result of unrealistic or poorly defined transistor models. The problem was also observed with a Thevinin interface, but the error was more pronounced (orders of magnitude larger) with the voltage source. When tighter convergence parameters were required, the coupling process did not converge if a voltage source was used. The error was often insignificant compared to response, so the coupling process was not affected.

4.5.2 Finite Element Simulator

The search for FE simulators found FE toolkits and not simulators. Because of domain decomposition problem, a toolkit allowed greater flexibility to examine any structure or field-based problem. Consequently, this research had to construct a FE simulator, and the choice was made to use the deal-II toolkit [98] because of "well documented" C++ source code, but other choices were available [101]. For the implementation, the basics of the FE method with transient analysis were obtained from [99].

This FE simulator was developed to examine the coupling process and not for in-depth modeling. This research had to define the modeling and material properties, where the preferred approach was to have the built-in modeling. In this development, the FE simulator implemented the material characteristics as piecewise linear (PWL) functions. Still, the modeling process was made sufficiently complex to verify the coupling process and to examine the dynamic modeling procedure. The examples were not easily implemented in the electrical simulator.

4.6 Summary

This chapter has outlined the backplane structure and implementation over the four basic components of the backplane. The emphasis of this chapter was the control process, which determined the flexibility and capabilities of the backplane. The complete information on parameter setting is defined in Appendix-A while this section gives an overview of concepts considered in this backplane implementation.

Chapter 5. Results

This chapter presents a result summary on the coupling process for three different coupling categories: electrical-to-electrical coupling (section 5.2), electrical-to-mechanical coupling (section 5.3), and dynamic modeling process (section 5.4). An overview of the example types and sensitivity options is outlined in section 5.1. The goal of the two basic coupling tests was to find the optimum interface, where optimum interface minimized the total number of simulator iterations with the backplane and converged over the most examples. The dynamic modeling process demonstrated that the backplane allowed a broad range of analysis capabilities where designers have the flexibility during simulation to switch between different algorithms or design representations for accuracy or performance reasons. Most of these results are for functional verification of the backplane and a demonstration of the backplane's capabilities and failure modes.

5.1 Overview and Sensitivity

In the coupling tests, three types of examples were examined: no-feedback (effort gain=0), feedback with small gain (effort gain < 1000), and feedback with large gain (effort gain > 1000). The examples had varying degrees of causality with strong and weak coupling to examine the dynamic interface switching process and the variable-causality identification process in Appendix-C. The different feedback examples made certain that the backplane could solve problems that were poorly partitioned across multiple simulators with awkward interfacing characteristics. The electrical-to-electrical coupling tests implemented all of these tests, while the electrical-to-mechanical coupling implemented a small subset.

A common set of interfaces and sensitivity parameters were tested on each example in the coupling tests, so failures did occur. The failures were typically a result of four conditions: simulator non-convergence, interfaces violating causality assignment, deadlock limit (100), and backplane iteration limit (10000). The initialization procedure of the backplane was the most critical phase of the coupling process, since the backplane had to start the analysis based on the initial solutions of the simulator. In certain cases, the backplane initialization sequence contributed to the problems because of the calculation of erroneous sensitivity parameters due to poor initial solutions. The backplane initialization routines regarding sensitivity calculation procedures at initialization were deficient because the sensitivity delta definitions.

Without a doubt, the proper calculation of the sensitivity parameters had the major impact on the coupling process. Even minor variations in the sensitivity calculation procedures had a significant impact on the convergence and iteration efficiency of the backplane. The accuracy of the sensitivity parameters determined the number of iterations required to reach convergence, and the appropriate sensitivity parameters (local input causality) had to be identified. In most cases, the number of sensitivity calculations often dominated the total number of simulator iterations with the backplane. Optimizing the sensitivity calculation options was a critical task for achieving iteration efficiency because of the tradeoffs between calculating parameters and performing normal convergence iterations. Specifically, any minimization of the number of simulator iterations eliminated calculations, which improved the performance of the coupling process.

Otherwise, the backplane had no direct control over the individual aspects of a simulator's performance besides the flags defined in Table 4-3.

For the coupling process, the sensitivity scheduling option (GRADTIME) was the most critical definition. The number of iterations ($n_{iteration}$) was defined as

$$n_{iteration} = n_i + \alpha_i \cdot n_t \qquad \text{Equation 5-1}$$

The parameter n_i was the average number of iterations per timestep, which had a dependency on *all* sensitivity calculation options. The parameter n_t was the number of interfaces into a simulator. The parameter α_i was the scaling factor defined in Table 5-1. Of coarse, the interaction of an interface's coupling options with a simulator's internal algorithms did impact the performance and convergence of the simulation.

5.2 Electrical-to-Electrical (E2E) Coupling

This section focused on iteration efficiency (or iteration inefficiency) instead of performance because these examples were all relatively small problems with poor performance compared to a single simulator solution. Specifically, these examples could not

Table 5-1. Definition of the scaling factor for different sensitivity scheduling options.

Sensitivity Scheduling Option	**Scaling factor definition (α_i)**	
Iteration	$n_t - 1$	
Timestep	1	
Conditional	$\alpha_C \cdot n_i$	$\frac{1}{n_i} < \alpha_C < 1$
Error	$\alpha_E \cdot n_t$	$0 \le \alpha_E < 1$
Diagonal	0	
None, Identity	0	

overcome the overheads created by the communication and calculation processes of the backplane. In these tests, 31 examples (14 no-feedback, 9 low-gain feedback, and 8 high-gain feedback) over 76 interfaces and 18 different sensitivity and calculation parameters were tested (42408 analyses) where the average analysis time was 20 seconds. The iteration results for these examples are in Appendix-D.

As a summary, the average iteration results for the E2E coupling examples are presented in Figure 5-1, the convergence statistics are in Figure 5-2, and the average iterations per transient point are in Figure 5-3. A complete listing of the interface number to the particular interface combination is provided in Appendix-D. The average statistics and maximum variation for the iteration and convergence results is defined in Table 5-2, where variations values are for the different calculation and sensitivity options. In Figure 5-, the averages over the (n) examples that converged were based on relative statistics (X_j) of each example (j) with respect to the minimum value ($X_{j,\min}$), or

$$II = \frac{1}{n}\sum_{n}\left(\frac{X_j}{X_{j,\min}} - 1\right) \qquad \text{Equation 5-2}$$

This figure of merit defined the relative iteration inefficiency (II) of an interface compared to the optimum interface selection in each example. Using Equation 5-2, the goal was to eliminate timestep variations across the different examples. The interface combination with the smallest iteration inefficiency value had the best overall coupling performance. For direct comparisons between interface combination X and Y, an absolute iteration comparison (AIC) was required to convert the relative figure of merit to an absolute merit.

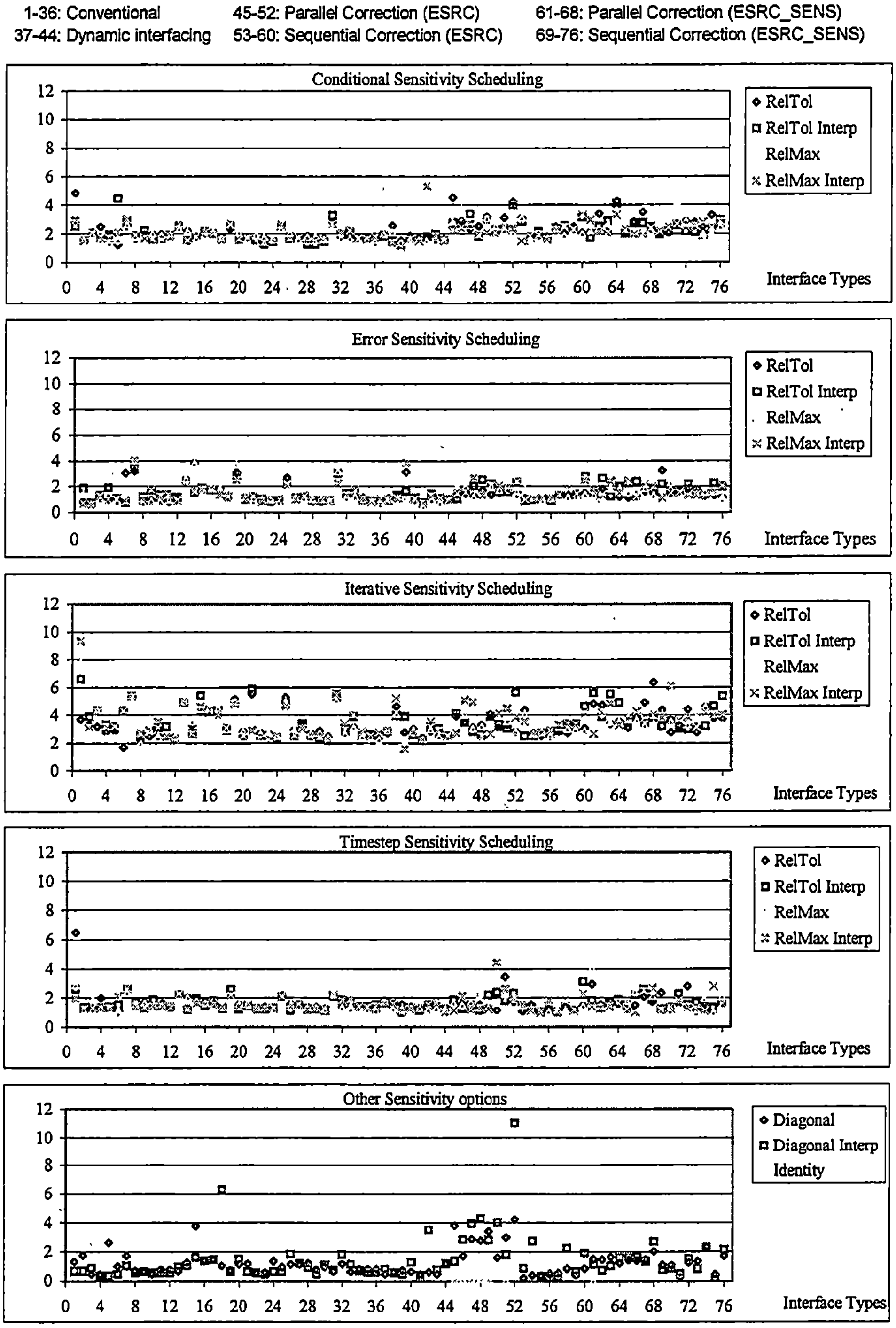


Figure 5-1. Iteration inefficiency statistics for all E2E coupling examples

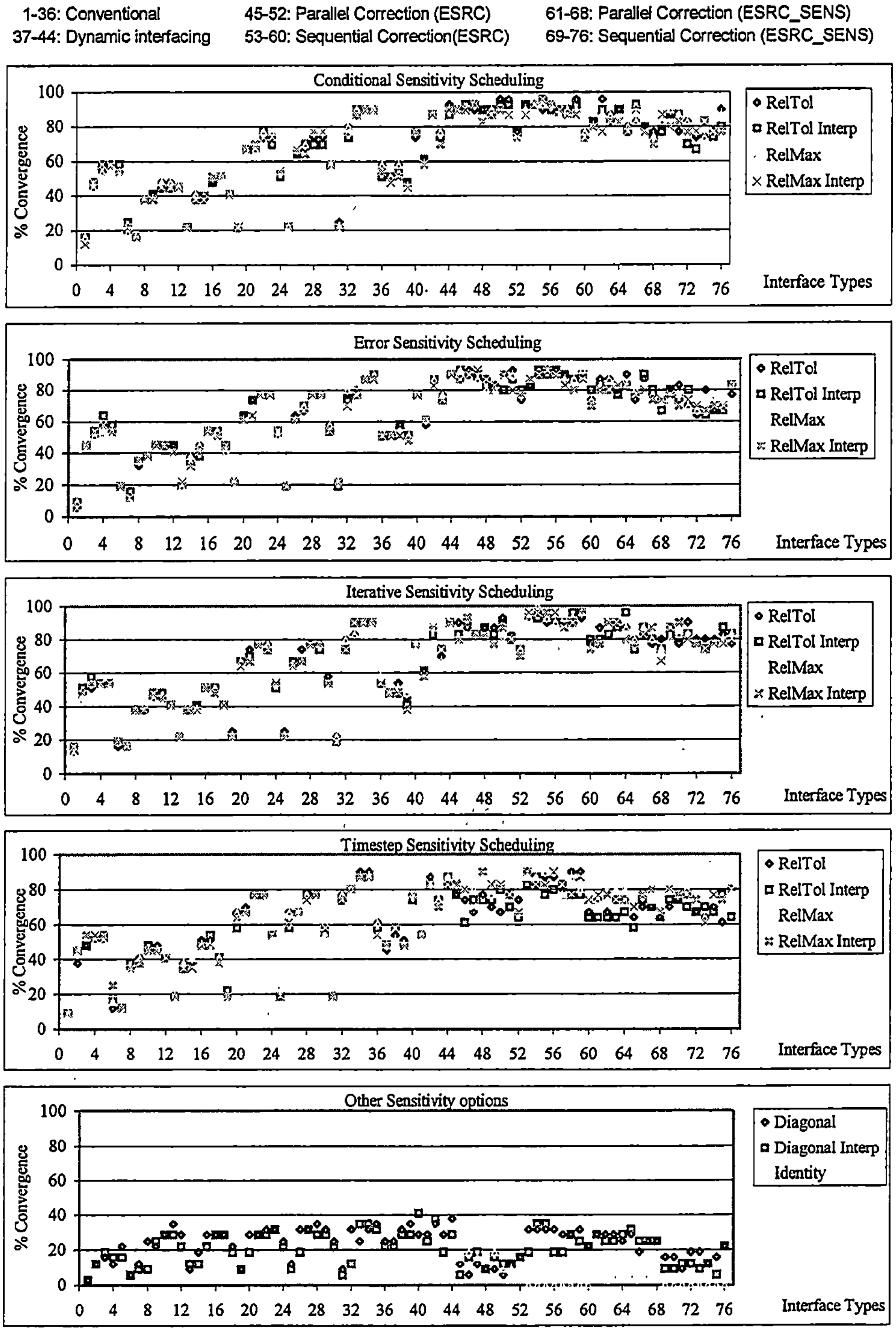


Figure 5-2. Convergence statistics for all E2E coupling examples

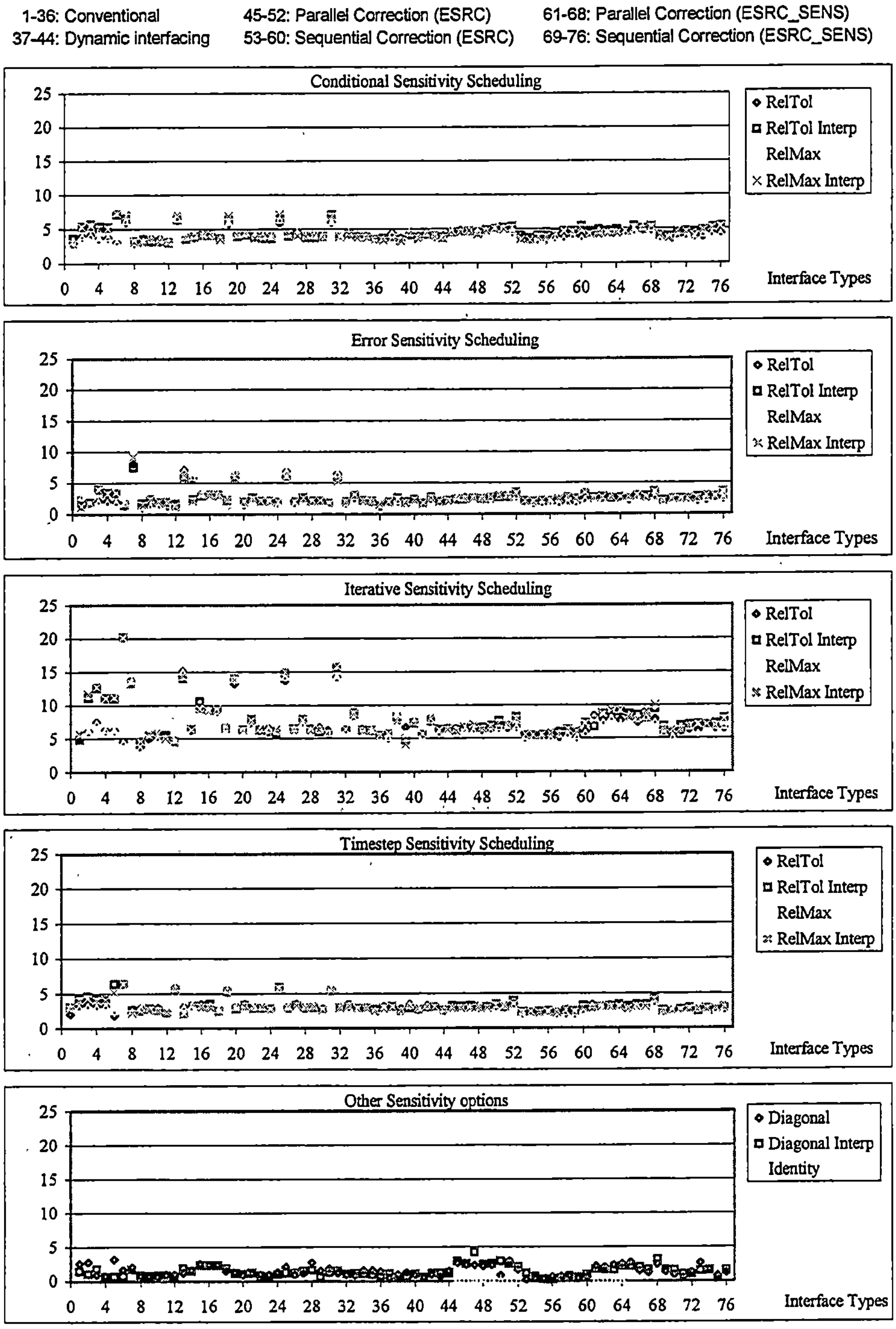


Figure 5-3. Average iterations per time point statistics for all E2E coupling examples

Table 5-2. E2E statistics based on the sensitivity options over all interfaces

Options	Flow Correction Interfaces		Conventional and Dynamic Interfaces	
	Average Convergence (%)	Average Iteration Inefficiency	Average Convergence (%)	Average Iteration Inefficiency
Conditional+Reltol	86.1	2.71	55.0	1.95
Conditional+Reltol+Intp	85.3	2.52	54.5	1.90
Conditional+ReltolMax	85.2	2.72	55.4	1.90
Conditional+ReltolMax+Intp	83.7	2.36	54.4	1.93
Diagonal	20.7	1.54	24.9	1.02
Diagonal+Intp	19.2	1.96	22.1	1.00
Error+RelTol	82.5	1.46	54.3	1.41
Error+Reltol+Intp	85.3	1.77	54.3	1.40
Error+RelTolMax	85.2	1.52	54.2	1.40
Error+ReltolMax+Intp	83.7	1.67	53.1	1.37
Iteration+RelTol	85.3	3.61	54.6	3.22
Iteration+Reltol+Intp	84.6	3.66	54.2	3.46
Iteration+RelTolMax	83.8	3.37	53.8	3.29
Iteration+ReltolMax+Intp	83.7	3.74	53.8	3.40
Timestep+RelTol	74.3	1.60	53.2	1.66
Timestep+Reltol+Intp	71.6	1.67	52.7	1.56
Timestep+RelTolMax	79.2	1.64	53.2	1.55
Timestep+ReltolMax+Intp	78.0	1.68	52.6	1.53

The absolute merit was defined as

$$AIC = 100 \cdot \frac{II_Y - II_X}{1 + II_X} \qquad \text{Equation 5-3}$$

The following sections discuss the results from the different interfacing tests.

The different calculation options caused a 16% convergence variation in the flow correction process compared to a 2.8% convergence variation with the conventional and dynamic interfaces as shown in Table 5-2. Since the convergence of the conventional and dynamic interfaces was consistent across the four major scheduling options, the combination of the backplane initialization procedure and the interface characteristics was considered the main factor in achieving convergence (especially initial convergence). The iteration inefficiency was most dependent on the sensitivity calculation options, although other secondary options did have a small (±10 percent) impact. Clearly, the CONDITIONAL, ERROR, ITERATION, and TIMESTEP option were valid techniques for scheduling sensitivity calculations. The DIAGONAL option could only be applied in decoupled situations.

The differences between the total number of iterations for different interfaces did not always depend on the backplane parameters. Instead, certain interface configurations often caused SPICE to reduce its timestep during a transient analysis, which increased the number of analysis points and the total iteration count. The result was usually a better iteration efficiency per timestep, but more steps were required to reach the final analysis point. Interface combinations were deemed less efficient if the overall number of iterations was larger.

5.2.1 Direct Configurations

As explained in Chapter 3, the direct configurations had the worst characteristics of any configuration or interface tested, and the simulation process typically diverged when direct configurations were used. The direct configurations could only be used when one simulator's interface characteristics dominated the other simulators contributions. In addition, the interface had to meet the gain requirements outlined in section 3.3.2 or have no coupling to other interfaces within the simulators. In the situations where these configurations converged, the diagonal-sensitivity calculation option with any interface combination was more iteration efficient than the direct configurations. For these reasons, no testing was performed on the direct configurations.

5.2.2 Conventional Configurations

The interfaces used by the simulator had an effect on the iteration inefficiency of the coupling process. An analysis by interface type over the CONDITIONAL, ERROR, ITERATION, and TIMESTEP GRADTIME sensitivity options is presented in Table 5-3, where any interface combination using the specific type was included in the average. Table 5-3 also shows the interface combinations with greater than 75% average convergence rate. The interface combination group with the highest convergence rate was the system configuration in partition 0 (driver circuit) with voltage, Thevinin, Norton, or Flexible

Table 5-3. Statistics for generic and specific E2E interfaces

Interface Definition	Convergence (%)	Iteration Inefficiency
Any Current	24.1	2.78
Any Voltage	54.3	1.85
Any Thevinin	58.4	1.96
Any Norton	62.8	1.78
Any Flexible	60.1	1.79
Any System	44.7	1.82
Norton-Norton	76.6	1.51
Norton-Flexible	75.8	1.46
Flexible-Norton	76.0	1.55
Flexible-Flexible	76.2	1.51
System-Voltage	76.4	1.86
System-Thevinin	83.6	2.01
System-Norton	88.7	1.56
System-Flexible	89.3	1.48

interfaces in partition 1 (device circuit).

From the generic interface comparisons in Table 5-3, a current source interface had the worst coupling characteristics of any conventional interface. This result was expected since a current source does not allow the calculation of the appropriate sensitivity parameters for nodal analysis. One of the surprising results was that the system configuration had a 29% lower convergence than the other sensitivity-based interfaces. The explanation for this lower convergence was poor interaction between the equivalent matrix and the simulator to calculate valid sensitivity parameters at initialization. In addition, the Thevinin interface was expected to have equivalent convergence with the Norton interface, but the results showed 7% lower convergence and 50% higher iteration inefficiency (AIC of 17%).

The Norton interface had higher convergence than the Flexible interface in a generic sense, although each interface used the same calculations. The only difference was that the Flexible interface required SPICE to calculate the return current through the voltage source. Most discrepancies in the general cases were in all interfaces that contained a voltage source. The modeling issue from section 3.5.1 was another explanation for the problems, so different interface combinations would have different characteristics based on how the interface interacted with the model. Based on specific interface comparisons in Table 5-3, the Flexible and Norton combinations had nearly equivalent results. The SYSTEM-FLEXIBLE and SYSTEM-NORTON combinations had the best convergence and iteration inefficiency results of the conventional interfaces.

5.2.3 Flow Correction Configuration

The flow correction process had the best initialization characteristics of any interface, and most examples found a stable initialization point. Unfortunately, this process was typically not as iteration efficient as the other interfaces. The iterations per analysis point of the correction process in Figure 5-3 were consistent with other interfaces, so the poor iteration efficiency was the result of more time steps in the analysis. The flow correction process had the worst problems with violating the backplane iteration limit. However, these results used the default backplane setting and no attempts were made to optimize the CORRSENSRATIO parameter for a particular problem.

The iteration results in Figure 5-1 showed very sporadic responses that were dependent on the sensitivity calculation options. The dependency on the sensitivity options indicated that the minimization criterion from section 3.4.3 was not very robust and the sensitivity parameters for the predictor model were erroneous. The problem was that the applied flow delta values were too small to generate good sensitivity information for the internal behavioral model. Clearly, a better delta-calculation method was required for this interface procedure. The flow correction process also became unstable as the modeling error increased based on the decrease in the convergence statistics. The decreasing convergence with increasing modeling error could have been the result of the sensitivity calculation problems.

Before testing the parallel and sequential sequencing options, the sequential process was certain to have lower performance because the processes occurred in series. The sequential option had little merit in the general coupling process, since most procedures do not have a wait state for results from another simulator. For these two reasons, the sequential procedure was considered very ineffective unless all simulators were in this mode. The sequential method had to be twice as iteration efficient as the parallel method values before the sequential method would be viable option in the normal coupling process. A summary of the flow correction process is presented in Table 5-4 over the different options considered in section 3.4.

The sequential method was only 20% percent better in iteration inefficiency compared to the parallel method while convergence characteristics were approximately equivalent. The CORRECTOR and CORRECTOR_SENS interfaces had approximately the same convergence, but the CORRECTOR interface had 5% better iteration efficiency than the CORRECTOR_SENS interface in the full sensitivity mode. In most cases, the DIAGONAL and IDENTITY options in the flow correction process were an unreliable approach to the problem, except in very specific problems.

Table 5-4. E2E flow correction process characteristics

Sensitivity Option	CORRECTOR		CORRECTOR_SENS	
	Parallel	Sequential	Parallel	Sequential
Convergence (%)				
Full	83	87	79	77
Diagonal	12	28	27	14
Identity	14	0	5	0
Iteration Inefficiency				
Full	2.46	1.98	2.58	2.41
Diagonal	3.48	0.85	1.49	1.18
Identity	0.14	-	0.03	-

5.2.4 Dynamic Interface Configuration

A summary of the dynamic interface configuration is presented in Table 5-5. The results were based on the default CAUSRATIO setting of ten, so further optimization of this process was possible. The Norton-Flexible-Voltage configuration had the highest iteration efficiency of ANY interface, but the convergence percent was too low to be a viable procedure. The dynamic interface processes that used a SYSTEM configuration had higher convergence rates, and these procedures were competitive with the optimum conventional interfaces. The present implementation did have flaws at initialization.

The initialization problem with the dynamic interface configuration represented a contradiction. This dynamic process had to determine the interface to calculate the sensitivity parameters properly, but the sensitivity parameters were required for determining the proper interface. Depending on the coupling problem, certain interfaces had better initialization characteristics to calculate valid sensitivity parameters. For example, the SYSTEM configuration was typically required in high-gain problems to complete the feedback paths while a voltage source was ideal for devices. In retrospect, the dynamic interface configuration should have implemented an interface diagnostic routine where all

Table 5-5. E2E results for dynamic interface configuration

Interface Definition	Convergence (%)	Iteration Inefficiency
Dynamic 1 Current-Flexible-Voltage	49.1	1.74
Dynamic 1 Current-Flexible-Thevinin	54.0	2.09
Dynamic 1 Current-Flexible-Norton	47.1	1.76
Dynamic 1 Norton-Flexible-Thevinin	76.6	1.62
Dynamic 1 Norton-Flexible-Voltage	58.1	1.33
Dynamic 1 System-System–Thevinin	85.8	2.07
Dynamic 1 System-System-Voltage	73.6	1.60
Dynamic 2 Tiered Method	88.8	1.49

interface configurations were tested one-at-a-time to determine the optimum initial configuration. These tests still showed that the dynamic interfacing process had merits in the coupling process, although additional procedures are required to make the interface "smarter".

5.2.5 Summary

The iteration efficiency of the coupling process was a strong function of the sensitivity-scheduling options while convergence was a strong function of the interface definition although certain interface and sensitivity options combinations did react poorly together. In particular, the convergence of the flow correction process showed larger dependencies on the sensitivity options than the conventional and dynamic interfaces. However, the flow correction process was the only interfacing procedure with 100% convergence across all the examples, but only one particular model representation using a specific sensitivity-calculation option achieved this result. The tiered dynamic interfacing, SYSTEM-FLEXIBLE, and SYSTEM-NORTON interface had the optimum convergence characteristics at 89% while the flow correction process using a CORRECTOR interface was at 83%. The tiered dynamic interfacing, SYSTEM-FLEXIBLE, and SYSTEM-NORTON interfaces had the same optimum iteration inefficiency at 1.5 while the flow correction process using a parallel CORRECTOR interface was at an iteration inefficiency of 2.5 (AIC of 40%).

5.3 Electrical-to-Mechanical (E2FE) Coupling

The SPICE to FESIM tests were variations of different examples from section 5.2. The coupling process with the FE simulator was more timing consuming than the previous tests, and the average analysis time of each E2FE analysis was 10-20 minutes. For this reason, only one of the 18 different calculation and sensitivity parameters was tested since the E2E examples showed that convergence was a strong function of the interface definition and a weak function of the sensitivity option. In these tests, the goal was to determine how the FE simulator responded to different interface configurations, since the fundamental procedures of the two simulators were different. This FE simulator solved for effort relationships and then calculated the flow response based on the effort values. The FE simulator had rounding problems calculating the return currents when an interface had large conduction values. In comparison, the SPICE simulator solved effort and flow relationships using nodal analysis or MNA.

Only the conditional sensitivity option was tested to evaluate the convergence and iteration efficiency of the 76 different interfaces over 14 (6 no-feedback and 8 high-gain feedback) examples for 1064 analyses. The interface definitions and the iteration results for the different examples are provided in Appendix-D. As a summary, the average iteration results are presented in Figure 5-4, the convergence statistics are in Figure 5-5, and the average iterations per transient point are in Figure 5-6. The E2E coupling results for the same sensitivity parameter and examples were included for reference in Figure 5-4 through Figure 5-6. These results showed that the backplane coupled different types of simulators.

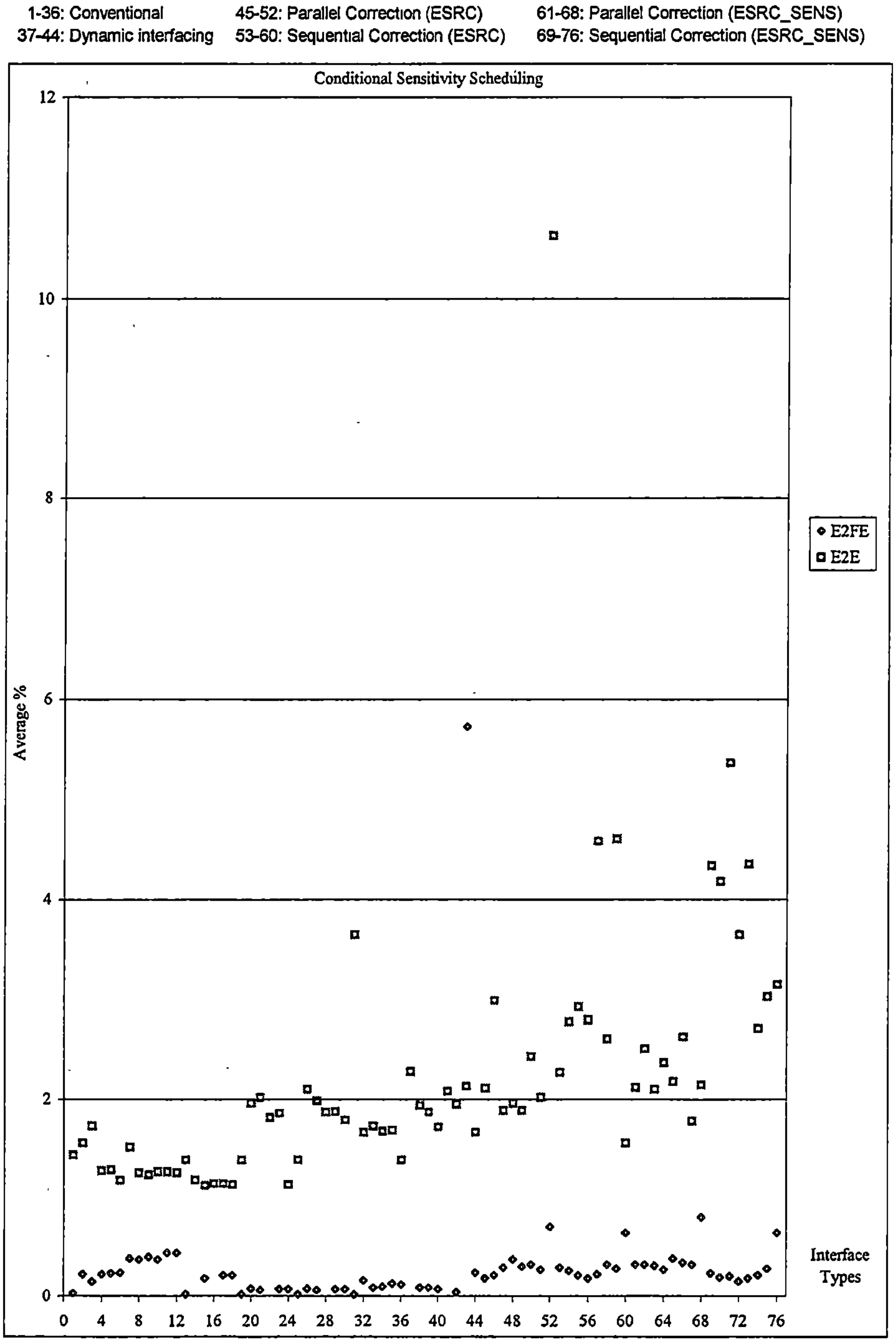


Figure 5-4. Iteration inefficiency statistics for all E2FE coupling examples

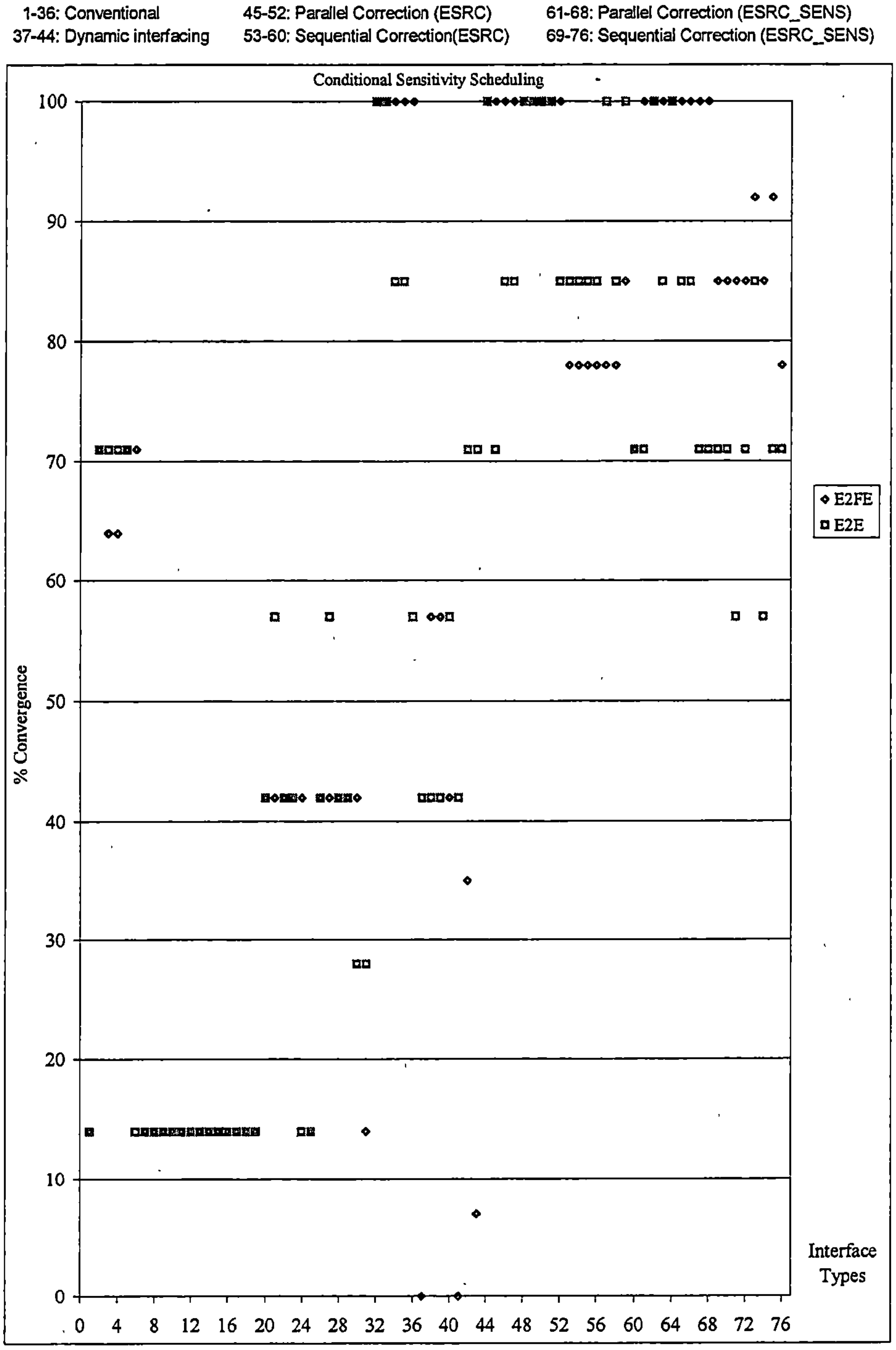


Figure 5-5. Convergence statistics for all E2FE coupling examples

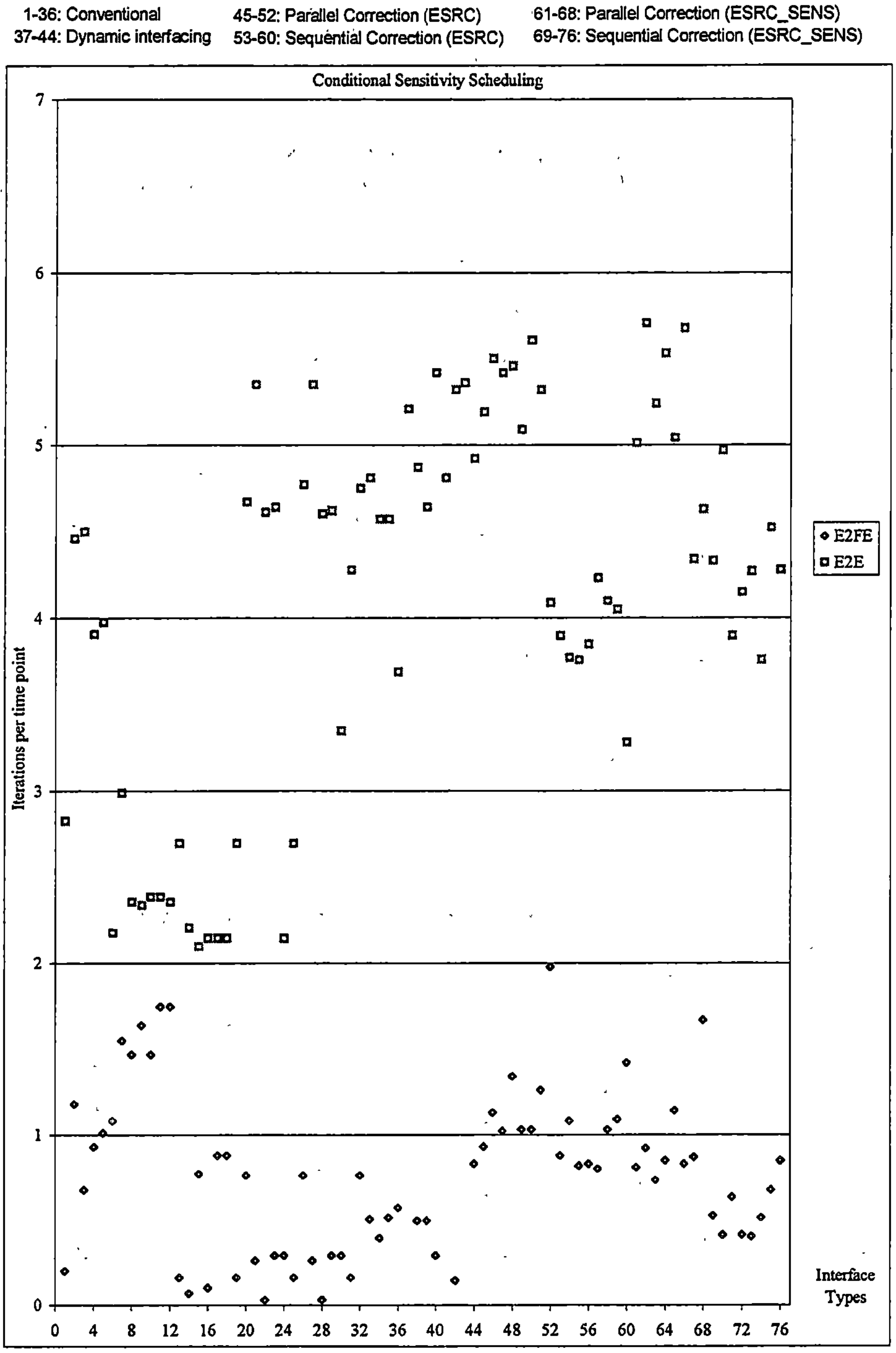


Figure 5-6. Iterations per time point statistics for all E2FE coupling examples

In addition, the flow correction procedure was examined using models that were not scaled versions of the true model representation.

5.3.1 Conventional Configurations

The interfaces used by the simulator had a small effect on the iteration efficiency of the coupling process. An analysis by interface type is presented in Table 5-6, where any interface combination using the specific type was included in the average. Table 5-6 also shows the interface combinations with greater than 85% average convergence rate. The generic results were almost equivalent with the E2E tests in terms of convergence except that the SYSTEM configuration was significantly improved. In these examples, the SYSTEM configuration for the FE Simulator implemented a FLEXIBLE configuration, since a matrix-loading feature was not available in the FE simulator. Several interfaces had 100% convergence and equivalent iteration inefficiency. The convergence and iteration

Table 5-6. Statistics for generic and specific E2FE interfaces

Interface Definition	**Convergence (%)**		**Iteration Inefficiency**	
	E2FE	E2E	E2FE	E2E
Any Current	14	17	0.02	1.85
Any Voltage	47	47	0.15	1.67
Any Thevinin	46	52	0.13	1.67
Any Norton	41	40	0.12	1.44
Any Flexible	43	40	0.21	1.36
Any System	37	14	0.21	1.30
System-Voltage	100	100	0.16	1.67
System-Thevinin	100	100	0.09	1.73
System-Norton	100	85	0.10	1.68
System-Flexible	100	85	0.13	1.69
System-System	100	57	0.12	1.39

results were improved compared to the E2E tests because of the linear modeling.

5.3.2 Flow Correction Configuration

A summary of the flow correction process is presented in Table 5-7 for the full-sensitivity calculation process with the two interface configurations. These examples did a better examination of the flow correction process, since the E2E tests had used scaled versions of the same basic model. In this series of analyses, the predictor model within SPICE was constructed manually with errors from the FEA. In most examples, the errors specified in the corrector process in Appendix-D were much larger than indicated.

The iteration inefficiency for the E2FE tests using the correction procedure was four times worse than the conventional interfaces, while the E2E tests were about two times worst than the conventional interfaces. A detailed analysis of the E2FE tests showed that the one and two terminal examples had ***equivalent*** iteration inefficiency with the conventional interfaces. The four-terminal E2FE examples had significantly skewed the iteration results, because the SPICE behavioral models in the four terminal examples were not defining all coupling information between the terminals like the FE models. Consequently, the flow correction process had to compensate for missing coupling information in addition to the problems from section 3.5.3.

Table 5-7. E2FE flow correction process characteristics

Interface Definition	**Convergence (%)**		**Iteration Inefficiency**	
	E2FE	**E2E**	**E2FE**	**E2E**
Parallel CORRECTOR	100	91	0.33	3.24
Parallel CORRECTOR_SENS	100	84	0.38	2.23
Sequential CORRECTOR	78	87	0.30	3.02
Sequential CORRECTOR_SENS	86	69	0.26	3.85

5.3.3 Dynamic Interface Configuration

Dynamic interface configuration had the characteristics shown in Table 5-8 for the E2FE examples. The FE simulator had a serious problem with the FLEXIBLE interfaces in a voltage source configuration, where rounding errors occurred in the flow calculation procedure due to the large PMAX conduction values and internal FESIM variable tolerances. Besides this problem, the tiered dynamic method had 12% higher iteration inefficiency than the conventional interfaces. This increased inefficiency was attributed to the unity conduction values, which were relatively large compared to the material definitions, and rounding problems occurred. Clearly, the tiered dynamic interface procedure required additional simulator-based constraints on the interface state decisions to optimize coupling efficiency.

5.3.4 Summary

The E2FE examples had several interfaces with 100% convergence and varying iteration inefficiencies: SYSTEM-VOLTAGE (0.16), SYSTEM-THEVININ (0.09),

Table 5-8. E2FE results for dynamic interface configuration

Interface Definition	**Convergence (%)**		**Iteration Inefficiency**	
	E2FE	E2E	E2FE	E2E
Dynamic 1 Current-Flexible-Voltage	0	42	-	2.28
Dynamic 1 Current-Flexible-Thevinin	57	42	0.09	1.94
Dynamic 1 Current-Flexible-Norton	57	42	0.09	1.87
Dynamic 1 Norton-Flexible-Thevinin	42	57	0.07	1.72
Dynamic 1 Norton-Flexible-Voltage	0	42	-	2.08
Dynamic 1 System-System–Thevinin	35	71	0.04	1.95
Dynamic 1 System-System-Voltage	7	71	5.73	2.13
Dynamic 2 Tiered Method	100	100	0.24	1.67

SYSTEM-NORTON (0.10), SYSTEM-FLEXIBLE (0.13), SYSTEM-SYSTEM (0.12), tiered dynamic configuration (0.24), and parallel flow correction with a CORRECTOR (0.33) or CORRECTOR_SENS (0.38) interface. The E2FE examples had significantly better iteration efficiency than the E2E examples because no model discontinuities were present and linear modeling was used. The FE simulator did have problems when large conduction values were applied to the interfaces with sensitivity elements because of the flow calculation procedures in the FE simulator. The dynamic configuration processes had the most difficulties with this problem.

5.4 Dynamic modeling via a simulation backplane

In simulation, the main goal is always to achieve the highest performance and highest accuracy possible, and dynamic modeling was implemented to achieve this goal. Except for the ACCURACY and RANGE triggers, most triggering options were dependent on the designer's viewpoint of how modeling should change to achieve certain performance or accuracy constraints. These types of triggers forced the user to have an interactive role in the simulation process to account for the modeling variations, different simulation algorithms, and different disciplines. Model verification was a critical issue across the different simulation environments, where different models were used to represent the same component or device. For an automated approach to the verification problem, the ACCURACY and RANGE triggers could dynamically find the model in the highest performance simulator that guaranteed certain levels of accuracy AT THE DEVICE LEVEL. Unfortunately, the real accuracy issues were at the system-level.

This section demonstrates and examines the performance and accuracy results of analyses based on an ACCURACY trigger. The ACCURACY trigger was a mechanism achieving concurrent verification and simulation, where the most accuracy model defined by the trigger was always tracking the analysis of the active modeling representation. The RANGE trigger was a pre-verified form of the ACCURACY trigger based on error information and simulation performance gathered from device testing or low-level analyses. For more information, Appendix-A outlines the procedures for the ACCURACY and RANGE trigger. Since the two methods were closely related, the RANGE trigger results should follow the ACCURARCY trigger results. For this reason, no examination was done on the RANGE trigger.

For these dynamic modeling tests, the E2FE-HF_DRVVIR2 example (Figure D-48) was used. The SPICE behavioral model used in these tests was shown in Figure D-2. This model was manually created from the FE analysis and errors were deliberately present to demonstrate the triggering procedures. The following coupling architectures, interfaces, and modeling constraints were tested:

- SPICE-only tests using the model derived from FEA.
- FEA using the SPICE response as the stimuli. The voltage stimuli were converted manually from the SPICE results into the FE simulator to drive the device.
- Normal coupling process (SYSTEM-ESRC interface).
- Flow correction process (using the model derived from FEA).
- Modeling breakdown problem.

The results from these tests are analyzed based on accuracy (section 5.4.1) and performance (section 0) to determine if the ACCURACY-based triggers in this research were a viable procedure. A model breakdown problem was included to demonstrate the inaccuracies of poor modeling by emulating a structural break. The break problem also determined if the flow correction process was capable of correcting for large (3+ orders of magnitude) errors in the SPICE behavioral model. The main issue was whether the performance improvement was justified by the loss in accuracy, but only the designer(s) knows the minimum constraints of the system.

5.4.1 Accuracy Comparison

The absolute tolerances for the coupling analyses were 0.1mV for the effort variables and 0.1 μA with a relative tolerance of 0.1 percent for the flow variables. The absolute tolerance for the ACCURACY trigger was 0.2mV for effort and the 0.2 μA for flow. In these analyses, the SPICE behavioral model had approximately 24 percent error compared to the FESIM model due to "capacitive" material effects not included in SPICE. A summary of the ACCURACY trigger results is presented in Table 5-9 for this example. Various error definitions and switch back (SB) conditions were examined to determine the impact on the system error compared to the absolute response from the non-switched coupled analysis.

The accuracy information in Table 5-9 showed that the system error did not vary according to the device error over all error tolerances. As the tolerance on the ACCURACY trigger increased, a tolerance value was reached (between 10% and 20% error) where the system error became independent of the trigger. Part of the problem was the length of the

Table 5-9. Error summary for the ACCURACY trigger tests

Comparisons	Effort Error			Flow Error		
(Most compares are to the	**Max**	**Avg**	**Peak**	**Max**	**Avg**	**Peak**
REFERENCE)	*mV*	*mV*	%	*μA*	*μA*	%
SPICE to FESIM	-	-	-	4.54	1.51	23.9
SPICE only	127.0	34.2	26.9	8.64	2.81	36.6
FESIM using SPICE output	127.0	30.6	26.9	5.55	1.45	23.6
Coupled analysis	REFERENCE					
5% error and 1 step SB	19.6	5.1	4.1	0.86	0.59	3.8
5% error and 2 step SB	19.9	5.1	4.2	1.49	0.27	6.3
10% error and 1 step SB	20.4	5.4	4.3	3.25	0.27	13.8
10% error and 2 step SB	20.4	5.4	4.3	3.25	0.27	13.8
20% error and 1 step SB	85.1	37.8	18.0	5.96	1.76	25.3
20% error and 2 step SB	40.4	10.2	8.6	3.37	0.66	14.3
50% error and 1 step SB	72.2	16.1	15.3	5.26	1.13	22.3
50% error and 2 step SB	50.2	19.3	10.6	4.57	1.12	19.4
Coupled analysis and the FESIM in TRACK mode	120.7	34.8	25.6	5.17	1.59	21.9
Flow correction analysis	0.7	0.2	0.1	0.03	0.00	0.1
5% error and 1 step SB	19.2	5.2	4.1	0.83	0.26	3.5
5% error and 2 step SB	15.2	4.5	3.2	1.48	0.25	6.3
10% error and 1 step SB	26.5	6.3	5.6	2.90	0.30	12.3
10% error and 2 step SB	45.3	11.4	9.6	2.56	0.52	10.9
20% error and 1 step SB	83.6	29.1	17.7	4.80	1.35	20.4
20% error and 2 step SB	50.5	24.2	10.7	4.58	0.98	19.4
50% error and 1 step SB	81.9	30.2	17.4	5.10	1.48	22.4
50% error and 2 step SB	12.7	4.4	2.7	4.51	0.68	19.2
Correction analysis and the FESIM in TRACK mode	119.3	33.8	25.3	5.07	1.54	21.5
Break coupled analysis	508.3	206.3	107.8	9.63	4.07	40.8
Break correction analysis	473.3	183.3	100.4	10.11	4.14	42.9

$$Absolute\ Maxium = MAX\left(\left|x_{REF} - x_{CMP}\right|\right) \qquad Absolute\ Average = \frac{1}{p}\sum_{0}^{p}\left|x_{REF} - x_{CMP}\right|$$

$$\mathrm{Re}lative\ Peak = 100 \cdot \frac{MAX\left(\left|x_{REF} - x_{CMP}\right|\right)}{MAX\left(x_{REF}\right) - MIN\left(x_{REF}\right)}$$

simulation, since the error accumulation would typically increase as the simulation proceeded. Conversely, the nature of system could be virtually independent of the device error. Ultimately, the value of the ACCURACY trigger in an analysis depended on sensitivity of the system to device error. However, the various analyses using the ACCURACY triggers were still more accurate than the SPICE behavioral model that had errors compared to the FE model.

The SPICE, FESIM, and non-switching coupling responses are presented in Figure 5-7 and Figure 5-8 to establish the basis of the comparisons. Figure 5-9 and Figure 5-10 showed the responses from the conventional interfacing procedures using various error tolerance of the ACCURACY trigger. Figure 5-11 and Figure 5-12 showed the responses from the flow correction procedure over the same error tolerances of the ACCURACY trigger. The break responses are in Figure 5-13 and Figure 5-14 to show extreme divergence between the SPICE behavioral and the FE model. In these plots, the effort information was taken from the SPICE simulator while the flow variable information was taken from the FESIM to illustrate the consequences of dynamic modeling. With large error constraints, the flow responses from the FE simulator had discontinuities at the trigger points, where the solution had large corrections.

The main reason for using the ACCURACY trigger was extreme modeling errors like a break condition in Figure 5-13 and 5-14, where the most accurate model (defined by the trigger) could be used in the analysis when required. By switching models based on the ACCURACY trigger, the designer(s) did not have to implement additional simulations to correct the error. The break analyses showed the merit of an ACCURACY trigger.

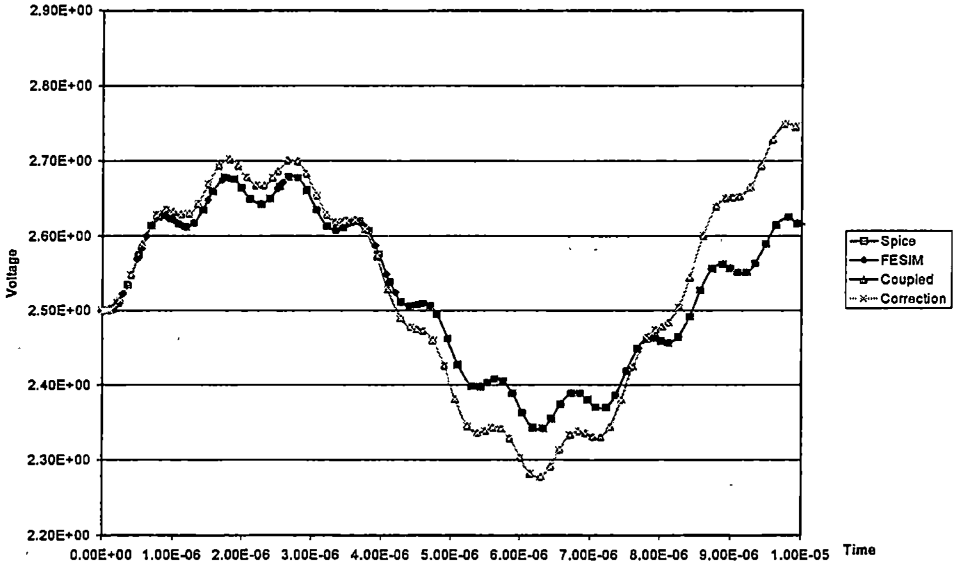


Figure 5-7. Primary effort (voltage) responses at terminal T2

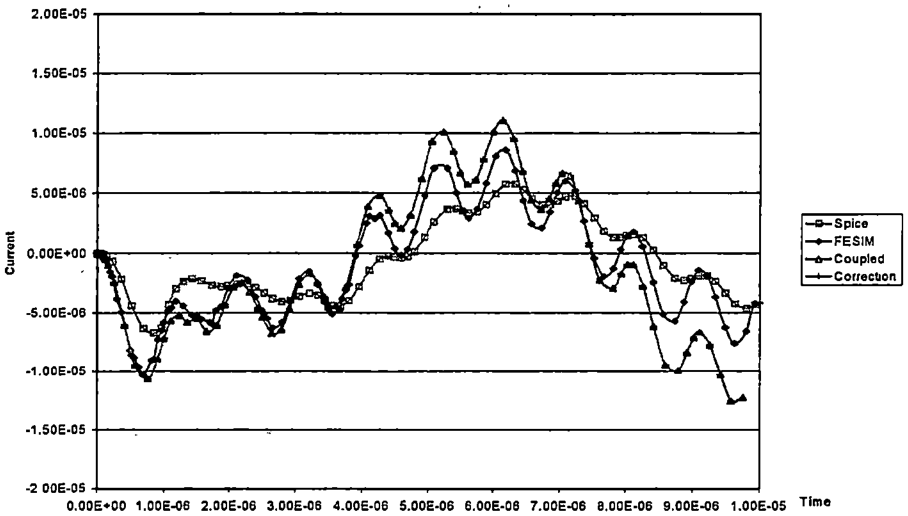


Figure 5-8. Primary flow (current) responses at terminal T2

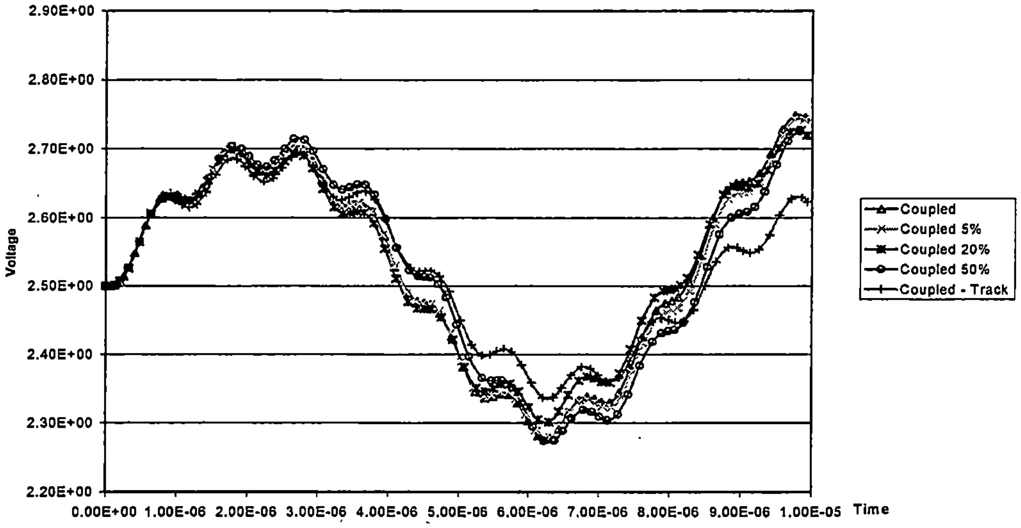


Figure 5-9. Effort responses at terminal T2 of the triggered coupled analyses

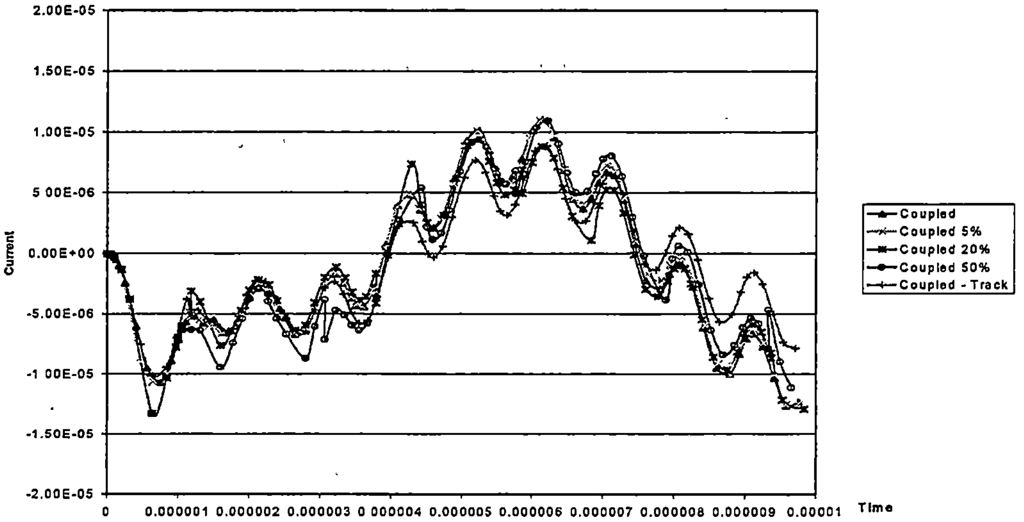


Figure 5-10. Flow responses at terminal T2 of the triggered coupled analyses

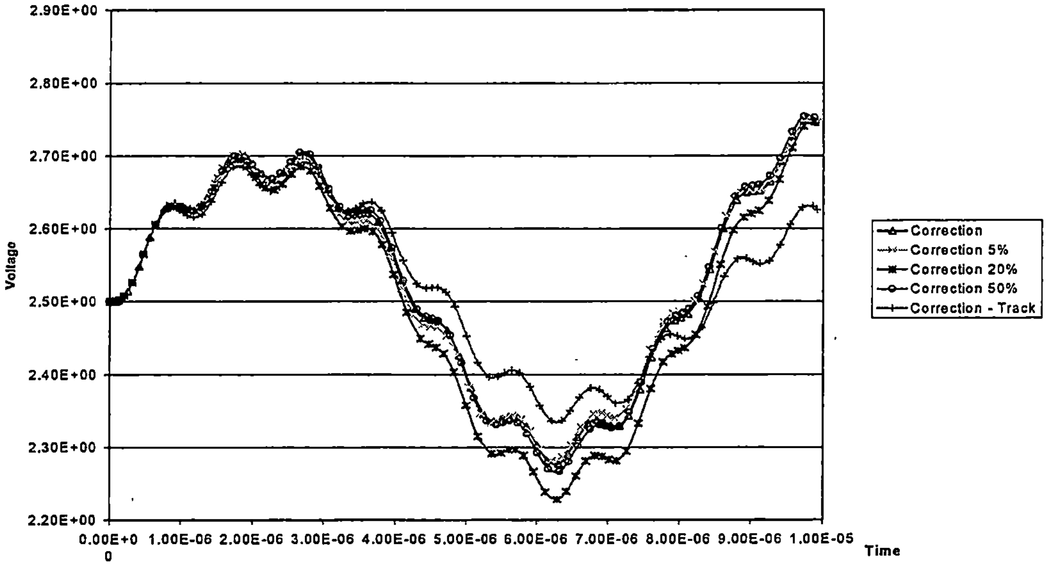


Figure 5-11. Effort responses at terminal T2 of the triggered correction analyses

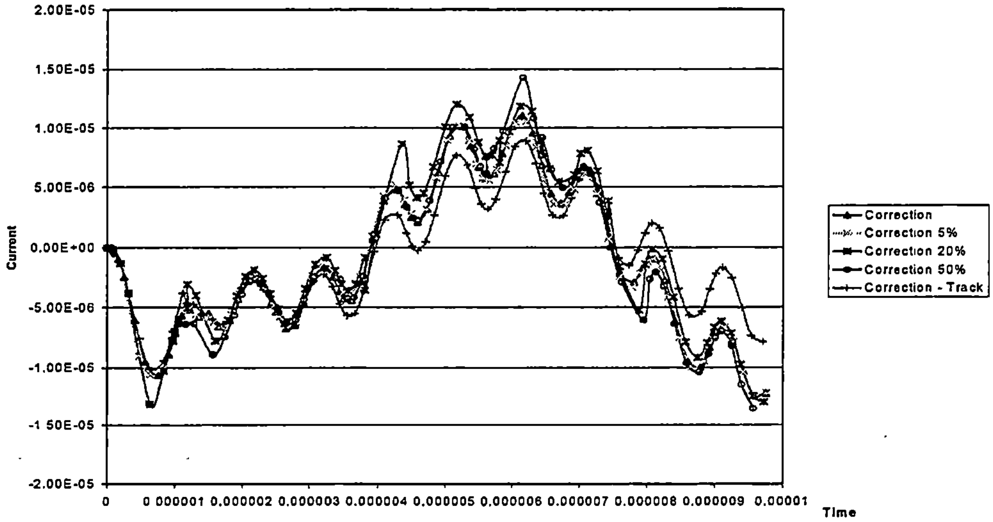


Figure 5-12. Flow responses at terminal T2 of the triggered correction analyses

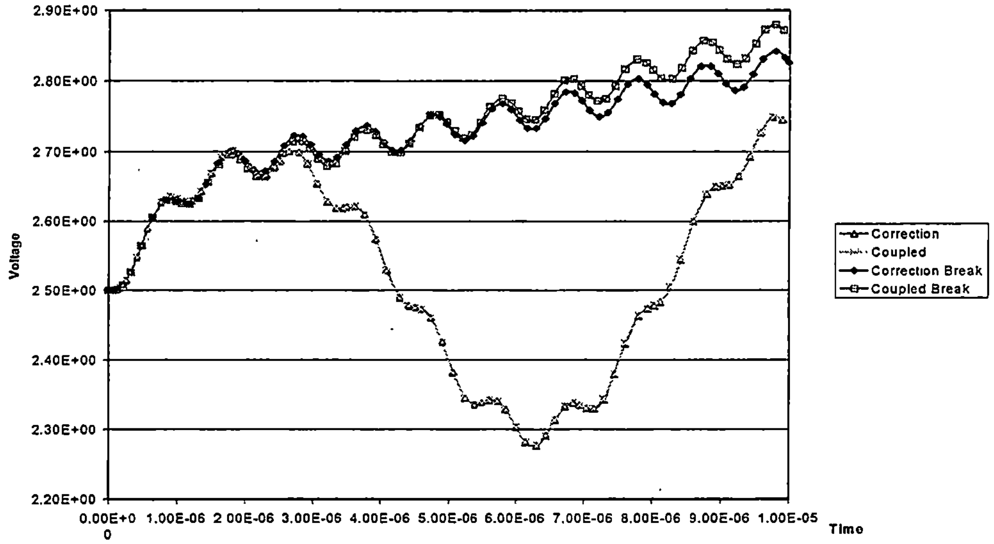


Figure 5-13. Effort responses at terminal T2 of the break analyses

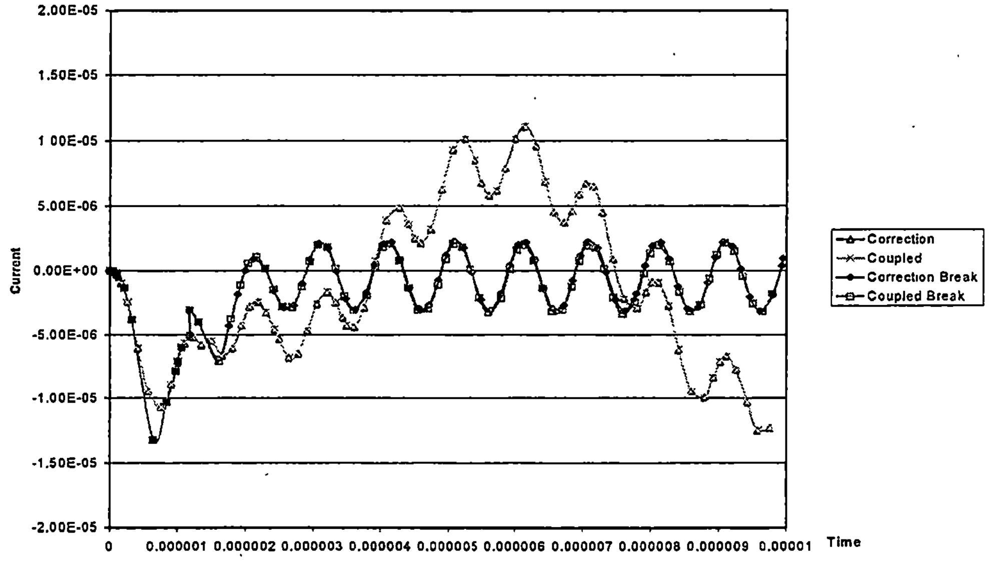


Figure 5-14. Flow responses at terminal T2 of the break analyses

One of the significant accuracy issues was whether the flow correction process would converge under extreme situations. Although the effort response began to diverge between the coupled and correction processes (see Figure 5-13), the flow correction process did converge in the break emulation situations. The analysis from section 3.5.2 (Equation 3-40) indicated that overestimating the model's conduction characteristics would converge in any situation with sufficient iterations. The SPICE behavioral did overestimate the response compared to the break situation. However, the inverse problem of a structural contact or short condition would have failed based on Equation 3-40. The validity of this statement is left for future research.

5.4.2 Performance Comparison

In coupled analyses, the FE simulator completely dominated the analysis time as shown in Table 5-10, which outlines the variation in simulation time with different error tolerances of the ACCURACY trigger. As expected, performance was improved by eliminating the interaction between the SPICE simulator and the FE simulator, and the value of eliminating sensitivity calculations can be observed. The standard analyses had between 86 and 133 time steps based on the SPICE and FESIM analysis. In the coupled analyses, the total number of simulator iterations was at least 2-3 times the number of time steps. Finally, the performance of the FEA was improved by 8% when using the tracking mode compared to applying a waveform from the analysis. The tracking mode eliminated breakpoint conditions implemented in the FE simulator and reduced the number of time steps in the analysis.

Table 5-10. Performance results for the ACCURACY trigger tests

Simulation Task	Simulation Time (seconds)	Number of Iterations
SPICE only	< 1.0	86
FESIM only	25	133
Coupled analysis	94	439 / 434
With 5% error trigger	94	448 / 179 / 435
With 20% error trigger	81	493 / 222 / 422
With 50% error trigger	63	502 / 249 / 355
Coupled analysis and the FESIM in TRACK mode	24	435 / 250 / 124
Flow correction analysis	87	424 / 430
With 5% error trigger	89	423 / 445
With 20% error trigger	77	382 / 405
With 50% error trigger	63	323 / 352
Correction analysis and the FESIM in TRACK mode	23	125 / 119
Break coupled analysis	90	470 / 200 / 446
Break correction analysis	83	414 / 430

5.5 Summary

The E2E and E2FE tests demonstrated the backplane process had the ability to couple two different types of simulators although the backplane had some initialization and sensitivity calculation problems. Different degrees of convergence and iteration efficiency were achieved over a variety of interfaces and sensitivity calculation options. The convergence of the coupling process was a strong function of the interface definition while the iteration efficiency was a strong function of the sensitivity parameter and general calculation options. However, the testing of the backplane was only over a small number of sensitivity calculation options, so parameter settings are potentially not optimized for each problem. Based on the E2E examples, the conditional sensitivity scheduling option had the highest convergence statistics of scheduling options available in this backplane. However, the error sensitivity scheduling option had the best iteration efficiency by approximately 15% with a one-percent lower convergence rate.

The statistics for all interfaces with 85% or greater average convergence is presented in Table 5-11 over all examples based on the conditional sensitivity scheduling option. As expected, the interface configurations that completed the feedback or gain paths in a simulator had the highest convergence characteristics as indicated by the four interface combinations that contained a SYSTEM interface in partition 0. In the coupling tests, partition 0 usually contained the driver or gain circuitry while partition 1 contained the device. The tiered procedure for dynamic interface configuration was overall the optimum interface procedure at 95% convergence because the interface could be used in any

Table 5-11. Final Interface Convergence and Iteration Inefficiency comparison

Interface	Convergence (%)	Iteration Inefficiency
System-Voltage	86	1.18
System-Thevinin	91	1.19
System-Norton	93	1.12
System-Flexible	93	1.08
Dynamic 2- Tiered method	95	1.15
Parallel Flow Correction with CORRECTOR	93	2.21
Parallel Flow Correction with CORRECTOR_SENS	89	2.01
Sequential Flow Correction with CORRECTOR	86	1.67

situation. The flow correction process had sensitivity calculation problems that limited the success of the procedure and increased the iteration inefficiency compared to the other processes. However, the flow correction process was the ONLY interface that did not have any initialization problems.

The dynamic modeling process was very difficult to evaluate although the results showed how performance and accuracy tradeoffs were possible. The dynamic process using ACCURACY triggers allowed concurrent verification of a model/device/design with the option to switch dynamically to a different representation if errors exceeded certain limits. However, the impact of device errors on the system error was impossible to define without analyzing the system and the sensitivity of the system to the device variations. Consequently, dynamic modeling had the potential to create errors, at least compared to an absolute reference. In the best case scenario, system error could only be minimized because error was typically accumulated during the analyses and the switching procedures. For the other type of triggering options, dynamic modeling procedures were very dependent on the designer viewpoint of how the modeling should change based on conditions in the system.

Chapter 6. Conclusions

The previous chapters have outlined the fundamental mathematical and implementation aspects of a simulation backplane with dynamic configurability. With the mathematical foundation based on bond graph theory, this coupling procedure can be extended to any domain with minimum effort. This backplane had a large number of options to control and optimize the coupling process between simulators. The backplane also had the capabilities to transverse the different coupling architecture. Consequently, this backplane structure provided backplane configuration management that supported hybrid coupling environments and dynamic changes. The simulation backplane developed in this research was called FLEXBKPL. This section answers the interfacing questions from Chapter 1 that have not been answered in the previous sections. Conclusions about the coupling process are made and suggestions are given for future research.

6.1 Interface conclusions for coupling simulators

The convergence of the coupling process was clearly a function of the interface definitions and the backplane initialization procedure to calculate valid sensitivity information and to achieve the "sufficient close" initial solution (guess) required by all iteration routines. After initial convergence was achieved, the iteration efficiency was a function of the calculation and sensitivity calculation parameters, which did cause approximately 3% degradation of the convergence statistics in the electrical-to-electrical coupling tests from Chapter 5. Based on these characteristics, the optimum interface was the interface or interface procedure with the higher convergence.

In this research, the optimum interface was the tiered procedure for dynamic interface configuration using the Flexible interface. This interface had the best average convergence at 95% (using the conditional sensitivity scheduling option). The failure conditions were the result of backplane initialization procedures that failed to find a "sufficiently close" initial solution and that did not calculate valid sensitivity information. A secant-based initialization procedure using large delta values could potentially eliminate these problems. Compared to the tier dynamic interface procedure, certain specific interface combinations did have equivalent characteristics, but only this configuration could be randomly assigned in the coupling process to any simulator and still achieved the highest convergence rate. The tiered dynamic configuration process can still be improved with better initialization procedures based on the simulator's interfacing characteristics.

The flow correction process had sensitivity calculation problems that limited the success of the interface procedure to a convergence of 93% using the conditional sensitivity scheduling option. The iteration statistics of this process was 50% larger than the other interfaces with greater than 85% convergence. Because of the internal modeling requirements, this overlapped modeling process had more coupling constraints than the other interfaces and procedures examined in this research, which did limit the usefulness of the process. However, this procedure over the given tests always found a valid initialization condition, which was the main problem with the other interfaces. If the sensitivity calculation problems can be eliminated (see Chapter 3 and Chapter 5), then the flow correction process was able to implement dynamic modeling while also spanning different coupling architectures. By eliminating the corrector function from an analysis, the system-level simulator switched to the independent simulator architecture, which had the potential for the highest simulation throughput. When the corrector function was included, the flow correction process had the smallest device error compared to the "most accurate" device model in an external simulator. Another feature of the flow correction process was the ability to use the coupled response as a method of back annotation between simulators to eliminate additional coupled solutions. However, this feature was not investigated in this research.

6.2 Pros and Cons of Dynamic Operations in simulation

This simulation backplane performed three types of dynamic configuration: variable causality for determining the proper sensitivity parameters, dynamic interface configuration, and dynamic model switching. In the coupling process, the determination of the proper sensitivity parameters was essential for convergence, but the input causality detection was not an absolute requirement. In certain gain situations, the backplane was able to achieve convergence without input causality detection because of the interface definitions in the simulators. When an interface's input variable causality was identified, the sensitivity parameters did a better characterization of the models in a simulator. In addition, the calculation of the sensitivity parameters did not have the divergence potential of the procedure without the causality detection because of small delta information that caused rounding problems (See Appendix E).

The interface selection was critical in the coupling process because the interface selection did impact the convergence and the input-causality detection process. Dynamic interface configuration was implemented to eliminate the designer's task of choosing the best interface for the coupling process to maximize convergence (and iteration efficiency if possible). The results from Chapter 5 demonstrated that this configuration process did improve the convergence of the coupling process by changing the interface based on the relative coupling characteristics of a simulator. A fixed interface had to follow certain predefined calculation procedures that were not sufficiently flexible to meet any coupling characteristic of a simulator. By freeing the designer from manually choosing the interface, dynamic interface configuration was a beneficial feature in the coupling process.

The dynamic model switching procedure was a very straightforward extension of rebuilding an internal matrix within the backplane and changing which matrix elements of a simulator were added to the system-level interconnection matrix. This procedure allowed tradeoffs between performance and accuracy during an analysis. Besides performance and accuracy, each different simulation provided the designers with assurances that no fundamental problems had occurred by switching between different abstraction levels. Essentially, this procedure was another a form of verification. At the same time, the increased flexibility provided more opportunities to cause problems in the analysis, since error could accumulate in the system. As an analogy, dynamic modeling allowed the user to map the earth by using a transportation mechanism that could be any vehicle from an airplane to a bicycle. At one transition point, the user had to careful that the plane did not change to a bicycle and crash into the ground. On the reverse condition, bicycle might be unable to reach sufficient velocity to fly or the sudden change in altitude could cause the plane to spin off into space.

The ACCURACY or RANGE triggers in the dynamic modeling process were implemented as a form of verification to eliminate large-scale errors. However, error in the analysis depended on the system definition and the sensitivity of the system to the model variations. Consequently, these triggers could only minimize error at the system level while the present error remained and even accumulated in an analysis. At the same time, these trigger options had the potential to eliminate simulation errors dynamically. Regardless of the trigger mechanism, dynamic model switching placed the emphasis on the designers to identify problems. For these reasons, the only conclusion on the dynamic model switching procedure was "User Beware".

6.3 Future Research

In the present version of the backplane, most object parameters are defined by the user, and very few parameters are automatically changed during the analysis to optimize performance. Because the sensitivity calculation options had the largest impact on performance, these options should change dynamically to optimize performance without affecting convergence. Another technique to improve the iteration efficiency of the sensitivity calculation process would be grouping objects and variables within a simulator for parallel calculation of the parameters. However, a criterion would have to be developed for recalculating this information if internal grouping relationships became invalid. The information from the causality identification process in Appendix-C could be used to optimize the sensitivity calculation process. However, an immediate need for this backplane was improving the initialization procedures to calculate more accurate sensitivity information and to find a better initial solution (guess).

The simulators used in this backplane did not have the quality or the model capabilities to fully test the backplane operation. Future implementations should use commercial simulators with better modeling and more coupling between different domains to rigorously test the backplane operation. Additional interface definitions are also required in the backplane, so mixed mode simulation can be included in an analysis. With these additional interfaces, the dynamic modeling procedure could be used for verification of GDS layout using RTL and transistor level representations.

This work has been done with hand-partitioned designs to examine the coupling process. With the coupling issues resolved, a system for partitioning domains was needed where one database contained the information about all domains. The database had to track modifications across different sub-design environments, but the main database maintained ALL modeling representations. With a central database, the system could be partitioned based on any criteria or hierarchical level. The designer could select different model representations and different model revisions for the same design in the same simulation like [17].

Given the number of domains possible for MEMS, a system-level configuration and verification mechanism (architecture and tradeoff manager) would improve the system development and shorten the design cycle. The tool would define the optimum partitioning of the system based on the simulation tools, model representations, and the design stage. Based on constraints defined for a subsystem, the tool would enforce rules for modeling and simulation to avoid pitfalls. In addition, the different partitioning options can be used to guarantee an acceptable level of testing via specification verification. Specifically, the tool would require a sufficient number of cross-verified analyses using different partitions, model representation, and stimuli before defining a system to be flaw tolerant (no detectable flaws between the specifications and analyses).

REFERENCES

REFERENCES

[1] W.L. Bryan, G.T. Alley, ORNL internal communication on LDRD.

[2] H. Sandmaler, H.L. Offereins, B. Folkmer, "CAD tools for micromechanics", *Journal of Micromechanics & Microengineering*, Vol. 3, 1993, pp. 103-106.

[3] J. Bryzek, K. Peterson, "Micromachines on the march", *IEEE Spectrum*, May 1994, pp. 20-31.

[4] K.D. Wise, "Integrated Microelectromechanical Systems. A Perspective on MEMS in the 90s", *Proc. IEEE MEMS 1991*, pp. 33-38.

[5] G. Pelz, J. Bielefeld, F.J. Zappe, G. Zimmer, "Simulating Micro-Electromechanical Systems", *IEEE Circuits and Devices Magazine*, Vol. 11, Issue 2, 1995, pp. 10-13.

[6] C.C. McAndrew, "Compact Device Modeling for Circuit Simulation", *Proceedings Of The 1997 IEEE Custom Integrated Circuits Conference*, Santa Clara, May 5-8 1997, Section 8.2.1-8.2.8, pp. 151-158.

[7] S. Robinson, "SIMULATION PROJECTS. BUILDING THE RIGHT CONCEPTUAL MODEL", *Industrial Engineering*, Vol. 26, No. 9, Sept 1994, pp. 34-36.

[8] J. Gilbert, "Integrating CAD Tools for MEMS Design", *IEEE Computer*, April 1998, pp. 99-101.

[9] S.D. Senturia, "Cad For Microlectromechanical Systems", *The 8th International Conference on Solid-State Sensors and Actuators: Eurosensors IX*, June 1995, Vol. 2, Section 232-A7, pp. 5-8.

[10] A.R. Newton, A.L. Sangiovanni-Vincentelli, "Relaxation-Based Electrical Simulation", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-3, No. 4, October 1984, pp.308-331.

[11] M.P. Desai, I.N. Haljj, "On the Convergence of Block Relaxation Methods for Circuit Simulation", *IEEE Transactions on Circuits and Systems*, Vol. 36, No. 7, July 1989, pp. 948-958.

[12] N.B. Guy Rabbat, A.L. Sangiovanni-Vincetelli, H. Y. Hsieh, "A Multilevel Newton Algorithm with Macromodeling and Latency for Analysis of Large-Scale Nonlinear Circuits in the Time Domain", *IEEE Transactions on Circuits and Systems*, Vol. CAS-26, No. 9, September 1979, pp. 733-741.

[13] E. Lalarasmee, A.E. Ruehli, A.L. Sangiovanni-Vincetelli, "The Waveform Relaxation Method for Time-Domain Analysis of Large Scale Integrated Circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-1, No. 3, July 1982, pp. 131-145.

[14] G.K Ananthasuresh, R.K. Gupta, S.D. Senturia, "Approach to macromodeling of MEMS for nonlinear dynamic simulation", *Proceedings of the 1996 ASME International Mechanical Engineering Congress and Exposition*, Atlanta, Nov 17-22 1996, v 59, pp. 401-407.

[15] S. L. Garverick, M. Mehregany, "METHODOLOGY FOR INTEGRATED MEMS DESIGNS", *Proceedings of the 1996 IEEE International Symposium on Circuits and Systems, ISCAS*, Atlanta, May 12-15 1996, pp. 1-4.

[16] F. Goodenough, "Mixed-Mode Simulators Go Beyond Spice", *Electronic Design*, Oct 27 1988, pp. 77-91.

[17] E.L. Acuna, J.P. Dervenis, A.J. Pagones, F.L. Yang, R.A.Saleh, "Simulation Techiques for Mixed Analog/Digital Circuits", *IEEE Journal of Solid-State Circuits*, Vol. 25, No. 2, April 1990, pp. 353-363.

[18] S.P. Menzel, J.A. Barby, J. Vlach, "A MULTI-LEVEL SIMULATION SYSTEM FOR MOS VLSI NETWORKS", *Proc. Intl. Symp. Circuits and Systems: ISCAS 1989 Part 2*, pp. 1145-1148.

[19] P. Odryna, "A Unified Mixed-Mode Digital/Analog Simulation Environment", *Proceedings - IEEE International Symposium on Circuits and Systems*, Jun 7-9 1988, Vol. 1, pp. 893-896.

[20] D. Overhauser, I. Hajj, "IDSIM2: An Environment for Mixed-Mode Simulation", *IEEE 1990 Custom Intergrated Circuits Conference*, pp. 5.2.1-4.

[21] R. Beale, R. Chadha, C. Chen, A. Prosser, K. Tham, "Design Methodology and Simulation Tools for Mixed Analog-Digital Integrated Circuits", *IEEE Intl. Symp. on Circuits & Systems*, Vol. 2, May 1990, pp.1351-1355.

[22] S.P. Menzel, J. Vlach, "A Mixed-Mode Analogue And Switch Level Simulator", *Intl. Journal of Circuits, Theory, & Applications*, Vol. 18, Issue 1, 1991, pp. 35-50.

[23] C. Yuan, R. Lucas, P. Chant, R. Dutton, "Parallel Electronic Circuit Simulation on the iPSC(R) System", *IEEE 1988 Custom Integrated Circuits Conference*, pp. 6.5.1-4.

[24] P. Sadayappan, V. Visvanathan, "Circuit Simulation on Shared-Memory Multiprocessors", *IEEE Transactions on Computers*, Vol. 37, No. 12. December 1988, pp. 1634-1642.

[25] P.F. Cox, R.G. Burch, D.E Hocevar, P. Yang, B.D Epler, "Direct Circuit Simulation Algorithms for Parallel Processing", *IEEE Transactions on Computer-Aided Design*, 1991, Vol. 10, Issue 6, pp. 714-725.

[26] V. Klinger, "DiPaCS: A New Concept for Parallel Circuit Simulation", *Proceedings of the IEEE Annual Simulation Symposium, Proceedings of the 28th Annual Simulation Symposium*, Apr 9-13 1995, pp. 32-41.

[27] J.L. Calvet, A. Titli, "Overlapping vs Partitioning in Block-Iteration Methods: Application in Large-Scale System Theory", *Automatica*, 1989, Vol. 25, No. 1, pp. 137-145.

[28] D. A. Gates, P.K. Ko, D.O. Pederson, "MIXED-LEVEL CIRCUIT AND DEVICE SIMULATION ON A DISTRUTED-MEMORY MULTICOMPUTER", *IEEE 1993 Custom Intergrated Circuits Conference*, May 1993, pp. 8.5.1-8.5.4

[29] K. Mayaram, D.O. Pederson, "Coupling Algorithms for Mixed-Level Circuit and Device Simulation", *IEEE Transactions on Computer-Aided Design*, Vol. 11, No. 8, August 1992, pp. 1003-1012.

[30] K. Mayaram, J. Chern, P. Yang, "Algorithms for Transient Three-Dimensional Mixed-Level Circuit and Device Simulation", *IEEE Transactions on Computer-Aided Design*, Vol. 12, No 11, Nov 1993, pp. 1726-1733.

[31] W. L. Engl, R. Laur, H.K. Dirks, "MEDUSA – A Simulator for Modular circuits", *IEEE Transactions on Computer-Aided Design*, Vol. CAD-1, No 2, April 1982, pp. 85-93.

[32] V.B. Dmitriev-Zdorov, B. Klaassen, "An Improved Relaxation Approach For Mixed System Analysis With Several Simulation Tools", *1995 IEEE European Design Automation Conference*, pp. 274-279.

[33] S. Wuensche, C. Clauss, P. Schwarz, F. Winkler, "Microsystem Design Using Simulator Coupling", *Proceedings of the 1997 European Design & Test Conference*, Paris, Mar 17-20 1997, pp. 113-118.

[34] P.C. Eccardt, M. Knoth, G. Ebest, H. Landes, C. Clauss, S. Wuensche, "Coupled finite element and network simulation for microsystem components", *Microsystem Technologies 1996*, Potsdam 1996, pp. 145-150.

[35] N.C. Petrellis, A.N. Birbas, M.K. Birbas, E.P. Mariatos, G.D. Papadopoulos, "Simulating Hardware, Software and Electromechanical Parts Using Communicating Simulators", *Proc. 7th IEEE International on Rapid System Prototyping*, June 1996, pp. 78-82.

[36] A. Schroth, T. Blockwitz, G. Gerlach, "Simulation of A Complex Sensor System Using Coupled Simulation Programs", *The 8th International Conference on Solid-State Sensors and Actuators; Eurosensors* IX, June 25-29, 1995, Stockholm, Section 239-PA7, Vol. 2, pp. 33-35.

[37] K.A. Sakallah, S.W. Director, "SAMSON2: An Event Driven VLSI Circuit Simulator", *IEEE Transactions on Computer-Aided Design,* Vol. CAD-4, No 4, Oct. 1985, pp. 668-684.

[38] W.M. Zuberek, "Software Interfaces For Integrated Simulation Applications", *Annual Phoenix Conference Proceedings - Ninth Annual International Phoenix Conference on Computers and Communications,* Mar 21-23 1990, pp. 832-839.

[39] G. Schuster, F. Breitenecker, "Coupling simulators with the model interconnection concept and PVM?", *Proceedings of the 1995 EUROSIM Conference,* Sep 11-15 1995, Vienna, Austria, pp. 321-326.

[40] R. Saleh, S. Jou, A.R Newton, *Mixed-Mode Simulation and Analog Mutlilevel Simulation,* Kluwer Academic Publishers: Boston 1994.

[41] M. Zwolinski, C. Garagate, Z. Mrcarica, T.J. Kazmierski, A.D. Brown, "Anatomy of a simulation backplane", *IEE Proceedings: Computers and Digital Techniques,* Vol. 142, No. 6, Nov 1995, pp. 377-385.

[42] A.R.W. Todesco, T.H.Y. Meng, "Symphony: A Simulation Backplane for Parallel Mixed-Mode Co-Simulation for VLSI Systems", *Proceedings of the 1996 33rd Annual Design Automation Conference,* JUN 3-7 1996, pp. 149-154.

[43] M. Zwolinski, C. Garagate, T.J. Kazmierski, "Mixed-Signal Simulation Using The Alfa Simulation Backplane", *Proceedings IEEE International Symposium on Circuits and Systems,* Jun 7-9 1988, Vol. 1, pp. 390-393.

[44] J. Singh, *Techniques for Analog Multilevel Simulation,* Ph.D Dissertation 1994.

[45] P. Odryna, "The Application of Co-Simulation in Today's Design Environment", *ECN,* December 1996, pp. 161.

[46] R. Chadha, C. Viswesariah, C. Chen, "M^3 - A Multilevel Mixed-Mode Mixed A/D Simulator", *IEEE Transactions on Computer-Aided Design,* Vol. 11, No. 5, May 1992, pp. 575-585.

[47] L. Maliniak, "A/D Simulators: An Expanding Array of Choices", *Electronic Design,* December 5, 1994, pp. 95-102.

[48] S. Crary. Y. Zhang, "CAEMEMS: An Integrated Computer-Aided Engineering Workbench for Microelectromechanical Systems", *Proc. IEEE MEMS 1990*, pp. 113-114.

[49] S. Crary, O. Juma, Y. Zhang, "Software Tools for Designers of Sensor and Actuator CAE Systems", *Proc. IEEE MEMS 1991*, pp. 498-501.

[50] R.A. Buser, S.B. Crary, O.S. Juma, "Integration of the Anisotropic-Silicon-Etching Program ASEP^{TM} within the CAMEMS^{TM} CAD/CAE Framework", *Proc. IEEE MEMS 1992*, pp. 133-138.

[51] S.D. Senturia, R.M. Harris, et. al, "A Computer-Aided Design System for Microelectromechanical System (MEMCAD)", *J. of Microelectromechanical Systems*, Vol. 1, No 1, March 1992, pp. 3-13.

[51] F. Maseeh, R. M. Harris, S.D. Senturia, "A CAD Architecture for Microelectromechanical Systems", *Proc. IEEE MEMS 1990*, pp. 44-49.

[52] R.M. Harris, F. Maseeh. S.D. Senturia, "Automatic Generation of a 3-D Solid Model of a Microfabricated Structure", *Sensors and Actuators "90*, pp. 36-41.

[53] R.M. Harris, S.D. Senturia, "A Solution of the Mask Overlay Problem in Microelectromechanical CAD (MEMCAD)", *Tech. Dig. IEEE Solid-State Sensor and Actuator Workshop*, 1992 , pp. 598-562.

[54] M.A. Shulman, M. Ramaswamy, M.L. Heyens, S.D. Senturia, "An Object-Oriented Material-Property Database Architecture För Microelectromechanical Cad", *Proc. Transducers 1991*, pp. 486-489.

[55] P. Osterberg, H. Yie, X. Cal, J. White, S. Senturia, "Self-Consistent Simulation and Modelling of Electrostatically Deformed Diaphragms", *IEEE Micro Electro Mechanical Systems*, Vol. 4, 1994, pp. 28-32.

[56] J.R. Gilbert, P.M. Osterberg, R.M. Harris, S.D. Senturia, et. al., "Implementation of a MEMCAD System for Electrostatic and Mechanical Analysis of Complex Structures from Mask Descriptions", *Proc. IEEE MEMS 1993*, pp. 207-212.

[57] J.R. Gilbert, G.K. Ananthasuresh, S.D. Senturia, "3D Modeling of Contact Problems and Hysteresis in Coupled Electro-Mechanics", *Proceedings of the 1995 9th Annual International Workshop on Microelectromechanical Systems*, Feb 1996, pp. 127-132.

[58] P.M. Osterberg, S.D. Senturia, ""MEMBUILDER" AUN AUTOMATED 3D SOLID MODEL CONSTRUCTION PROGRAM FOR MICROELECTROMECHANCICAL STRUCTURES", *The 8th International Conference on Solid-State Sensors and Actuators: EUROSENSORS IX*, June 1995, Vol. 2, Section 236-A7, pp. 21-24.

[59] J.R. Gilbert, R. Legtenburg, S.D. Sensturia, "Coupled Electro-mechanics for MEMS: Applications of CoSolve-EM", *Proc. 1995 International Conference on Microelectromecahanical Systems*, pp. 122-127.

[60] J.G. Korvink, J. Funk, M. Roos, G. Wachutka, H. Baltes, "SESES: A Comprehensive MEMS Modeling System", *IEEE Microelectromechanical Systems*, Vol. 4, 1994, pp.22-27.

[61] J.M. Funk, J.G. Korvink, J. Buhler, M. Barchtold, H. Baltes, "SOLIDIS: A tool for Microactuator Simulation in 3-D", *Journal of Microelectromechanical Systems*, Vol. 6, No. 1, March 1997, pp. 70-81.

[62] J. Funk, J.G. Korvink, M Bachtold, J. Buhler, H. Baltes, "Coupled 3D Thermo-electro-mecahanical Simulations of Microactuators", *Proceedings of the 1995 9th Annual International Workshop On Micro Electro Mechanical Systems*, Feb 1996, pp. 133-138.

[63] H. Baltes, O. Paul, J.G. Korvink, "Simulation Toolbox and Material Parameter Data Base for CMOS MEMS", *MHS 1996: Micro Machine & Human Science. 7th International Symposium on Micro Machine and Human Science*, 1996, pp.1-8.

[64] I. Zelinka, J. Diaci, V. Kunc, L. Trontelj, "MODELING AND SIMULATION OF A MICROSYSTEM WITH SPICE SIMULATOR", *Informacije Midem*, Vol. 27, No. 1, 1997, pp. 18-22.

[65] Z. Mrcarica, D. Glozic, V.B. Litovski, H. Detter, "Describing space-continuous models of microelectromechanical devices for behavioural simulation", *Proceedings of the 1996 European Design Automation Conference with Euro-VHDL 1996 and Exhibition*, Sept. 1996, Geneva, pp. 316-321.

[66] J. Scholliers, T. Yli-Pietila, "Simulation of Mechatronic Systems Using Analog Circuit Simulators", *IEEE Intl. Conf. Robotics & Automation*, May 1995, pp. 2847-2852.

[67] S. Meinzer, A. Quinte, M. Gorges-Schleuter, W. Jakob, W. Sub, H. Eggert, "Simulation and Design Optimization of Microsystems Based on Standard Simulators and Adaptive Search Techniques", *European Design Automation Conference*, 1996, pp. 322-327.

[68] N. Sadowski, Y. Lefevre, C.G.C. Neves, R. Carlson, "Finite Elements Coupled to Electrical Circuit Equations in the Simulation of Switched Reluctance Drives: Attention to Mechanical Behavior", *IEEE Transactions on Magnetics*, Vol. 32, No. 3, May 1996, pp. 1086-1089.

[69] B. Nuseibeh, J. Kramer, A. Finkelstein, "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification", *IEEE Transactions on Software Engineering*, Vol. 20, No. 10, October 1994, pp. 760-773.

[70] J. Daniell, S.W. Director, "An Object Oriented Approach to CAD Tool Control", *IEEE Transactions on Computer-Aided-Design*, Vol. 10, No. 6, June 1991, pp. 698-713.

[71] J.L. Wolf, D.M. Davis, B.R. Iyer, P.S. Yu, "Multisystem Coupling by a Combination of Data Sharing and Data Parititioning", IEEE *Transactions on Software Engineering*, Vol. 15, No. 7, July 1989, pp. 854-860.

[72] S. Z. Hussian, D. Overhauser, "Automatic Dynamic Mixed-Mode Simulation through Network Reconfiguration", *ISCAS 1995: IEEE Intl Symp Circuits & Systems*, April 1995, pp. 582-586.

[73] R. Righter, J.C. Walrand, "Distributed Simulation of Discrete Event Systems", *Proceedings - 1987 IEEE International Conference On Computer Design: VLSI in Computers & Processors*, Vol. 77 No. 1, January 1989, pp. 99-113.

[74] W. Najjar, J.L. Jezouin, J.L. Gaudiot, "PARALLEL EXECUTION OF DISCREATED-EVENT SIMULATION", *1987 IEEE International Conference on Computer Design*, pp. 668-671.

[75] R.M. Fujimoto, "Parallel And Distributed Discrete Event Simulation: Algorithms And Applications", *Proceedings of the 1993 Winter Simulation Conference*, pp. 106-114.

[76] E.J. Williams, *Regression Analysis*, New York: John Wiley & Sons, Inc, 1959, pp. 37-40.

[77] C. W. Ho, A.E. Ruehli, P.A. Brennan, "THE MODIFIED NODAL APPROACH TO NETWORK ANALYSIS", *Proc. IEEE Int. Symp. on Circuits And Systems*, 1974, pp. 505-509.

[78] J.M Ortega, W.C. Rheinboldt, *Iterative Solution of Nonlinear Equations In Several Variables*, ACADEMIC PRESS, New York, 1970.

[79] L.B. Rall, "SOLUTION OF NONLINEAR SYSTEMS OF EQUATIONS", *Numerical Solution of Partial Differential Equations*, Editor: J.G. Gram, D. Reidel Publishing Company, Boston, 1973, pp. 55-105.

[80] C. Farhat, M. Lesoinne, "AUTOMATIC PARTITIONING OF UNSTRUCTED MESHES FOR THE PARALLEL SOLUTION OF PROBLEMS IN COMPUTATIONAL MECHANICS", *International Journal for Numerical Methods in Engineering*, Vol. 36, 1993, pp. 745-764.

[81] Y. Escaig, G. Touzot, M. Vayssade, "Parallezation Of Multilevel Domain Decomposition Method", *Computing Systems in Engineering*, Vol. 5 No. 3, 1994, pp. 253-263.

[82] A.V. Hudli, R.M.V. Pidaparti, "Distributed finite element analysis: scalability and performance", *Advances in Engineering Software*, Vol. 21, 1994, pp. 1-9.

[83] E. DeSantiago, K.H. Law, "An Implementation of Finite Element Method on Distributed Workstations", *Proceedings of the 1996 12th Conference on Analysis and Computation*, Apr 15-18 1996, pp. 188-199.

[84] L. Grosz, C. Roll, W. Schonauer, "A Black-Box Solver for the Solution of General Nonlinear Functional Equations by Mixed FEM", *FEMS 50 Years of Courent Elements*, Marcel Dekker Inc, New York, 1994, pp. 225-234.

[85] N.R. Lo, E.C. Berg, S.R. Quakkelaar, J.N. Simon, M. Tachiki, H. Lee, K.S.J. Pister, "Parameterized Layout Synthesis, Extraction, and Spice Simulation for MEMS", *ISCAS: Proc. IEEE Intl Symp. Circuits & Systems Conn*, Vol. 4 1996, pp. 481-484.

[86] C. F. Gerald, P. O. *Wheatley, Applied Numerical Analysis*, Ed. 5, ADDISON-WESLEY PUBLISHING COMPANY: Massachusetts, 1994.

[87] E. Lindberg, "Simulation", *The Circuits and Filters Handbook*, 1995 IEEE, CRC Press, pp. 1058-1071.

[88] R. Pratap, "MATLAB", *The Handbook of Software for Engineers and Scientists*, IEEE 1996, CRC Press, pp. 963-1004.

[89] T. A. Johansen, "SEMI-EMPIRICAL MODELING OF NON-LINEAR DYNAMICAL SYSTEMS", *Proc. IEE Colloqium Adv. Neural Networks*, 1994, No. 136, pp. 4/1-4/3.

[90] J. Sjoberg, Q. Zhang, L. Ljung, A. Benveniate, B. Delyon, P. Glorennec, H. Hjalmarsson, A. Juditsky, "Nonlinear Black-box Modeling in System Identification: a Unified Overview", *Automatica*, Vol. 31, pp. 1691-1724.

[91] L. Ljung, T. Glad, *Modeling of Dynamic Systems*, PTR Prentice Hall: Englewood Cliffs, 1994.

[92] J. E. Kleckner, *Advanced Mixed-Mode Simulation Techniques*, Ph.D Dissertation, 1984.

[93] J. Sjoberg, "ON ESTIMATATION OF NONLINEAR BLACK-BOX MODELS: HOW TO OBTAIN A GOOD INITIALIZATION", *Proceedings of the 1997 7th IEEE Workshop on Neural Networks for Signal Processing, NNSP'97*, Sept. 24-26 1997, pp. 72-81.

[94] M.C Kohn, *Practical Numerical Methods: Algorithms and Programs*, Macmillan Publishing Company: New York, 1987.

[95] O. Nagler, M. Trost, B. Hillerich, F. Kozlowski, "Efficient design and optimization of MEMS by intergrating commerical simulation tools", *Sensors and Actuators A*, Vol. 66, 1998, pp. 15-20.

[96] W.R. Stevens, *Unix Network Programming*, Prentice Hill: New Jersey, 1990, pp. 358-340.

[97] T. Quarles, "SPICE3F4 SOURCE CODE", Online ftp://ic.berkeley.edu/pub/Spice3, May 1996.

[98] W. Bangerth, G. Kanschat, "THE DEAL.II HOMEPAGE", Online http://gaia.iwr.uni-heidelberg.de/~deal, Sept 2000.

[99] K.H. Huebner, E.A. Thornton, *The Finite Element Method for Engineers*, Edition II, John Wiley & Sons: New York, 1982.

[100] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, "PVM: PARALLEL VIRTUAL MACHINE A USERS' GUIDE AND TUTORIAL FOR NETWORKED PARALLEL COMPUTING", Online http://www.epm.ornl.gov/pvm, March 1999.

[101] R. Young, I. MacPhedran, "INTERNET FINITE ELEMENT RESOURCES", Online http://www.engr.usask.ca/~macphed/finite/fe_resources/fe_resources.html, May 1999.

[102] V.B. Dmitriev-Zdorov, "Multicycle Generation- A New Way to Improve the Convergence of Waveform Relaxation for Circuit Simulation", *IEEE Transactions on Computer-Aided Design*, Vol. 17, No 5, May 1998, pp. 435-443

APPENDIXES

APPENDIX-A

Appendix-A. Backplane Control Syntax

This appendix describes the different parameter values and formats for each keyword (command) in the backplane. A brief overview of the commands is provided in Table A-1. The following information is also notes on the backplane operations and instructions for integrating the backplane into a simulator. The three categories of keywords are DATA (Table A-4 through Table A-7), CONTROL (Table A-8 through Table A-11), and COMMAND (Table A-12 through Table A-14) at the end of this appendix. The DATA keywords are for information exchange, the CONTROL keywords define the parameters for the interfaces and tasks, and COMMAND keywords initiate operations within the backplane. The backplane processing procedures are case insensitive. Both simulators and user control script interact using these commands.

Table A-1. Keywords for the simulation backplane

Keyword Codes	Description
DEVDEF	Define a device within a simulator.
GLOBPRM	Define global parameters common to all backplane elements.
GRADDATA	Define object gradient data from a simulator.
ID	Define the simulator ID during the initialization process.
INCLUDE	Include a file as a subcommand file.
MESSAGE	Messages (warnings, errors, etc...) sent to the backplane controller.
MODELDEF	Define a multi-facetted model for the dynamic switching process.
OBJDATA	Define object variable data of a simulator.
OBJPRM	Interface controls and configurations for an object.
REQSTAT	Request the status of a simulator or its internal objects, parameters, etc.
SIMCMD	Send a command directly to a simulator.
SIMPRM	Define simulator control instructions and parameter definitions.
START	All simulators are to start processing data based on their MODE parameter.
STOP	All simulators are to stop and await further instructions.
TOLERANCE	Define the convergence tolerance for variables of root objects.
TRIGDEF	Define the trigger conditions for switching between models.

Appendix-A.1 Integrating the backplane into a simulator

The main routines to interface the simulator with the backplane are described in Table A-2. The following steps should be followed to modify a simulator:

1) Create a global variable (BKPL_DEF *BKPL_def) in the main code of the simulator. All backplane structures are generated within this variable. The routine BKPL_setup() creates the BKPL_def structure. Internal simulator commands or an initialization file defined via an environment variable or filename to BKPL_setup is required to specify backplane initialization parameters for a particular simulator.
2) Create operations within the simulator to define the interface structure between the simulator and the backplane. The routine BKPL_INTERFACE_find creates an interface into the backplane and activates the backplane. If a backplane interface is not created, then the backplane does not become active and normal simulation is performed. The interface contains indexing variables for simulator access to the local solution matrix. The index variables are E_index and F_index for the EFFORT and FLOW variables solution points, respectively. The ESYS_index is for adding an external matrix (system) to the local simulator matrix, and this index variable defines the EFFORT point for adding conduction information. README files are available with the backplane source code to define the process of adding new interface or configuration types.

Table A-2. Backplane to simulator interface routines

Main Iteration Routines
BKPL_setup(char *filename) Create the backplane structure within a simulator and define the initialization control file. The filename for the simulator can be defined via this routine. If filename is NULL, then the environment variable BKPL_INIT_FILE defines the filename. A global variable called BKPL_DEF *BKPL_def is required in the simulator in order to create the backplane. All backplane structures are created within BKPL_def.
BKPL_main() This routine is the main backplane algorithm that links all backplane elements together, performs calculations, etc. Very few operations are performed outside the main routine.
BKPL_shutdown() This routine sends a shutdown message to the backplane controller and frees most internal structures created by the backplane. Sufficient structure remains for the simulator to perform a shutdown.

BKPL_REF_input(BKPL_REF *Ref, BKPL_REF *RefDelta)
BKPL_REF_input_double(double Ref, double RefDelta)
Load the simulator reference and reference delta into the backplane.

BKPL_REF_output(BKPL_REF *PrevRef, BKPL_REF *PresentRefDelta)
BKPL_REF_output_double(double *PrevRef, double PresentRefDelta)
Output the new simulator reference point and reference delta to the simulator. The present reference point used in the calculations is $\mathrm{Ref} = \mathrm{PrevRef} + \mathrm{PresentRefDelta}$.

BKPL_INTERFACE_add_attr **(BKPL_INTERFACE *interface, char *name, char *value)** Define an interface parameter (name) with value (value).
BKPL_SIM_add_attr(char *name, char *value) Define a simulator parameter (name) with value (value).
BKPL_SIM_cmdbufprs (char *Line) Process the given string as a backplane command.
BKPL_SIM_cmdwordprs (char **Words, int count) Process a list of words with length count as a backplane command.
int BKPL_INTERFACE_find **(char *objname, char *intfname, char *confname,** **BKPL_INTERFACE **interface, int domain)** Create an interface (interface) with an object name (objname), interface (intfname), and configuration (confname) for a domain (domain) (See vardef.h for domain definitions). If the interface is valid, then the routine returns 0. The routine returns a 1 if an error occurred.

Table A-2. Continued

Main Iteration Routines
Void BKPL_OBJ_INTERFACE_find **(char *objname, BKPL_INTERFACE **interface)** Find the interface that corresponds to a particular object, but the interface will be created if it does not exist. However, this routine expects the interface to created by BKPL_INTERFACE_find.
double BKPL_INTERFACE_varload **(BKPL_INTERFACE *interface, double Ref, int vartype)** Load the variable (vartype) for the interface at the given reference (REF) point. The valid definitions for vartype are define in Figure B-1 (Appendix B).

3) Add new interfaces or modify existing routines to load variable information from the backplane to the simulator. The routine BKPL_INTERFACE_varload loads variable information from the interface structure.

4) Modify the main loop of the simulator's iteration process to include BKPL_REF_input, BKPL_main, and BKPL_REF_output (in this order). The BKPL_main controls the loop and defines the flags to the simulator (See Table A-3) for controlling the simulator analysis. The majority of all backplane operations are performed in BKPL_main. The other routines are for reference control of a transient analysis. The BKPL_REF_input routine requires the simulator to define REF and REFDELTA to the backplane and BKPL_REF_output returns REFPREV and REFDELTA' as shown in Figure A-1. In this format, the simulator can adjust REF or REFDELTA'.

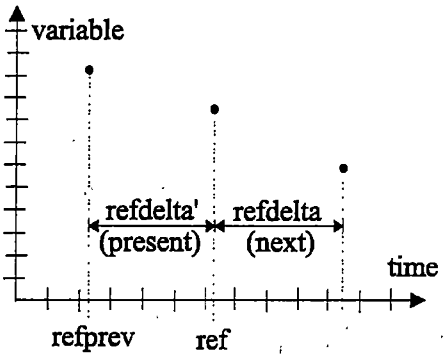


Figure A-1. Backplane reference stepping diagram

Table A-3. Flags from the backplane to the simulator

Flag (**)	Description
SHUTDOWN	Indictor to shutdown the simulator in an orderly fashion.
OPERATION	Indictor that a backplane interface was in the analysis.
SOLV_SAVE	Indictor to save the present iteration data as the solution. Without this flag, the simulator stored interim iteration data.
LOAD_FILE	Indictor to load external file that contains simulator solution and state information (BACKUP/restore function).
LOAD_STATE	Indictor to load a simulator's backup solution and states from memory (BACKUP/restore function).
ROLLFORWARD	Indictor to a simulator that the backplane will be synchronizing the simulator to new data in the future. If possible, the simulation should move to the new reference point, and restarted the analysis assuming initialization conditions applied.
NEW_TIME	Indictor that the REF variable has changed.
SAVE_FILE	Indictor to save a simulator's solution and state information into an external file (BACKUP/save function).
SAVE_STATE	Indictor to save a simulator's solution and state information into memory (BACKUP/save function).
SENS_CALC	Indictor that sensitivity calculations are being performed. The simulator's local model evaluation process can be skipped if possible.
SENS_SSAVE	Indictor of the beginning of sensitivity calculations. The simulator should save the present solution values to restore the values at the end of the sensitivity process.
SENS_SLOAD	Indictor of the ending of sensitivity calculations. The simulator can load the stored solution values, so an extra iteration is not required to return the solution back to the previous value.

** The prefix BKPL_ is applied to all flags to avoid naming conflicts with the simulator.

5) Custom routines are required prior to BKPL_main to load the new variables values (simulator output) from the local solution process into the interface structure.

6) Include BKPL_shutdown routine at all exit points from the simulator. At this point, the simulator and backplane should be capable of interacting with other simulators containing a backplane element.

7) Test and debug the backplane and simulator interaction using a simple problem.

8) **OPTIONAL** In the basic implementation, the initialization file defines object and simulator parameter settings. The initialization file is unnecessary if the simulator can internally pass information directly to the backplane. Several command processing routines are also available to implement this function.

9) **OPTIONAL** Modify the SIMCMD command so the backplane can issue commands directly to a simulator.

10) **OPTIONAL** Add or modify existing external matrix loading routines to interact with the backplane. A simulator can link to the SIMLOCAL->EqvMtx matrix and then load the equivalent matrix information using ESYS_index from two different backplane interfaces.

Appendix-A.2 Parameter Definitions

Most commands within the backplane are defined using keyword parameters. All parameters are assigned a STATIC, DYNAMIC, READONLY, or CTRLONLY property. STATIC parameters cannot be modified during calculation sequences. In most cases, these parameters are for calculation and task control. DYNAMIC parameters can be varying during any simulator mode with no restrictions. READONLY parameters are specified by the local simulator and are not modified by any source other than the local simulator, but these parameters are considered DYNAMIC parameters by other simulators. In case of problems, the READONLY parameters can be assigned a value during the OVERRIDE mode of the simulator. The CTRLONLY parameters can only be modified via the backplane routines. **The default parameter mode is STATIC unless specified otherwise.**

Appendix-A.3 Object and Device Structures

Within the backplane, two types of objects are defined: local and root. The simulator defines the local objects while the backplane constructs the root objects at the beginning of a task. A local object is the standard object definition for a simulator where each local object has a root object or system level connection. The local objects have definable attributes and define an interface for a single domain. The MERGE option in OBJPRM specifies the root object. If MERGE is not specified, then the default root object is the same name as the local object. A root object is the master node that defines how objects from simulators are merged or interconnected. The root objects have no definable attributes except for the tolerance parameters. Unlike the local object, the root objects support multiple domains, so interfaces from different domains can be merged into a single root object.

The device structures also have a local and root structure like an object, but equivalent devices in different simulator must have the same device name. Devices are essentially a collection of objects that represent a known entry with a district modeling representation. The device definitions are used to define the predictor and corrector interfaces in the flow correction process (described in section 3.5) and for the dynamic model switching process (described in section 4.3.3 and by the TRIGDEF command). At present, local objects cannot be shared by multiple device definitions. This rule is strictly enforced with the predictor and corrector interfaces, but sharing is possible with other interfaces.

Appendix-A.4 Automatic Configuration

Because of the dynamic switching capabilities of the backplane, the central controller of all backplane elements will configure all simulators especially if a device is defined for a simulator. If any simulator within the backplane is in an ITERATE mode, the controller configures the STATUS parameter of objects using the device STATUS, which the objects are used to construct. The mapping is defined as follows:

- Device ON forces objects to ON.
- Device OFF forces objects to OFF.
- Device TRACK forces objects to TRACK.

Then, a simulator's configuration is determined from the objects' STATUS property using the following rules:

- If any object in a simulator has a STATUS=ON, then the simulator can only be in an ITERATE mode. Objects with a STATUS=OFF in a simulator with a MODE=ITERATE are changed to STATUS=TRACK.
- The simulator is set to the TRACK mode if all objects have STATUS=TRACK.
- If all objects have STATUS=OFF, then the simulator is set to the DEACTIVE mode parameter and the objects of the simulator are configured appropriately.

Once the simulator status is determined, the object in the simulator are modified if necessary where:

- For a simulator in an ITERATE mode, the object status can be ON or TRACK.
- For a simulator in TRACK mode, the object status can only be TRACK.
- For a simulator in OFF mode, the object status can only be OFF.

Besides these rules, the central backplane controller performs automatic ITERROGATION of simulators with UNKNOWN modes. In addition, each backplane element will also automatically synchronize with other simulators when the element is changed from an OFF to ITERATE mode.

Appendix-A.5 Triggering Rules and Procedures

The triggering procedures are very straightforward except for the ACCURACY and RANGE procedures. The accuracy trigger is a method of concurrent verification, where the reference model defined by the trigger tracks the operation of the active model (model in the analysis). When the accuracy between the active and reference model is not within a certain tolerance, the procedure switches to another model that meets the accuracy criteria. The trigger can always switch back to the reference model, which is essentially the most accurate, but most time-consuming model in the analysis. The range trigger is a variation of the accuracy trigger, where accuracy information is already defined in lookup tables and concurrent simulation is not required. Each trigger has additional rules about model switching decisions and options.

For the ACCURACY trigger, the procedure is to chose the simulation model with the best performance once the model (as called a device) has meet the sequential accuracy criteria. The sequential accuracy criterion requires a model to meet the accuracy requirement (variable errors are less than MAXERROR) over a certain number of sequential points (SEQCNT_SWITCH) compared to the reference model. Once valid, a certain sequential number (SEQCNT_BACK) of accuracy requirement violations causes the model to become invalid. If multiple valid models are present, then the switching procedure will change models ONLY if the performance of the new model is better than the performance (P) of the active model by a certain percentage (PERFDIFF), or

$$P_{NEW} < P_{ACTIVE} \cdot (1 - PERFDIFF) \qquad \text{Equation A-1}$$

If no valid models are found, then the reference device becomes the active simulation device.

For the RANGE trigger, the device must meet the sequential range criteria. Like the sequential accuracy criteria, the sequential range criterion requires a model (device) to meet the range requirement over a certain number of sequential points (SEQCNT_SWITCH). The range requirement is that specified range error for all variables of the simulation model is less than MAXERROR of the trigger. Once valid, a certain number of sequential violations (SEQCNT_BACK) of the range requirement caused the model to become invalid. However, a model immediately becomes an invalid device if the device range exceeds MIN-MINEXT or MAX+MAXEXT for a present region without finding another region within the error tolerance parameter. If multiple valid models are present, then the switching procedure will change between valid models based on the following rules:

1. Better performance (P) than the active model by a certain percentage (PERFDIFF).

$$P_{NEW} < P_{ACTIVE} \cdot (1 - PERFDIFF)$$

2. Equivalent performance of the active model within a certain percentage (PERFDIFF) and an improved ratio of range error (E) over the minimum range (R) by a certain percentage (RANGEDIFF).

$$\begin{gathered} P_{NEW} < P_{ACTIVE} \cdot (1 + PERFDIFF) \\ \frac{E_{new}}{R_{new}} < \frac{E_{ACTIVE}}{R_{ACTIVE}} \cdot (1 - RANGEDIFF) \end{gathered} \qquad \text{Equation A-2}$$

If no valid models are found, then the SIMMOD defined in the TRIGDEF becomes the active simulation models.

Table A-4. Data keywords for the backplane

Keyword Codes	DATA Description
DEVDEF	Define a device called DEVNAME in a particular simulator and define the properties of the device. Format: DEVDEF SIMULATOR DEVNAME PARAMETERS PARAMETERS **MODELDEF={model name}** Define the model that the device references. This parameter must be specified. **SEQCORR={YES,NO}** Perform a sequential correction process. The default is NO. **SEQCORRITR={YES,NO}** On the first iteration of a new reference point, make the CORRECTOR track the PREDICTOR interface. The default is NO. **SIMMOD={simulation sub-model}** Define the specific simulator model that used in the simulator. This parameter must be specified. **SYNC-SIMMOD={simulation sub-model}** Define the specific simulator model that the given device will synchronize. The backplane controller defines this parameter. **STATUS ={ON,OFF,TRACK}** Define the status of the device in the dynamic modeling process. The default is ON. Because of the dynamic switching process, this parameter can be controlled by the user and the backplane controller. ON:_The device is active in the dynamic modeling process. OFF: The device is inactive in the modeling process. TRACK: The device is to track the operation of the ON device. **{simulator object}={model object}** Define how simulator interfaces (simulator objects) are connected to the basic model (model objects). This relationship listing provides the backplane with a secondary netlist to double-check the system-level connectivity.

Table A-4. Continued

Keyword Codes	DATA Description
DEVDEF (continued)	**CORRSENSRATIO={ floating point value }** Define the correction sensitivity ratio (RATIO) for the flow-correction minimization process. The default value is 5.0. The flow correction procedure does the following: $F^{i+1}_{C,A} = F^{i}_{C} + \Delta F^{i}_{CALC} + G_{present} \cdot \Delta E^{i}_{CALC}$ (Normal) $F^{i+1}_{C,B} = F^{i}_{C} + \Delta F^{i}_{CALC} + G_{previous} \cdot \Delta E^{i}_{CALC}$ (Compare) $if \quad (\lvert F^{i+1}_{C,A}\rvert > RATIO \cdot \lvert F^{i+1}_{C,B}\rvert \quad then \quad F^{i+1}_{C} = F^{i+1}_{C,B}$ $else \quad F^{i+1}_{C} = F^{i+1}_{C,A}$
GRADDATA	Specify the gradient information to other simulators. Format: GRADDATA SIMULATOR PARAMETERS PARAMETERS **NAME$_X$:VAR$_A$|NAME$_Y$:VAR$_B$={floating point value}** Gradient results for object variables. Examples are: T1:V|T2.A={floating point value}=$\frac{\partial V_x}{\partial I_y}$ T1:A|T2.A={floating point value}=$\frac{\partial I_x}{\partial I_y}$ T1: V|T2.V={floating point value}=$\frac{\partial V_x}{\partial V_y}$ T1:A|T2.V={floating point value}=$\frac{\partial I_x}{\partial V_y}$
OBJDATA	Define object variable data to other simulators. Format: OBJDATA SIMULATOR OBJNAME PARAMETERS PARAMETERS **REF = {hexadecimal number}** Define a reference point. An example is REF=0A43D. **VARIABLE={floating point value}** Physical variables. Examples are: V = 0.5212 (Voltage). Units are in volts (V). A = 0.3289 (Current). Units are in amps (A).

Table A-4. Continued

Keyword Codes	DATA Description
INCLUDE	Include a file with additional backplane instructions. Format: INCLUDE file
MESSAGE	Message or information to the backplane controller. Format: MESSAGE SIMULATOR TYPE {message} **TYPE={COMPLETED, ERROR, FATAL_ERROR, DATA, WARNING, TRIGGER}** Define the type of message. COMPLETED Indictor of successful completion of a task. WARNING A condition occurred in the simulator that can cause errors in the backplane. ERROR The present task encountered an error, which forced the simulator to halt the present task. The message is an error description for diagnostics. The entire backplane process can be jeopardized by the error and data is certainly suspect.] FATAL_ERROR The simulator encountered an error that was unrecoverable. The message is a description of the simulator error for diagnostics if available. In all cases, the simulator has shutdown. DATA The message is simulator information from the REQSTAT command. The information is read by the backplane controller module(s).

Table A-4. Continued

Keyword Codes	DATA Description

MODELDEF	Define a model and all facets of the model with respect to the different simulation models available.
	
	Format:
	MODELDEF *{name}*
	MODEL-VAR variables
	PARAM {**MODELDEF Parameters**}
	SIM-MODELDEF {*name*}
	PARAM {**SIM-MODELDEF Parameters**}
	RANGEDEF$_1$ {maxerror}
	RANGE {variable} {**RANGE attributes**}
	END-RANGEDEF
	{etc}
	RANGEDEF$_N$ {maxerror}
	{etc}
	END-RANGEDEF
	END-SIMMODELDEF
	{etc}
	SIM-MODELDEF *{name}*
	{etc}
	END-SIM-MODELDEF
	END-MODELDEF
	
	A separate section is provided for **MODELDEF parameters, RANGE attributes,** and **SIM-MODELDEF parameters.**
	
	Command Definitions
	END-MODELDEF, END-SIM-MODELDEF, END-RANGEDEF
	Indictors to terminate a particular command sequence within MODELDEF.
	
	MODEL-VAR OBJECT:VARIABLE ...
	Define the reference variables or IO information for the model.
	
	Example: MODEL-VAR T1:V T1:A T2:V ....
	
	(continued)

Table A-4. Continued

Keyword Codes	DATA Description
MODELDEF (continued)	**SIM-MODELDEF name** Define the start of a simulator-specific model definition. Example: SIM-MODELDEF spice3 **RANGEDEF {maxerror}** Define the regions of the device that have been mapped and the maximum error for the region (% error). The default maxerror is 0.01. Example: RANGEDEF 0.05 ➔ 5% error maximum. **RANGE OBJECT:VAR PARAMETERS** Define the ranging information for the model object and variable (usually an input variable) from MODEL-VAR command. If range information is not provided for a variable, then the variable is assumed valid over all values. See **RANGE Parameters** for more information on the parameters. Example: RANGE TEST1:V MIN=0.3 MAX=0.75

Table A-5. RANGE attribute definitions used within SIM-MODELDEF

RANGE Attributes	Attribute Descriptions
MAX	**{floating point value}** The model has the specified accuracy below this maximum value. The default is 1e10.
MAXEXT	**{floating point value}** Define how far the model can be extended above the MAX parameter in RANGE command on trigger conditions with negligible lose of accuracy. The default is 1.0.
MIN	**{floating point value}** The model has the specified accuracy above this minimum value. The default is –1e10.
MINEXT	**{floating point value}** Define how far the model can be extended below the MIN parameter in RANGE command on trigger conditions with negligible lose of accuracy. The default is 1.0.

Table A-6. SIM-MODELDEF parameter definitions

SIM-MODELDEF Parameters	PARAMETER Descriptions
LEVEL	**{(WHITE, PHYSICAL),(GRAY, DEVICE), (BLACK,FUNCTIONAL)}** Define a general level of modeling for the simulator model. The numbers define the level that is considered the most accuracy. Lower values indicate higher accuracy and usually less performance. The default is BLACK. WHITE or PHYSICAL –level 0 Model based on the physical structure using material characteristics. GRAY or DEVICE –level 1 Parameterized or lumped sum model based on observed or extracted physical behavior with modeling error. BLACK or FUNCTIONAL –level 2 A device model with highest performance and questionable accuracy to represent the base characteristics of the device.
MAXERROR	**{floating point value}** Define the worst case relative error for the model. This value is used if MAXERROR is not defined within RANGEDEF. The default is 100.
PERFORMANCE	**{floating point value}** Define the relative performance metric for the given model representation. Smaller values represent faster analysis models. The metric is presently the worst-case time for a simulator to iterate once with the given model. The default is 1e10.

Table A-7. MODELDEF parameter definitions

MODELDEF Parameter	PARAMETER Descriptions
DEFAULTSIMMOD	**{name}** Define the initialization SIMMOD if a conflict exists. Otherwise, performance and accuracy values in SIM-MODELDEF are used to determine the best model.

Table A-8. Control keywords for the backplane

Keyword Codes	CONTROL Description
GLOBPRM	Global parameter definitions that apply to ALL simulators. All global parameters are static. Format; GLOBPRM PARAMETERS The parameters are defined in GLOBPRM PARAMETERS.
OBJPRM	Interface control instructions to define how a node or object interacts with different simulators. Format: OBJPRM SIMULATOR1 OBJNAME PARAMETERS The parameters are defined in OBJPRM PARAMETERS.
SIMPRM	Define the simulator specific parameters. The parameters are STATIC unless otherwise specified. Format: SIMPRM SIMULATOR PARAMETERS The parameters are defined in SIMPRM PARAMETERS.
TOLERANCE	Define the tolerance for root objects between simulators to define when the variables have converged. See section 2.3.4. A reserved object and variable name is ALL where all objects and variables can be updated with one command. This command also defines the delta tolerance values for the sensitivity parameters. Format: TOLERANCE OBJNAME {variable} PARAMETERS PARAMETERS **ABSTOL={floating point value}** Define the absolute tolerance for a variable. See below. **RELTOL={floating point value}** Define the relative tolerance for a variable. See below. (continued)

Table A-8. Continued

<table>
<tr><th>Keyword Codes</th><th colspan="4">CONTROL Description</th></tr>
<tr><td rowspan="7">TOLERANCE
(continued)</td><td colspan="4">**ABSSENS={floating point value}**
Define the absolute sensitivity for a variable. See below

RELSENS={floating point value}
Define the relative sensitivity for a variable. See below.

RATIO={floating point value}
Define the scaling ratio for GRADCALC=RELPLUSMAX. See below.</td></tr>
<tr><td></td><td>**EFFORT variables**</td><td>**FLOW variables**</td><td>**Other variables**</td></tr>
<tr><td>**ABSTOL**</td><td>1.0e-4</td><td>1.0e-5</td><td>1.0e-4</td></tr>
<tr><td>**RELTOL**</td><td>0.0</td><td>1.0e-3</td><td>0.0</td></tr>
<tr><td>**ABSSENS**</td><td>1.0e-4</td><td>1.0e-5</td><td>1.0e-4</td></tr>
<tr><td>**RELSENS**</td><td>0.05</td><td>0.05</td><td>0.05</td></tr>
<tr><td>**RATIO**</td><td>0.5</td><td>0.5</td><td>0.5</td></tr>
</table>

Table A-9. GLOBPRM parameter definitions

GLOBPRM Parameters	PARAMETER Descriptions
COMM	**{FILE,SOCKET,PVM}** Define the communications protocol between backplane elements. The default is SOCKET. • FILE is for file-based transfers. This option is very slow (10 to 100 times) compared to the other options. • SOCKET is for UNIX based transfer mechanisms. • PVM is for the Parallel Virtual Machine mechanism (Not implemented).
DEADLOCKLIMIT	**{integer value}** Define the iteration limit for simulation process for a solution at the same reference point. The default is 50.
DEBUG	**{ON,OFF }** Output information from the backplane based on the DEBUG compile definitions. The default is OFF. The compile definitions are outlined in the include file "config.h".
MAXITR	**{long integer value}** Define the maximum number of iterations between backplane elements before the process is stopped. The default is 0, which means no limit.
PMAX	**{floating point number}** Define the maximum parameter for backplane interfaces and calculations. The default is 1e10.
PMIN	**{floating point number}** Define the minimum parameter for backplane interfaces and calculations. The default is 1e-10.
REFSTOP	**{integer value in hexadecimal format}** Define the stop reference point for the analysis. A solution is not generated at this point. The default is 0.
REFDELTAMIN	**{integer value in hexadecimal format}** Define the minimum reference data for the analysis. The backplane attempts to increase the reference delta until this minimum value is reached. The default is 1ns.
RESOLUTION	**{floating point number}** Define the minimum (usually time) resolution for the reference variables in the backplane. The default is 10 pico seconds.

Table A-9. Continued

GLOBPRM Parameters	PARAMETER Descriptions
RETAIN	**{integer value}** Specify the number of data points that each simulator must maintain for synchronization of the simulators. The default is 8.
SENS_DREF	**{SKIP,ROLLBACK,STOP, SCALE}** Define how the backplane reacts to a reference rollback during sensitivity calculations. The default is the SCALE. • SCALE option implements a linear scaling of the reference point to expected reference point. If the scale is greater than 2, then the process implements a SKIP. • ROLLBACK option forces the simulator to change the reference delta and the sensitivity analysis is restarted. • SKIP option skips the calculation of the sensitivity parameters affected by the rollback. • STOP option stops the sensitivity calculation process and returns to the previous operation.
SYNC	**{LOCKSTEP}** Define the synchronization method for the backplane. The default is LOCKSTEP. • LOCKSTEP forces all ITERATE mode simulators to keep the same REFPREV, REF, and REFDELTA.
TIMEOUT	**{integer value}** Define the timeout in seconds for the communications process. If the simulator has not received a communication in a given span, then the process checks for valid communication mechanism. The default is 120.
VARMAX	**{floating point number}** Define the maximum variable value allowed in interfaces and calculations where |variable| < |VARMAX|. The default is 1e3. This parameter is not checked during initialization.
VARINTERP	**{YES,NO}** Extrapolate new interface values after convergence for the next reference point based on previous values. The default is NO.

Table A-10. OBJPRM parameter definitions

OBJPRM Parameters	PARAMETER Descriptions
CAUSAL	**{EFFORT,FLOW}** Define which variable in the electrical interface is the input. The default is EFFORT. This parameter is **READONLY.**
CONF	**{Interface Configuration type. See Appendix 2}** This parameter defines the configuration or emulation mode of the interface. The default is DEFAULT or basic interface definition. The parameter is **DYNAMIC.**
DOMAIN	**{UNKNOWN,ELECTRICAL,MECHANICAL}** Define the domain that the object's variables. This parameter is **READONLY** and the default is **UNKNOWN.**
INTF	**{Interface type. See Appendix 2}** This parameter defines the electrical interface definition for an object. No default is defined. The parameter is **READONLY.**
MERGE	**{root object name}** Merge this object into the given root-object. The default is NULL (use object name as root object name).
STATUS	**{OFF, ON, TRACK}** Transmission or interface I/O control. This parameter defines how an object is connected to other objects in external simulators. The default is ON. • OFF. The information from the interface is not used by other simulators. • ON. The information is used by the other simulators and sensitivity parameters are calculated. • TRACK. The object follows another object and mirrors its response. This information is not used in the calculations to define the next solution and no sensitivity calculations are required.

Table A-10. Continued

OBJPRM Parameters	PARAMETER Descriptions
GRADCALC	**{RELTOL, RELPLUSMAX,ERROR}** Specify the variation parameters to the perturbation method for the sensitivity calculation process. The default is RELTOL, which is the conventional approach. • RELTOL applies a relative parameter times the variable value plus an absolute relative parameter. $delta = RELSENS * \|value\| + ABSSENS$ • RELPLUSMAX applies the RELTOL delta and adds the largest, previous applied delta as a ratio. $delta = ratio * RELTOL * \|value\| +$ $(1 - ratio) * DELTAMAX + ABSSENS$ • ERROR uses the present error information between the simulator value and the calculated value until the values become too small. NOTE: Not very reliable due to rounding errors. $delta = \|value(i) - value(i-1)\|$
GRADMODE	**{IDENTITY, DIAGONAL, FULL}** Specify how the backplane is to calculate the sensitivity information. Prior to the first valid solution, the FULL option is ALWAYS used. The default is FULL. • IDENTITY is used primarily for the correction process, Other modes maintains the present values. • DIAGONAL implements a SECANT based approach where sensitivity information is gathered during the analysis process. Variables are assumed decoupled. NOTE: New parameters are calculated if any delta variables are above the too-be-applied ideal delta defined by GRADCALC. Consequently, the sensitivity tolerance variables need to be reduced compared to the TOLERANCE defaults. Suggested tolerance parameters are RELSENS<1e-3 and ABSSENS<1e-5. • FULL option has the backplane perform the perturbation method where deltas are applied to each interface independently.

Table A-10. Continued

OBJPRM Parameters	PARAMETER Descriptions
GRADSIGN	**{M0,M1,M2,POS,NEG}** Define how the sign of the applied delta is determined for the sensitivity calculations of the object. The default is M1. • M0. Obsolete method of moving toward the calculated solution, which had an error but worked. • M1. This method determines the delta direction based on moving the simulator value toward the calculated value. On the first iteration of the new sequence, this routine uses the previous results to predict the direction of the solution. • M2. The delta direction is determined based on the stable or converged events in the object, so the delta is constant during the sequence. Until sufficient events are stored in the object, this process uses M1. • POS. Apply a positive value to the variable values. • NEG. Apply a negative value to the variable values.
GRADTIME	**{NONE, CONDITIONAL,ERROR, ITERATION, TIMESTEP}** Specify when the backplane is to initiate a gradient calculation. The default is TIMESTEP. • NONE maintains the present values. • TIMESTEP generates gradients on each new reference point after a variable change occurs. • CONDITIONAL implements the TIMESTEP process, but performs new sensitivity parameters when variables exceed the region encircled by the previously applied delta. • ERROR examines the error between the calculated values and the simulator values to determine when new sensitivity calculations are performed. • ITERATION generates new gradients after very iteration until variable convergence is reached.

Table A-10. Continued

OBJPRM Parameters	PARAMETER Descriptions
STATE	**{Causality states}** Defines the causality state of interface for the dynamic interface switching process depending on the STATE_ENABLE method. The different methods are defined in section 3.2 (method 1) and in section 3.4.5 (method 2). This parameter is **DYNAMIC.** For method 1, the default state is GENERIC, and the DEFCONF, CSCONF, and EFCONF parameters are used to redefine an object's CONF in the specific states. In method 2, the configurations are predefined. State decisions are made at the reference done level of the iteration process, after changes in the CAUSAL variable of an interface, and during initialization procedure. Causality states **Method 1 States** • GENERIC. Use the DEFCONF configuration. • CAUSE. Use the CSCONF configuration. • EFFORT. Use the EFCONF configuration. • GENERIC_TO_CAUSE. Transition state to CAUSE. • GENERIC_TO_EFFORT. Transition state to EFFORT. • CAUSE_TO_GENERIC. Transition state to GENERIC. • EFFECT_TO_GENERIC. Transition state to GENERIC. **Method 2 States** • SYSTEM_GENERIC. Use a SYSTEM configuration. • SYSTEM_LOCK. The interface is locked in a SYSTEM configuration. • CAUSE. Use a FSRC_SENS configuration to indicate that the system had entered the EFFECT or EFFECT_SET state. • EFFECT. Use the ESRC_SENS configuration. • EFFECT_SET. Use the ESRC_SENS configuration with the sensitivity parameter set to one.

Table A-10. Continued

OBJPRM Parameters	PARAMETER Descriptions
STATE_ENABLE	**{METHOD1,METHOD2, NO}** Enable the given object to implement a specific causality method. The default is NO. • METHOD1. Use the causality detection criteria proposed in section 3.2. • METHOD2. Use the improved causality detection criteria from section 3.4.5. • NO. Disable the causality detection process.
CAUSCRTR	**{ POWER,SENSITIVITY}** Define the causality detection criteria. The default is SENSITIVITY (See section 3.2.1). • POWER uses the power criteria. • SENSITIVITY uses the sensitivity criteria.
CAUSRATIO	**{floating point number}** Define the cause ratio for the causality detection process. The default is 10.
CSCONF	**{Interface type. See Appendix 2}** Define the CAUSE configuration emulation if the element is identified as CAUSE.
DEFCONF	**{Interface type. See Appendix 2}** Define the default or GENERIC emulation configuration.
EFCONF	**{Interface type. See Appendix 2}** Define the EFFECT configuration if the element is identified as an EFFECT.

Table A-11. SIMPRM parameter definitions

SIMPRM Parameters	PARAMETER Descriptions
BACKUP	**{NONE,MEMORY,FILE,MEM_FILE}** Parameter from the simulator to the backplane that defines how the simulator can do a backup for a LOAD and SAVE operation. The parameter is **READONLY**. The default is NONE. • NONE. No backup capability. • MEMORY. Store information in the computer memory. • FILE. Store information in a file. • MEM_FILE. Both MEMORY and FILE operations are available.
BACKUPCONF	**{ NONE,MEMORY,FILE}** Defines how the simulator will do a backup during a LOAD and SAVE operation. The default is NONE. • NONE. No backup capability. • MEMORY. Store information in the computer memory. • FILE. Store information in a file.
BACKUPNAME	**{name}** Name of the backup. A name must be specified if the LOAD or SAVE mode is performed.
COMMTYPE	**{See COMM types }** Define how the simulator is used in the communications scheme. The default is CLIENT. COMM types • SERVER. The element receives all information and then broadcasts it to the clients. • CLIENT. The element sends and receives information only from the server. • HYBRID. This element is a client and a server (Not fully implemented).

Table A-11. Continued

SIMPRM Parameters	PARAMETER Descriptions
CTRLTYPE	**{See CTRL types }** Define how the simulator is used in the control process. The default is SIMULATOR. CTRL types • CONTROLLER. This element controls other elements and implements the configuration rules. Only one CONTROLLER can be defined within the backplane. • SIMULATOR. This element performs calculations and controls a simulator. • SIM+CONTROL. This element implements a CONTROLLER process and SIMULATOR process simultaneously (Not fully implemented). • OBSERVER. This element only examines data.
DEACTIVE_MODE	**{OFF, UPTODATE, TRACK}** Define the mode of the simulator if all internal objects have STATUS=OFF. The default is UPTODATE.
END	**{See END options}** This output parameter defines the status of the simulator. This parameter is **DYNAMIC**, and is typically defined by the backplane in the simulator. The default is NO. • NO. The simulator has not finished the present iteration. • YES. The simulator has finished the present iteration. • DONE. The simulator has variable convergence with other simulators, so the iteration process can move to the next iteration and reference point. • STOP. The simulator has finished the present task or had an error. A simulator may wait for other simulators completion before continuing. NOTE Backplane elements redefine other simulator's END parameters (to a NO) after iteration sequence completion in order to eliminate transmissions between elements.
IONAME	**{name}** Input and Output name for communications. The default is BKPLCON.
LOGFILE	**{name}** Log file containing the iteration characteristics of the simulator and object interface parameters.

Table A-11. Continued

SIMPRM Parameters	PARAMETER Descriptions
MATRIX	**{ON,OFF}** Parameter from the simulator to the backplane defining that the simulator has a matrix loading capability. The default is OFF. This parameter is **READONLY**.
MARK	**{long integer value}** Iteration marker value for debugging the synchronization process between backplane elements. This parameter is **DYNAMIC**.
MODE	**{See MODE parameters }** This parameter defines the operation to be done on a START command, where MODEPRS is set to MODE. The default is ITERROGATE.
MODEPRS	**{See MODE parameters }** Define the present operational mode of a simulator. The default is UNKNOWN. The start command transfers the MODE definition to the MODEPRS. This parameter is **CTRLONLY**.
MODE parameters	<u>Backplane non-calculation modes</u> • IDLE or OFF. Waiting mode for information and commands. • ITERROGATE. Debugging and configuration mode. This mode provides feedback to the external simulator about all facets of the local simulator. • OVERRIDE. Override control mode for redefining non-CTRLONLY backplane variables. • SHUTDOWN. shutdown mode. An orderly shutdown of the simulator is to be performed. <u>Simulator interaction modes without calculations</u> • LOAD. A recovery mode. The simulator is instructed to a load a file (IOFILE) or rollback to a previous internal state defined by REFROLLBACK and BACKUPCONF. This mode is used in conjunction with SYNCHRONIZE and SAVE. • SAVE. A backup mode. The simulator is instructed to a save a file (IOFILE) or internal contents for recovery, synchronization, or rollback. The destination of the save is defined by BACKUPCONF. This mode is used in conjunction with SYNCHRONIZE and LOAD.

Table A-11. Continued

SIMPRM Parameters	PARAMETER Descriptions
MODE parameters (continued)	Simulator interaction modes with non-iterative calculations • INDEPENDENT. Backplane-off mode. The simulator has minimum interaction with the backplane except to load stimuli. • SYNCHRONIZE. Synchronize mode. The backplane defines an event sequence to synchronization the operation of the local simulator with external operations. Concurrent interaction modes between simulators • ITERATE. Concurrent simulation mode. • SENSCALC. Internal calculation mode for sensitivity calculations. • TRACK. Synchronize mode. The objects within the simulator track the response of objects in other simulator. Track simulators are iteration locked at the reference done level. • UPTODATE. Synchronize mode. Similar to TRACK, this mode is not iteration lock with the other simulators. The simulator *tries* to remain up-to-date with the other simulators like the SYNCHRONIZE mode.
REF	**{Hexadecimal integer value}** Parameter defining the present simulation reference point from the analysis. This parameter is **READONLY**. The default is 0.
REFDELTA	**{Hexadecimal integer value}** Parameter defining the next delta reference point from the analysis for step control. This parameter is **READONLY**. The default is 0.
REFPREV	**{Hexadecimal integer value}** Parameter defining the previous simulation reference point from the analysis. Specifically, this parameter is the known stable solution that all calculations are referred. This parameter is **READONLY**. The default is 0.

Table A-11. Continued

SIMRM Parameters	PARAMETER Descriptions
REFROLLBACK	**{integer value in hexadecimal format}** Define the rollback reference point for the analysis for a load operation. The simulator should use this parameter to shift its reference points of the loaded information to this point for fast synchronization. The default is 0.
SENSABSTOL	**{floating point value}** The relative absolute tolerance for the sensitivity calculations performed by a simulator. The default is 1e-8.
SENSRELTOL	**{floating point value}** The relative tolerance for the sensitivity calculations performed by a simulator. The default is 1e-3.
SERVER	**{simulator ID}** Name or ID of the backplane server. The default is BKPLCON.
SIMDEF	**{name}** Name of the simulator being reference by the backplane. This parameter must be defined uniquely for each simulator. This parameter is **READONLY**.

Table A-12. Command keywords for the backplane

Keyword Codes	COMMAND Description
ID	Assign the simulator ID that all simulators will use as reference. This process can only be done once, and each simulator is given a unique ID prior to the analysis. Format: ID name
INCLUDE	Define a control file for the backplane. Format: INCLUDE file
REQSTAT	Request status of a simulator and any internal backplane components. This command is similar to the ITERROGATE mode, but the command can be issued during any mode. This command is more powerful than the ITERROGATE because any internal parameters or data from any simulator can be examined within the backplane elements of the given simulator. This command is intended for diagnostics. The information is returned using the MESSAGE DATA format. Three formats are available. Format(1): REQSTAT Get the status of all simulators. All local OBJPRM and SIMPRM are returned. Format(2): REQSTAT SIMULATOR Get the status of a particular simulator. All local OBJPRM and SIMPRM are returned. Format(3): REQSTAT SIMULATOR PARAMETERS Get the status of the local simulator parameters or other simulator's information in the simulator beyond OBJPRM and SIMPRM. PARAMETERS **DATA** Returns the event data for the given object. **GLOBPRM={Parameter name}{ALL}** Returns the given global parameter. The ALL parameter returns all parameters. (Continued)

Table A-12. Continued

Keyword Codes	COMMAND Description
REQSTAT (continued)	**OBJ={Object name}** Define the object for the REQSTAT process. If no object is defined, then all objects for the given simulator are returned. **OBJPRM={parameter name}{ALL}** Returns the object parameters for the specified object. The ALL parameter returns all parameters. **PING** A simple mechanism to make certain that a backplane element is alive. The MESSAGE data response is sent to the backplane controller containing SIMPRM simulator END={status}. **SENSITIVITY** Return the sensitivity information for the given simulator. **SIM={Simulator name}** Define the simulator. The default is the local simulator. **SIMPRM={parameter name}{ALL}** Returns the given simulator parameter for the specified simulator. The ALL parameter returns all parameters.
SIMCMD	Command instruction from the backplane to a specific simulator. Format: SIMCMD SIMULATOR COMMAND
START	All simulators are to start processing data based on the MODE definition. All MODE parameters are transfer to MODEPRS. Format: START
STOP	All simulators are to stop and await further instructions. The present MODEPRS of each simulator is set to an IDLE. Format: STOP Note: Certain modes do not process information until the given task is completed, so the STOP command could be ignored temporarily.

Table A-12. Continued

Keyword Codes	COMMAND Description
TRIGDEF	Define the trigger conditions for switching between different models of a device (DEVICE). Format: TRIGDEF NAME DEVICE PARAMETERS See **TRIGDEF Parameters** for PARAMETERS information.

Table A-13. TRIGDEF parameter definitions

TRIGDEF Parameters	PARAMETER Descriptions
MAXERROR	**{floating value}** The maximum error allowed for the model. The default is 0.01 or 1% error
ABSTOL_SCALE	**{floating value}** Define a scale factor for the normal variable absolute tolerance, so the trigger does not happen because of noise. The default is 10.0.
OBJ	**{string value}** Define an object within a simulator. The default is NULL.
PERFDIFF	**{floating point number}** Define the performance improvement between the models to cause a trigger condition. The default is 0.1 (10 percent).
RANGEDIFF	**{floating point number}** Define the range and accuracy improvement between the models to cause a trigger condition. The default is 0.1 (10 percent).
REF	**{Hexadecimal integer value}** Define a trigger reference point to switch to another model. The default is 0.
SEQCNT_BACK	**{integer value}** Define the number of sequential points needed by the new model under different constraint before the new model is replaced by the old model. The default is 2.
SEQCNT_SWITCH	**{integer value}** Define the number of sequential points needed by the new model before the present model can be replaced by the new model. The default is 3.

Table A-13. Continued

TRIGDEF Parameters	PARAMETER Descriptions
SIM	**{string value}** Define the simulator that contains the comparison variable. The default is NULL.
SIMMOD	**{simulator model}** Define the simulator model to replace the existing model. The default is NULL
TRIGCOND	**{COMPARE, IMMEDIATE, OFF, RANGE, REF, VAR}** Define the trigger condition. The default is OFF. See Table A-13 for more information on each trigger condition
VARCOND	**{GT, GTE, EQ, LTE, LT}** Define the variable condition for the generating a trigger. The default is EQ.
	GT Greater Than VARVALUE.
	GTE Greater Than or Equal to VARVALUE.
	EQ Equal to VARVALUE (within tolerance limits).
	LTE Less Than or Equal to VARVALUE.
	LT Less Than VARVALUE.
VARVALUE	**{floating point value}** Define the comparison value used by VARCOND.

Table A-14. An outline of the trigger conditions within the trigger definition command

TRIGCOND	PARAMETER Descriptions
ACCURACY	Switch models based on an accuracy comparison with a reference device. This condition uses the following TRIGDEF parameters: • MAXERROR. • PERFDIFF. • SEQCNT_SWITCH. • SEQCNT_BACK. • SIMMOD. This parameter now defines the reference model that other models are compared. The accuracy switching rules are defined in Appendix-A.5.
IMMEDIATE	Switch to SIMMOD immediately. This condition uses only the SIMMOD parameter.
OFF	The trigger has been completed by the dynamic switching process or the trigger has been disabled.
RANGE	Use the RANGEDEF information in the model definition to determine the switching conditions. This condition uses the following TRIGDEF parameters: • MAXERROR. • PERFDIFF. • RANGEDIFF. • SEQCNT_SWITCH. • SEQCNT_BACK. The range switching rules are defined in Appendix-A.5.
REF	Switch to the given SIMMOD when the trigger REF point is reached or exceeded. This condition uses the following TRIGDEF parameters: • REF. • SIMMOD.
VAR	Trigger on a specific variable condition. This condition uses the following TRIGDEF parameters: • VAR. • OBJ. • SIM. • SIMMOD. • VARCOND. • VARVALUE.

APPENDIX-B

Appendix-B. Interface Methodology

This section defines the interface types known to the backplane and the basic structure of the backplane interface into the simulator. The interface process has an interface level and a configuration level. The interface level defines the primitive structure of the interface while the configuration level defines the calculation or solving procedures beyond the primitive level. A configuration cannot be used as an interface while an interface is also a configuration. A definition for the different interfaces and configuration is provided in Table B-1. Table B-2 defines the configurations for the different interfaces.

All interface structures between the backplane and the simulator are based on bond graph techniques. An interface has the variable input/output characteristics defined in Table B-3. The interfaces are domain specific, but all domain variables are defined as an EFFORT and a FLOW. The EFFORT and FLOW designations define how variables between simulator interact. Three domains are presently defined:

- Signal Domain (Section B-1).
- Electrical domain (Section B-2).
- Mechanical domain (Section B-3).

The interface component values for the different configurations of the primitive interfaces are defined in Table B-4 in terms of the calculated FLOW and EFFORT variables. The sensitivity delta calculation is defined in Table B-5, where Δ is the variable delta function. Several integer index variables are also provided for internal simulator operations. Additional variables and domains can be easily added to the backplane. The final section describes the initialization of the different interfaces.

Table B-1. List of all interface and configurations recognized by the backplane

Interfaces and Configurations	**Descriptions**
CORRECTOR	This configuration is used in the flow correction process with a PREDICTOR.
CORRECTOR_SENS	This configuration is a variation of the CORRECTOR configuration with a sensitivity element to improve convergence by including loading effects.
EAPPLY	This configuration applies a list of effort data.
EDATA	This interface defines the effort data applied by an EAPPLY.
EFFORTEQ	This configuration is a direct interface that follows the CAUSE element in the root-level object.
EMONITOR	An effort monitor configuration.
ESRC	This interface defines the effort variable.
ESRC_FSRC_2SENS	This flexible interface can mimic any type of interface (FLEXIBLE).
ESRC_SENS	This interface uses diagonal component of the sensitivity information with an ESRC, so the simulator can emulate loading effects and provide local feedback to the backplane (THEVININ).
FAPPLY	This configuration applies a list of flow data.
FDATA	This interface defines the flow data applied by an FAPPLY.
FLOWSUM	This configuration is a direct interface that is the CAUSE element in the root-level object. This interface defines E'_{CAUSE} to the EFFORTEQ interfaces in a sequential process.
FMONITOR	A flow monitor configuration.
FSRC	This interface defines the flow variable.
FSRC_SENS	This interface uses diagonal component of the sensitivity information with an FSRC, so the simulator can emulate loading effects and provide local feedback to the backplane (NORTON). Only this interface uses the GC component.
PREDICTOR	This configuration specifies that a simulator has an internal model for the device. A CORRECTOR type configuration is required to complete this configuration. However, the interface can operate in an independent mode without a corrector.
SYSTEM	This configuration defines coupling to other elements using the equivalent matrix G^{i}_{EQ} through a simulator's matrix loading routines.

Table B-2. Interface and configuration compatibility index

Configurations	Interfaces						
	A	B	C	D	E	F	G
CORRECTOR		X	X	X			
CORRECTOR_SENS			X	X			
EAPPLY		X	X	X			
EDATA	X						
EFFORTEQ		X	X	X			
EMONITOR				X	X	X	
ESRC		X	X	X			
ESRC_FSRC_2SENS				X			
ESRC_SENS			X	X			
FAPPLY				X	X	X	
FDATA							X
FLOWSUM				X	X	X	
FMONITOR				X			
FSRC				X	X	X	
FSRC_SENS				X	X		
PREDICTOR				X			
SYSTEM				X			

X indicates a valid configuration of the interface. Interface A corresponds to EDATA, B to ESRC, C to ESRC_SENS, D to ESRC_FSRC_2SENS, E to FSRC_SENS, F to FSRC, and G to FDATA.

Table B-3. Input/ Output characteristics of the backplane interface

Variable Name	Description
E_{IN} or E_{SRC}	Effort input to the simulator.
F_{IN} or F_{SRC}	Flow input to the simulator.
G_{IN1} or G_{SRC}	Sensitivity parameter input to the simulator.
G_{IN2} or G_{C}	Secondary sensitivity parameter input to the simulator.
E_{OUT} or E_{FSRC}	Effort output from the simulator.
F_{OUT} or F_{ESRC}	Flow output from the simulator.
E_{INDEX}	Local simulator index for E variable solution.
F_{INDEX}	Local simulator index for F variable solution.
$ESYS_{INDEX}$	Local simulator index for loading an external matrix.

Table B-4. Calculations definitions for the different configurations and interfaces

Interface Type	**Effort Value** (E_{SRC}^{i+1})	**Flow Value** (F_{SRC}^{i+1})	**Admittance Value** (G_C^{i+1}or G_{SRC}^{i+1})
CORRECTOR	E_{PRED}^{i} or E^{i+1}	0.0	PMAX
CORRECTOR_SENS	E_{PRED}^{i} or E^{i+1} $-G_{EQ}^{i}\big\|_{j,j}^{-1} F^{i+1}$	0.0	$G_{EQ}^{i}\big\|_{j,j}$
EAPPLY	$E_{DATA}(ref)$	0.0	PMAX
EDATA	$E_{DATA}(ref)$	-	-
EFFORTEQ	E_{CAUSE}^{i}	0.0	PMAX
EMONITOR	0.0	0.0	PMIN
ESRC	E^{i+1}	0.0	PMAX
ESRC_FSRC_2SENS	0.0	$F^{i+1} - G_{EQ}^{i}\big\|_{j,j} \cdot E^{i+1}$	$G_{EQ}^{i}\big\|_{j,j}$
ESRC_SENS	$E^{i+1} - G_{EQ}^{i}\big\|_{j,j}^{-1} F^{i+1}$	0.0	$G_{EQ}^{i}\big\|_{j,j}$
FAPPLY	0.0	$F_{DATA}(ref)$	PMIN
FDATA	-	$F_{DATA}(ref)$	-
FLOWSUM	0.0	$-\sum_{j=2}^{j=n_T} F_j^{i}$	
FMONITOR	0.0	0.0	PMIN
FSRC	0.0	F^{i+1}	PMIN
FSRC_SENS	0.0	$F^{i+1} - G_{EQ}^{i}\big\|_{j,j} \cdot E^{i+1}$	$G_{EQ}^{i}\big\|_{j,j}$
PREDICTOR	0.0	F_C^{i+1}	PMIN
SYSTEM	0.0	$F^{i+1} - G_{EQ}^{i} \cdot E^{i+1}$	PMIN. G_{EQ}^{i} defines the load matrix for the simulator

Table B-5. Sensitivity delta definitions for the different configurations and interfaces

Interface Type	**Effort Delta** (ΔE_{SRC}^{i+1})	**Flow Delta** (ΔF_{SRC}^{i+1})
CORRECTOR	ΔE	0
CORRECTOR_SENS	$0.71 \cdot \Delta E$	$-0.71 \cdot \lvert\Delta F\rvert \cdot sign(\Delta E \cdot G_{src})$
EAPPLY	-	-
EDATA	-	-
EFFORTEQ	-	-
EMONITOR	-	-
ESRC	ΔE	0
ESRC_FSRC_2SENS	$0.71 \cdot \Delta E$	$-0.71 \cdot \lvert\Delta F\rvert \cdot sign(\Delta E \cdot G_{src})$
ESRC_SENS	$0.71 \cdot \Delta E$	$-0.71 \cdot \lvert\Delta F\rvert \cdot sign(\Delta E \cdot G_{src})$
FAPPLY	-	-
FDATA	-	-
FLOWSUM	-	-
FMONITOR	-	-
FSRC	0	ΔF
FSRC_SENS	$0.71 \cdot \Delta E$	$-0.71 \cdot \lvert\Delta F\rvert \cdot sign(\Delta E \cdot G_{src})$
PREDICTOR	0	$MAX(\Delta F, \Delta F_C)$
SYSTEM	$0.71 \cdot \Delta E$	$-0.71 \cdot \lvert\Delta F\rvert \cdot sign(\Delta E \cdot G_{src})$

Appendix-B.1 Signal Domain

The signal domain has two interfaces: input and output. The input and output variables are equivalent to an EFFORT, where an input loads the E_{IN} value and output writes the E_{OUT} value. In the signal domain, no power is exchanged due to abstraction of the model representations. The input and output are for control system representations like Laplace and the Z-domain analysis.

Appendix-B.2 Electrical Domain

In the electrical domain, the two major variables are current (Flow) and voltage (Effort). This domain maps directly into the interface structure, which is shown in Figure 3-2. An interface translator from the electrical to primitive references is defined in Table B-6. To keep the interfacing structure simple, only the flexible interface was allowed to have a system-level or matrix loading capability.

Table B-6. Interface mapping between primitive elements and electric interfaces

Interface Type	Electrical Equivalent or alias	Interface Type	Electrical Equivalent or alias
EAPPLY	VAPPLY	FAPPLY	IAPPLY
EMONITOR	VMONITOR	FMONITOR	IMONITOR
ESRC	VSRC	FSRC	ISRC
ESRC_SENS	THEVININ	FSRC_SENS	NORTON
ESRC_FSRC_2SENS	FLEXIBLE		

Appendix-B.3 Mechanical Domain

This research concentrated on electrical properties, so the different configurations for the mechanical domain were not developed, but the primitive structures can still be used. However, the interface problem is more complex in the mechanical domain. The electrical domain had only two variables, while the mechanical domain can have multiple degrees of freedom. Consequently, the bond graphic approach could require multiple sub-domains to define all interfaces between mechanical simulators. Specifically, an X, Y, Z, and thermal domains would be expected in the full implementation. Since this research did not require these domain variables, full domain decomposition was not implemented. Only the mechanical stimuli information was required for the behavioral models in the electrical simulators to guarantee consistency between applied stimuli in both domains.

Appendix-B.4 Interface Initialization

The initialization of the interface was critical to eliminate divergence. Two levels of initialization were required: simulator and backplane. The simulator initialization of the interfaces was required to prevent the interface from dominating the first response prior to interaction with other simulators. The backplane initialization was a multiple calculation process that had the same purpose of preventing divergence in the backplane calculations. Both initialization processes were critical for preventing divergence and finding the "reasonable close" initial guess. For the simulator initialization, the ESRC, FSRC, and EFSRC values of the interface were zeroed and the sensitivity GSRC and GC values were typically 1e-6. The PREDICTOR configuration required GSRC and GC to be PMIN (1e-10). The internal conduction matrix that a simulator can load is zeroed.

There is a three-step setup sequence in the backplane. During this setup sequence, the CORRECTOR type configurations were forced to track the EFFORT variables of the PREDICTOR configurations. On the first step, all EFFORT and FLOW values are zeroed and applied back to the interfaces. The system configurations are given a nonzero conduction matrix. The other interface with sensitivity components will able to use other simulator's sensitivity information to define loading effects. On the second and third steps, the calculated EFFORT value was defined as an effort average, so all EFFORT variables were moved toward a common value. However, interfaces in other simulators were required to track any EFFORT output interfaces in the root node of the same domain. The calculated FLOW variables were set to zero. The SYSTEM configuration followed a different procedure, which set the EFFORT variables to the initial simulator's values. In addition, the coupling effects in the equivalent matrix for the SYSTEM configured interfaces were modified to make certain that coupling existed between all interfaces in the simulator. A new equivalent matrix was always calculated after the initialization sequence.

APPENDIX-C

Appendix-C. Interface Variable Causality Detection

The variable causality defined the sensitivity parameters that were required for the coupling process by identifying the input variable (EFFORT or FLOW) of an interface. For this process, the backplane created a matrix of coupling results between all interface variables. Initially, four different conditions were checked per variable where applicable:

COND1: If variable variation at A had no affect on variable B where $\alpha|\Delta A| > |\Delta B|$ and $0 < \alpha < 1$, then the variables are isolated. The value for α was 0.001.

COND2: If a variable variation at A had an affect on variable B, but a variable variation at B had no affect on A, then the sensitivity parameter for variable B to A ($\frac{\partial A}{\partial B}$) was forward (unidirectional) coupled. Otherwise, the variables were bi-directional coupled. (OBSELETE)

COND3: Variable variation at A has gain to variable B if $|\Delta B| > \alpha|\Delta A|$. The parameter α=10 guaranteed gain immediately. For small gain situations, a sequential gain check was performed. If the condition with α=2 is sequentially detected, then the variable is also considered to have gain. This sequential gain check prevented false gain detected due to poor simulator and backplane interaction.

COND4: If the summation of the flow variables of the sensitivity matrix with respect to A variation was below tolerance, the zero-flow summation test was passed. This test determined if the flow into simulator was equivalent to the flow out of the simulator. (OBSELETE)

These tests were used to derive seven coupling conditions outlined in Table C-1.

The gain relationships were the most critical effects to identify causality. Without an EFFORT gain, the causality identification process was defaulted to the EFFORT input condition. All interfaces were initialized to the EFFORT-input configuration. The causality identification process also used an interface's causality characteristics whenever possible. For example, a flow source did not have an EFFORT-input configuration and most

Table C-1. Variables created for identifying an interface's input causality

Parameter	Description
E_GAIN_CNT	The number of effort gains for an interface.
F_GAIN_CNT	The number of flow gains for an interface.
EFFORT_ENABLE	The interface can support effort as the input.
FLOW_ENABLE	The interface can support flow as the input.
GLOBAL_ISOL	The local effort-variable delta did not couple to other interface variables. All COND1 conditions were true with respect to the local effort variable.
LOCAL_ISOL	The local effort variable was isolated from the global variable deltas. All COND1 conditions were true with respect to all other variables.
EXTERNAL_FLOW	An external interface has already changed to flow as the input.
LOCAL_VAR	Defines if a particular variable is dominating the local interface causality. The values are NONE, FLOW, AND EFFORT. NONE means that both variables are reacting to the applied delta. FLOW means that the flow variable is isolated from the effort variable. EFFORT means that the effort variable is isolated from the flow variable.

sensitivity parameters were for the FLOW-input. As an EFFORT-input interface, the number of sensitivity parameters for the flow source was limited. If the backplane ever detected flow-input condition for the interface, then the process stayed in the flow-input configuration regardless of the detected conditions. This consideration maximized the number of sensitivity parameters that could be calculated.

In the final version of the causality identification process, several tests became unnecessary. COND4 was initially considered for identifying devices within a simulator, but COND1 provided the same information in a more generic format. COND2 was briefly used to redefine (zero) delta information in the perturbation matrix. However, switching causality from EFFORT to FLOW and hiding the inactive parameters corrected this problem. For identifying the input variable of an interface, the final criteria in Figure C-1 used only COND1 and COND3. For future development, the COND1 information was considered for identifying isolation conditions. With isolation identified, different calculation methods like GRADMODE=DIAGONAL could be automatically detected. Besides the GRADMODE option, multiple deltas could be applied simultaneously during the sensitivity calculation to eliminate calculation sequences between the backplane and the simulator.

In this detection process, only one FLOW input variable was allowed per root node domain. The EXTERNAL_FLOW variable blocked other interfaces from entering the same mode. For this reason, the causality identification process could not falsely identify FLOW inputs, so the process was conservative in identifying the FLOW inputs and liberal in identifying the EFFORT inputs. This rule followed the standard input and output

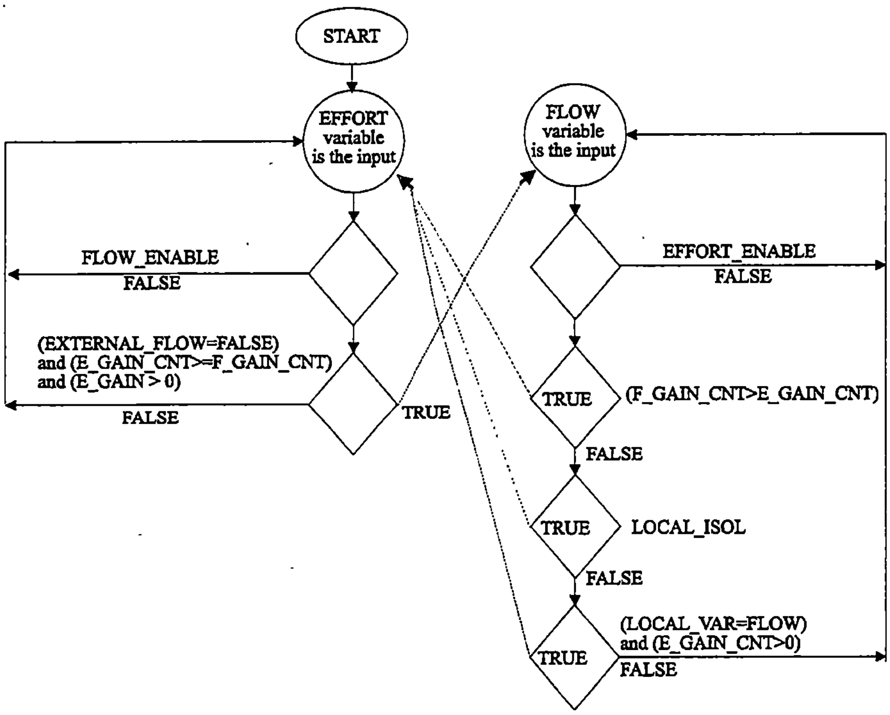

START
EFFORT variable is the input
FLOW variable is the input
FLOW_ENABLE
FALSE
(EXTERNAL_FLOW=FALSE) and (E_GAIN_CNT>=F_GAIN_CNT) and (E_GAIN> 0)
FALSE
TRUE
EFFORT_ENABLE
FALSE
TRUE
(F_GAIN_CNT>E_GAIN_CNT)
FALSE
TRUE
LOCAL_ISOL
FALSE
(LOCAL_VAR=FLOW) and (E_GAIN_CNT>0)
TRUE
FALSE

Figure C-1. Input causality detection process restrictions of only one output per node to avoid drive conflicts and poor matrix definitions in Equation 3-24. However, this rule was unnecessary if the $\frac{\partial E_X}{\partial F_X}$ parameter of the given interface was not zero. A simple limiting condition on this parameter during the sensitivity calculations eliminated the problem, and the backplane could have solved examples with multiple EFFORT outputs in the same node.

APPENDIX-D

Appendix-D. Coupling Results

This appendix contains the results from the interfacing tests over five different categories of sensitivity calculation and scheduling parameters:

- Iterative Scheduling.
- Conditional Scheduling.
- Error Scheduling.
- Timestep Scheduling.
- Other parameters. The diagonal option is explored in addition to the identity correction process. For the diagonal options, the sensitivity tolerance parameters were defined as ABSSENS=1e-8 and RELSENS=0.001.

For the first five categories, four sub-categories are defined:

- RELTOL- Parameter setting of GRADCALC=RELTOL.
- RELTOL with extrapolation- Parameter setting of GRADCALC=RELTOL and interface values are approximated via finite differences at a new reference point.
- MAXTOL- Parameter setting of GRADCALC=RELMAXTOL.
- MAXTOL with extrapolation- Parameter setting of GRADCALC=RELMAXTOL and interface values are approximated via finite differences at a new reference point.

In total, the 76 different configurations in Table D-1 are tested over these different categories as applicable.

In these examples, the CIR0 partition was typically the bounding (driving) component in the coupling process, while the CIR0 partition represented a device. Most interfaces used the default backplane-tolerance settings, but the flow tolerance for the HF (high-gain feedback problems) required ABSTOL to 1e-6 for proper accuracy. The electrical-to-electrical coupling examples are defined by E2E and the electrical-to-FE examples are defined by E2FE. In addition, the E2FE examples are only tested using the GRADTIME=CONDITIONAL because of the long simulation times.

There are 14 examples for E2FE and 31 examples for E2E. Most examples are simple problems with varying degrees of coupling complexity to check the backplane processes. For the examples, the primitive components used in the E2E and E2FE examples are in Figure D-1 and D-2. Figure D-2 also defines the SPICE equivalent behavioral models for the FE structures. The E2FE examples have a piezoelectric test (PE) and a piezoresistive test (PR1) using the same meshed structure. The E2E examples and iteration results are shown in Figure D-3 through D-47. The E2FE examples and iteration results are in Figure D-48 through D-63. All iteration results are presented as a ratio of the given interface iteration result (X_i) to the minimum iteration result (X_{MIN}) minus one, or

$$\frac{X_i}{X_{MIN}} - 1 \qquad \text{Equation IV-1}$$

The flow correction process defines the correctors as having a specified error compared to the base model. For the E2E examples, this percent was very precise since the models were scaled relative to a know base. The E2FE examples followed the same procedure, but the base predictor model in the SPICE was NOT a scaled version of the FESIM model. Consequently, the SPICE predictor model in the E2FE tests had larger errors than the E2E tests.

Table D-1. Interface numbers to interface definition table

Conventional Interface Combinations

Interface Number	Partition CIR0 Interface	Partition CIR1 Interface	Interface Number	Partition CIR0 Interface	Partition CIR1 Interface
1	FSRC	FSRC	19	FSRC_SENS	FSRC
2	FSRC	ESRC	20	FSRC_SENS	ESRC
3	FSRC	ESRC_SENS	21	FSRC_SENS	ESRC_SENS
4	FSRC	FSRC_SENS	22	FSRC_SENS	FSRC_SENS
5	FSRC	FLEXIBLE	23	FSRC_SENS	FLEXIBLE
6	FSRC	SYSTEM	24	FSRC_SENS	SYSTEM
7	ESRC	FSRC	25	FLEXIBLE	FSRC
8	ESRC	ESRC	26	FLEXIBLE	ESRC
9	ESRC	ESRC_SENS	27	FLEXIBLE	ESRC_SENS
10	ESRC	FSRC_SENS	28	FLEXIBLE	FSRC_SENS
11	ESRC	FLEXIBLE	29	FLEXIBLE	FLEXIBLE
12	ESRC	SYSTEM	30	FLEXIBLE	SYSTEM
13	ESRC_SENS	FSRC	31	SYSTEM	FSRC
14	ESRC_SENS	ESRC	32	SYSTEM	ESRC
15	ESRC_SENS	ESRC_SENS	33	SYSTEM	ESRC_SENS
16	ESRC_SENS	FSRC_SENS	34	SYSTEM	FSRC_SENS
17	ESRC_SENS	FLEXIBLE	35	SYSTEM	FLEXIBLE
18	ESRC_SENS	SYSTEM	36	SYSTEM	SYSTEM

Table D-1. Continued

FLEXIBLE interface with causality detection

Interface Number	Cause Type	Default Type	Effort Type
37	FSRC	FLEXIBLE	ESRC
38	FSRC	FLEXIBLE	ESRC_SENS
39	FSRC	FLEXIBLE	FSRC_SENS
40	FSRC_SENS	FLEXIBLE	ESRC_SENS
41	FSRC_SENS	FLEXIBLE	ESRC
42	SYSTEM	SYSTEM	ESRC_SENS
43	SYSTEM	SYSTEM	ESRC
44	Method 2 – Tiered implementation		

Table D-1. Continued

Different Predictor and ESRC configurations

Interface Number	Predictor and Coupling Characteristic	Interface Number	Predictor and Coupling Characteristic
45	Model with +10 Error and ESRC interface	61	Model with +10 Error and ESRC_SENS interface
46	Model with -10 Error and ESRC interface	62	Model with -10 Error and ESRC_SENS interface
47	Model with +20 Error and ESRC interface	63	Model with +20 Error and ESRC_SENS interface
48	Model with –20 Error and ESRC interface	64	Model with –20 Error and ESRC_SENS interface
49	Model with +50 Error and ESRC interface	65	Model with +50 Error and ESRC_SENS interface
50	Model with –50 Error and ESRC interface	66	Model with –50 Error and ESRC_SENS interface
51	Model with +90 Error and ESRC interface	67	Model with +90 Error and ESRC_SENS interface
52	Model with -90 Error and ESRC interface	68	Model with -90 Error and ESRC_SENS interface
53	Model with +10 Error and sequential ESRC interface	69	Model with +10 Error and sequential ESRC_SENS interface
54	Model with -10 Error and sequential ESRC interface	70	Model with -10 Error and sequential ESRC_SENS interface
55	Model with +20 Error and sequential ESRC interface	71	Model with +20 Error and sequential ESRC_SENS interface
56	Model with –20 Error and sequential ESRC interface	72	Model with –20 Error and sequential ESRC_SENS interface
57	Model with +50 Error and sequential ESRC interface	73	Model with +50 Error and sequential ESRC_SENS interface
58	Model with –50 Error and sequential ESRC interface	74	Model with –50 Error and sequential ESRC_SENS interface
59	Model with +90 Error and sequential ESRC interface	75	Model with +90 Error and sequential ESRC_SENS interface
60	Model with -90 Error and sequential ESRC interface	76	Model with -90 Error and sequential ESRC_SENS interface

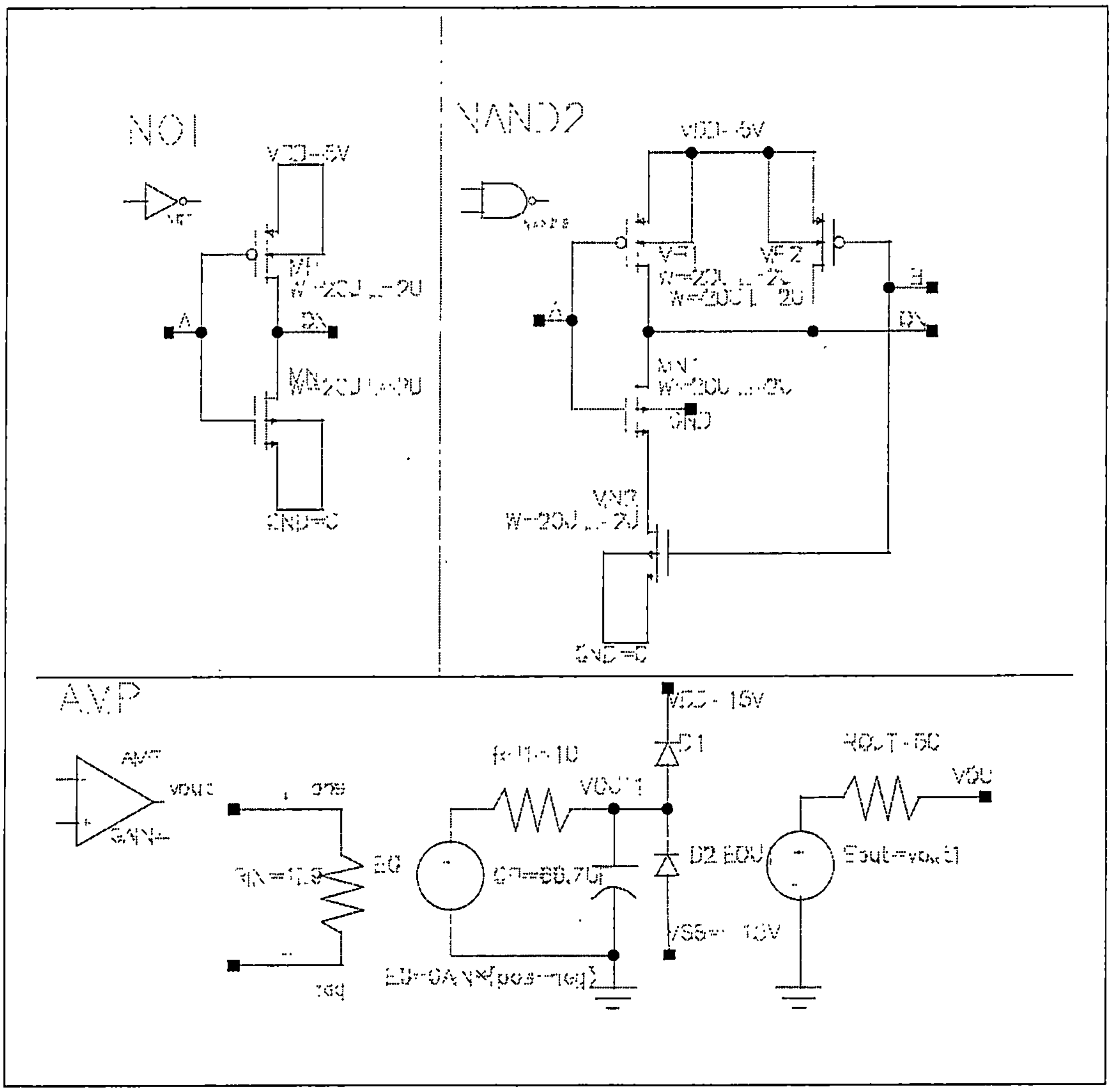


Figure D-1. SPICE primitive component definitions

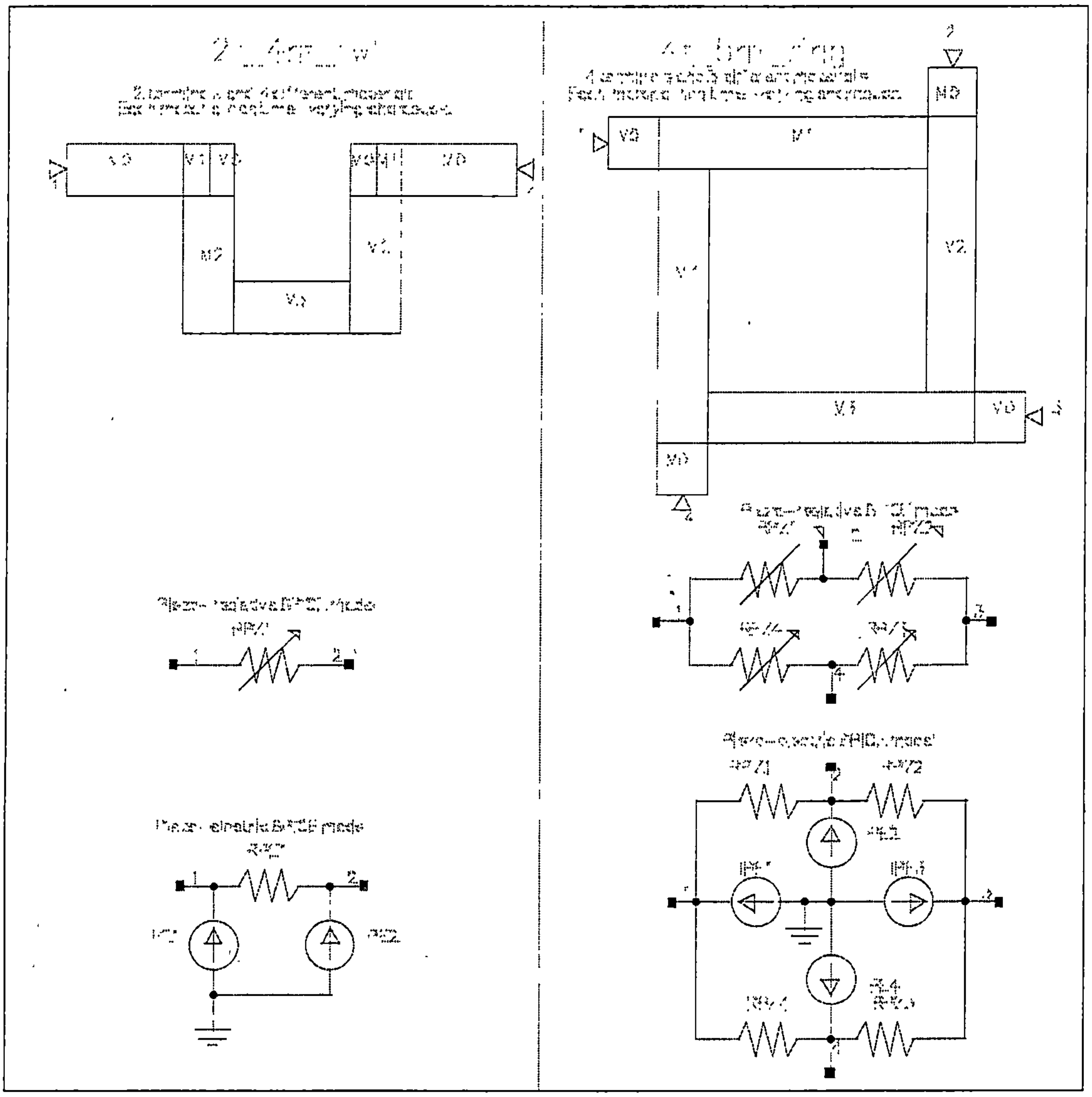


Figure D-2. FESIM and SPICE equivalent component definitions

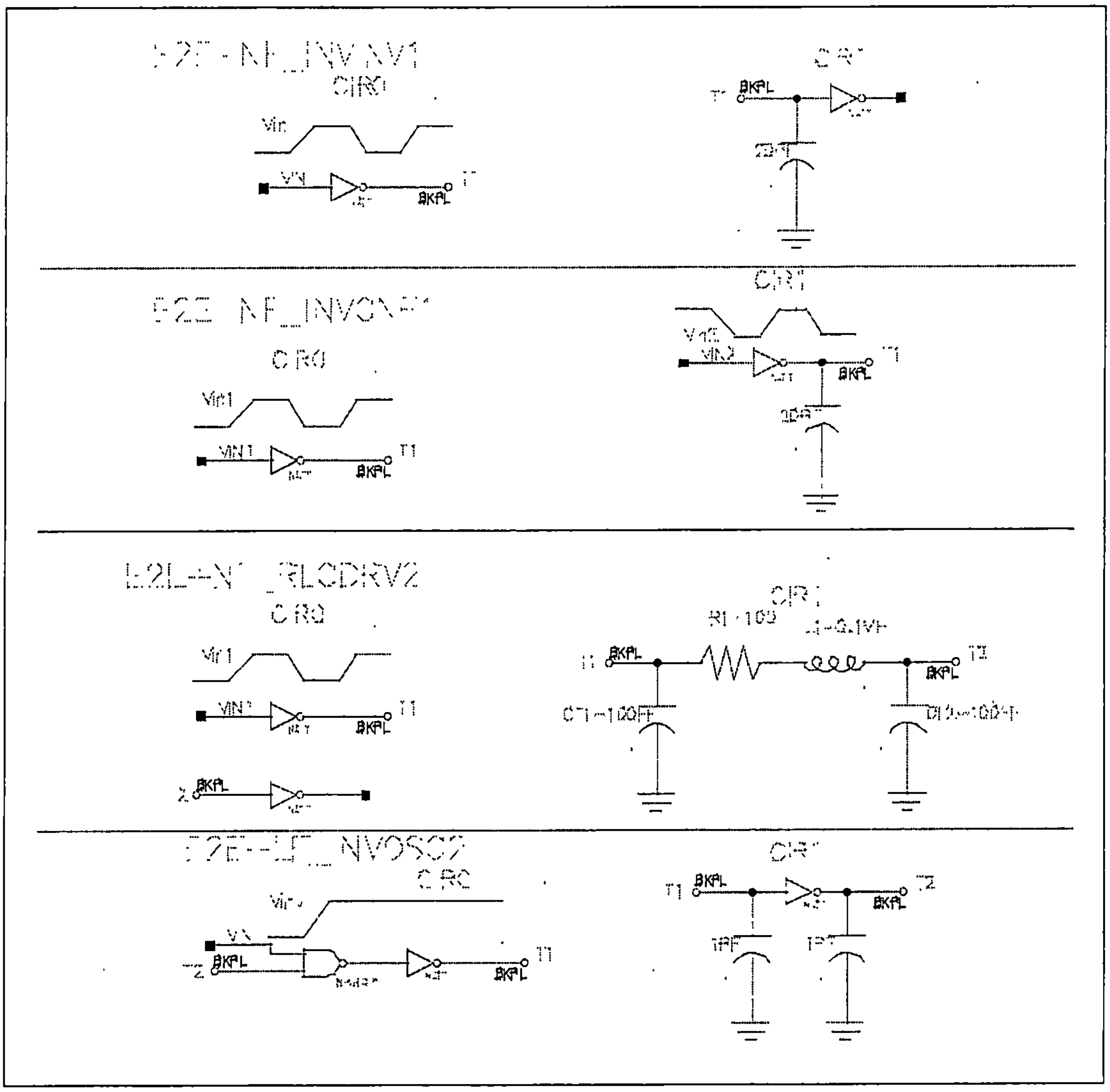


Figure D-3. E2E digital coupling tests

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

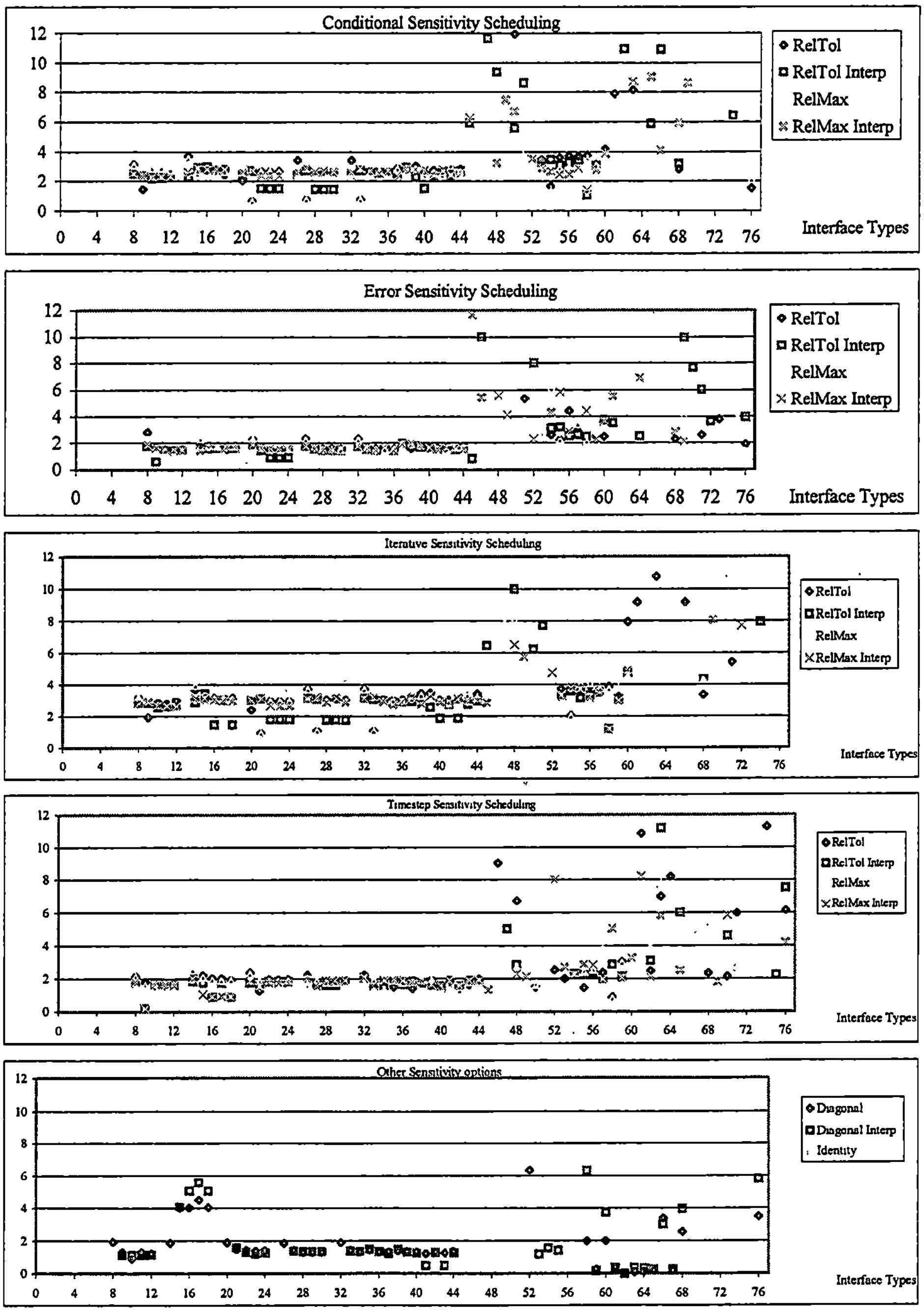


Figure D-4. Iteration results for E2E_NF_INVINV1

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

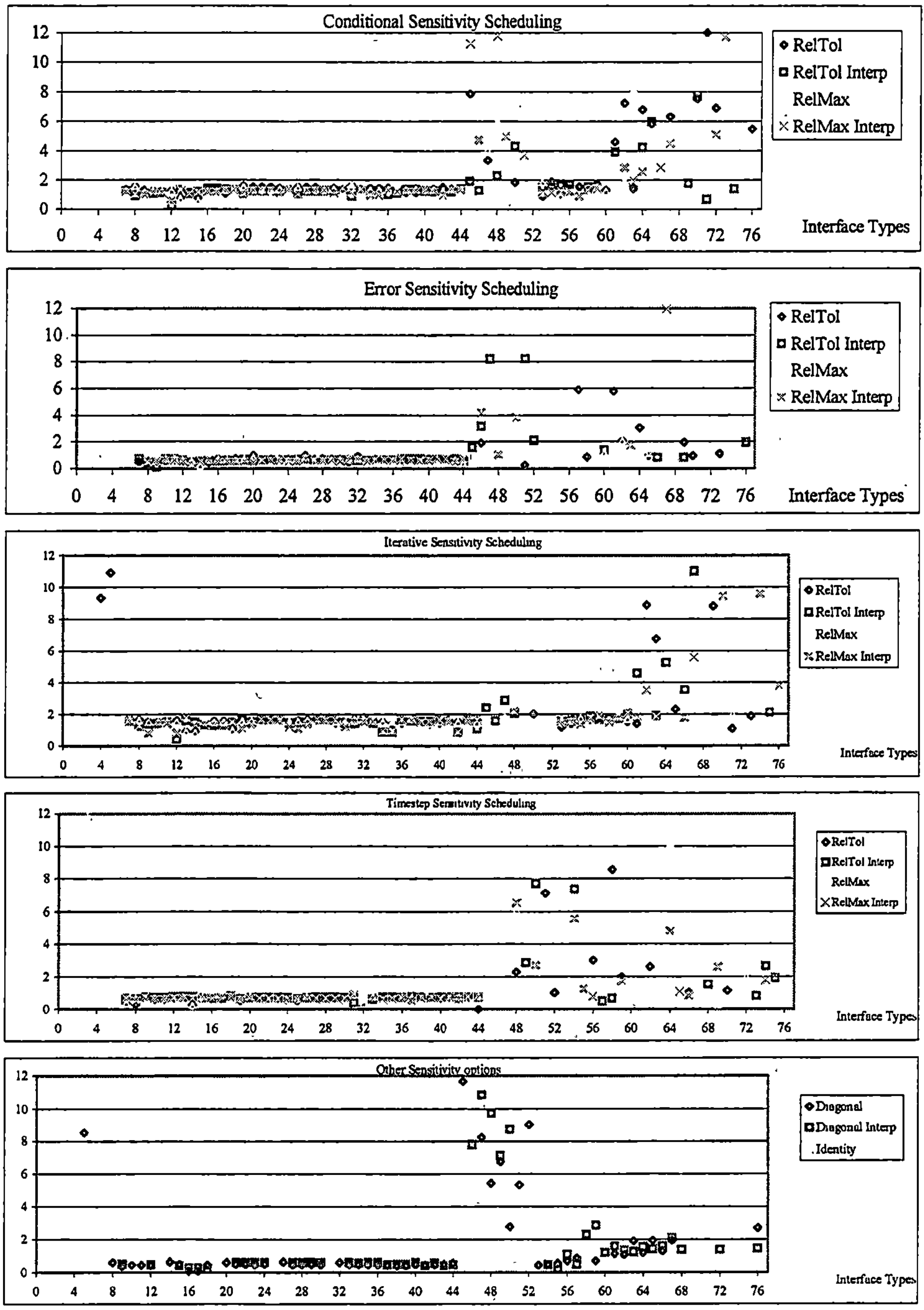


Figure D-5. Iteration results for E2E_NF_INVCNF1

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

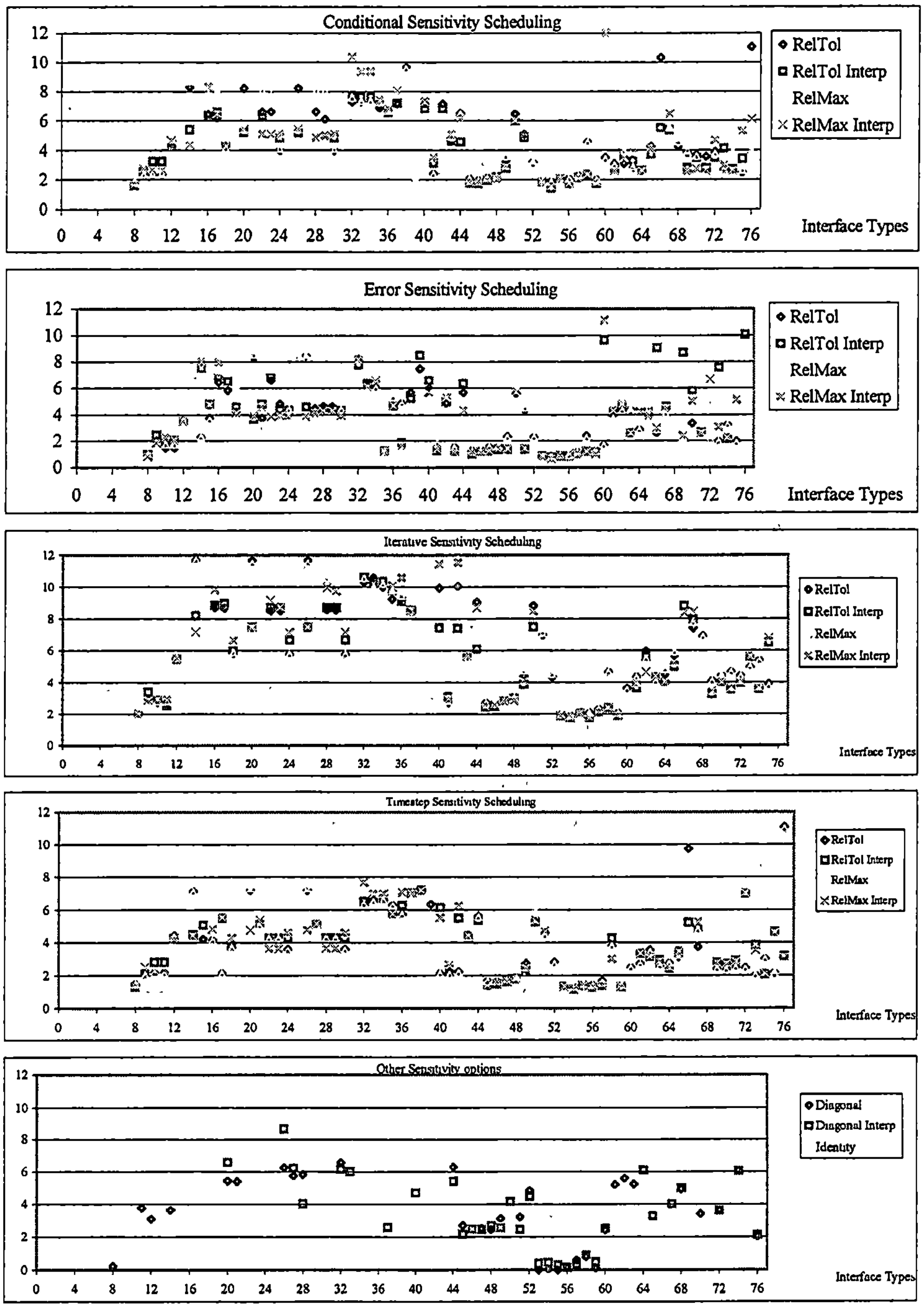


Figure D-6. Iteration results for E2E_NF_RLCDRV2

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

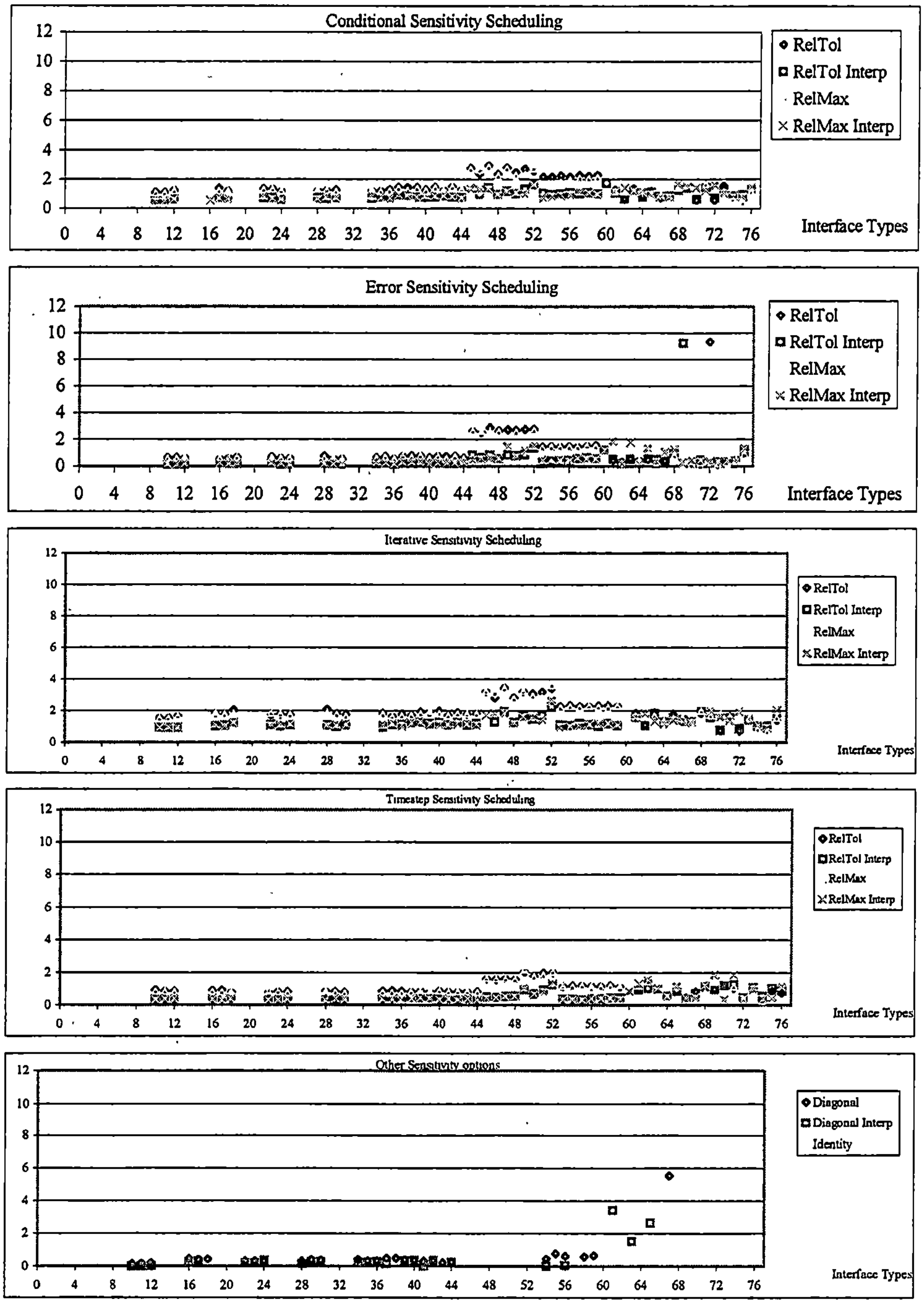


Figure D-7. Iteration results for E2E_LF_INVOSC2

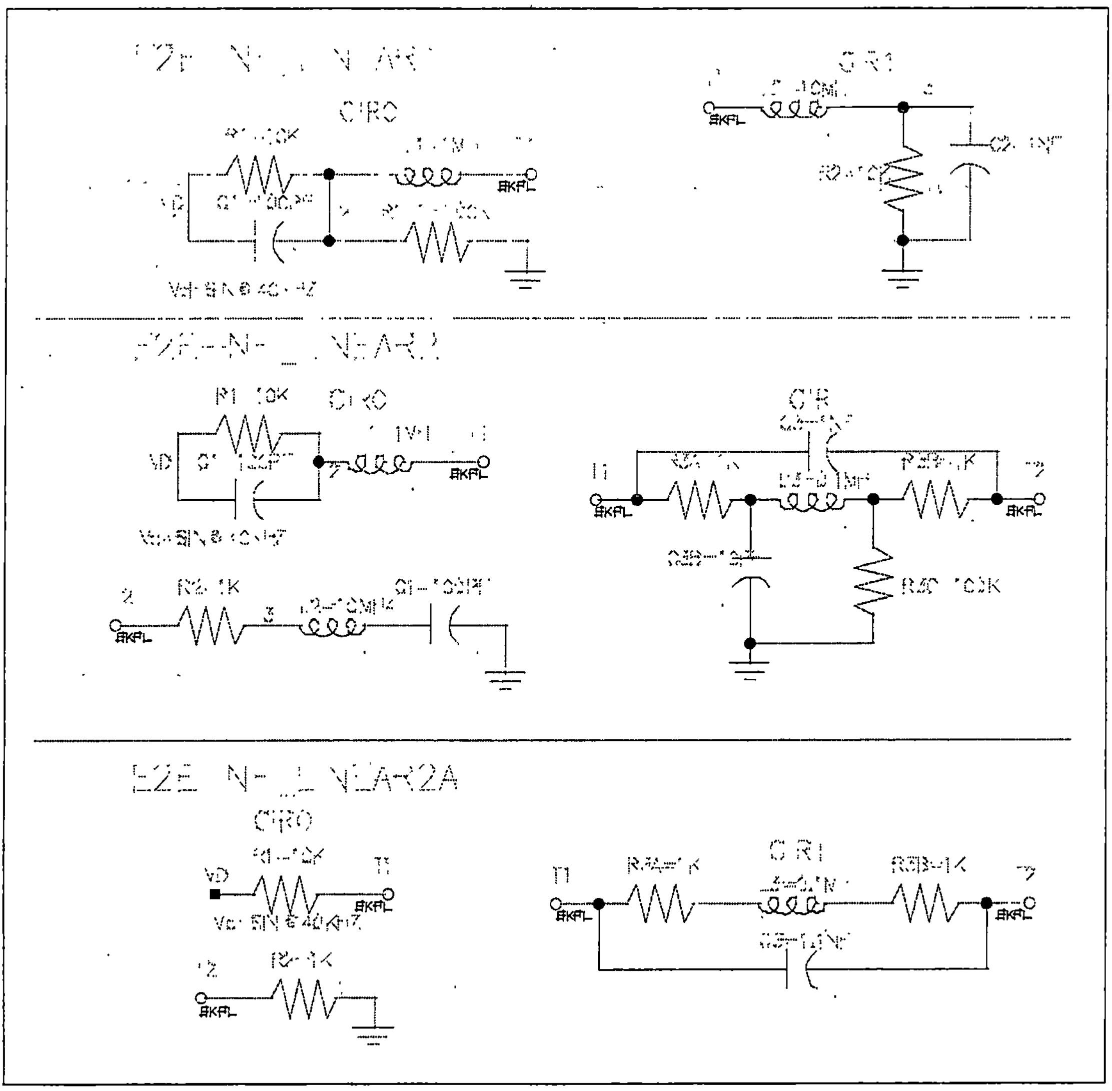


Figure D-8. E2E 1 and 2 terminal linear coupling tests

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

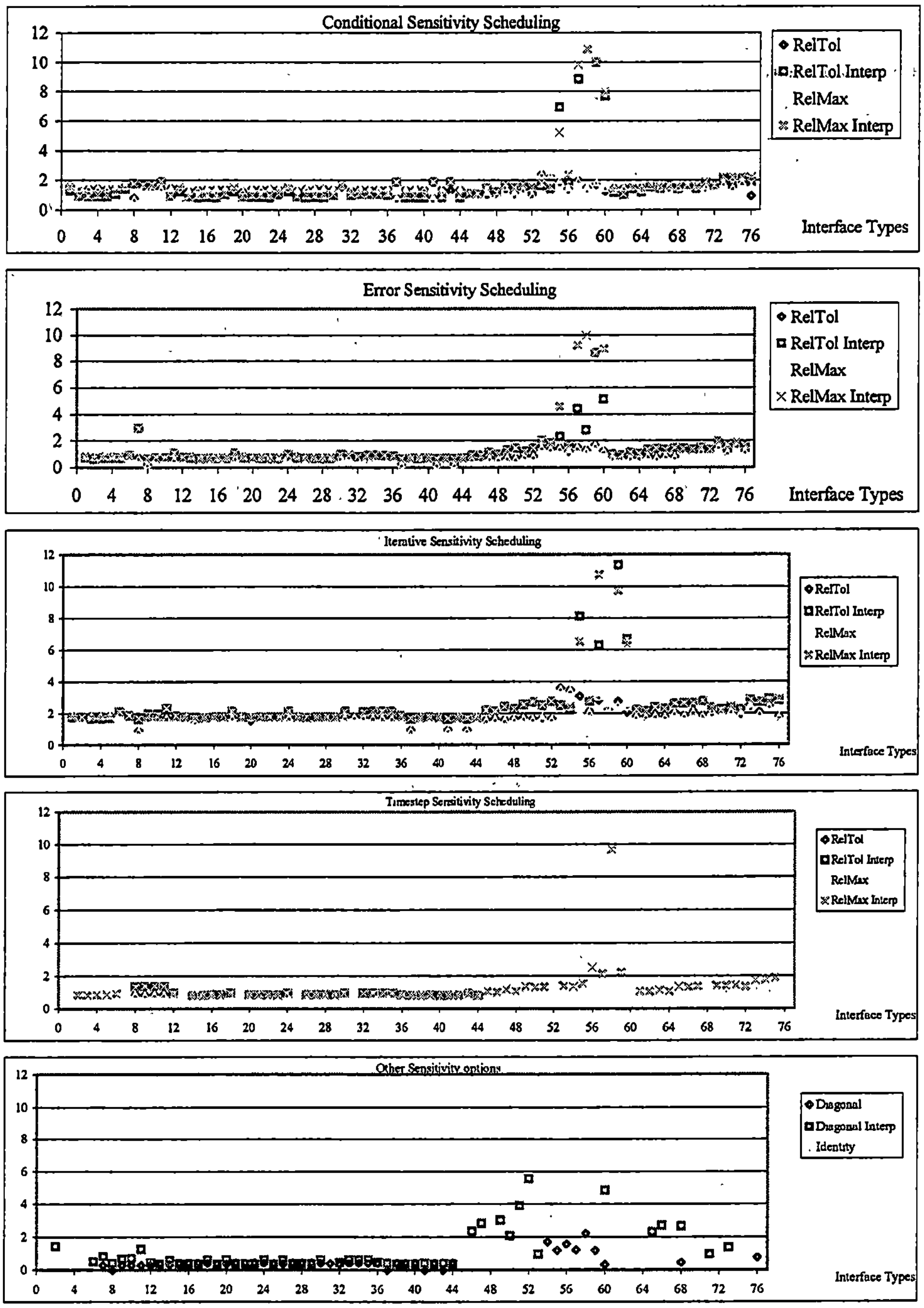


Figure D-9. Iteration results for E2E_NF_LINEAR1

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

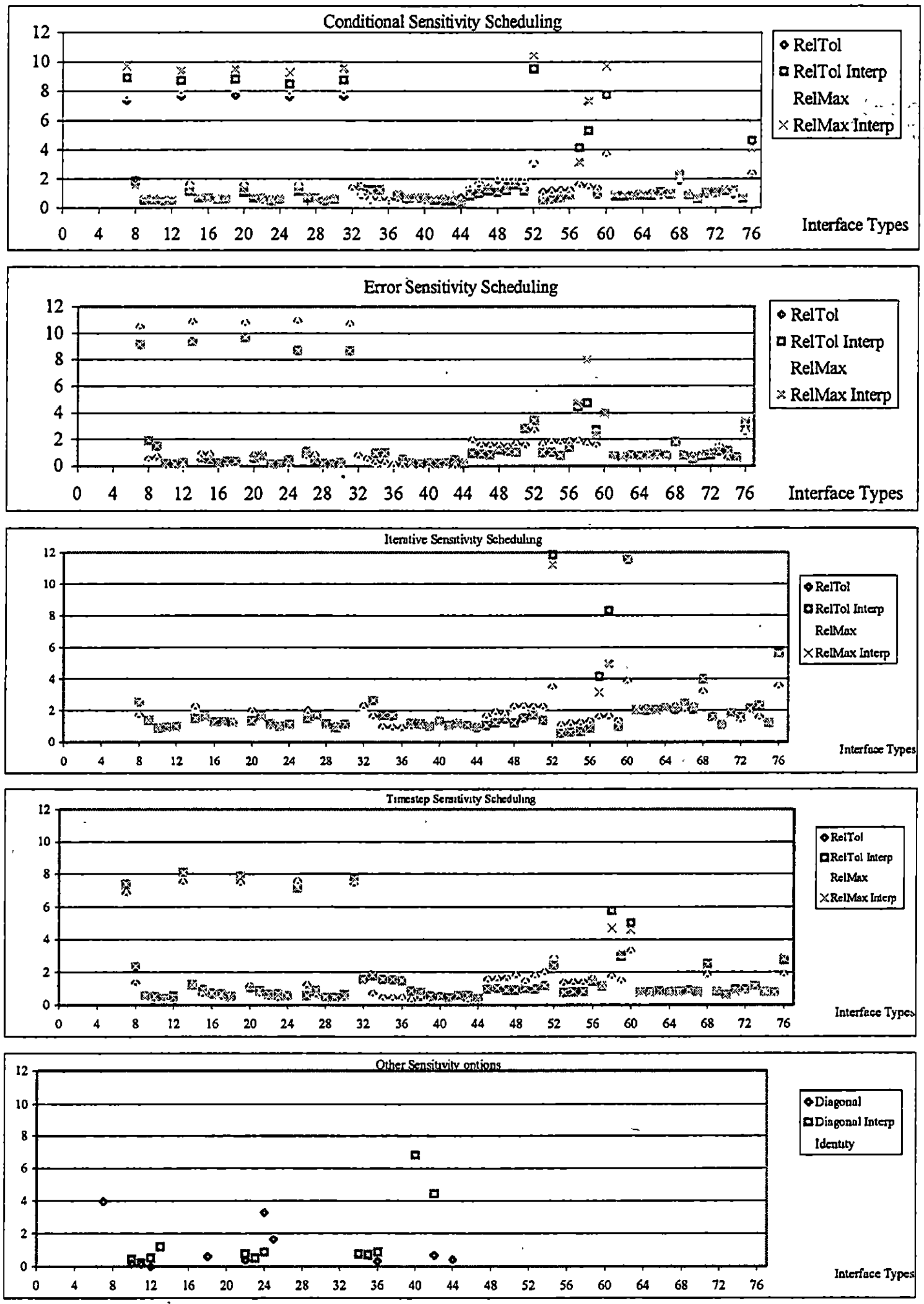


Figure D-10. Iteration results for E2E_NF_LINEAR2

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

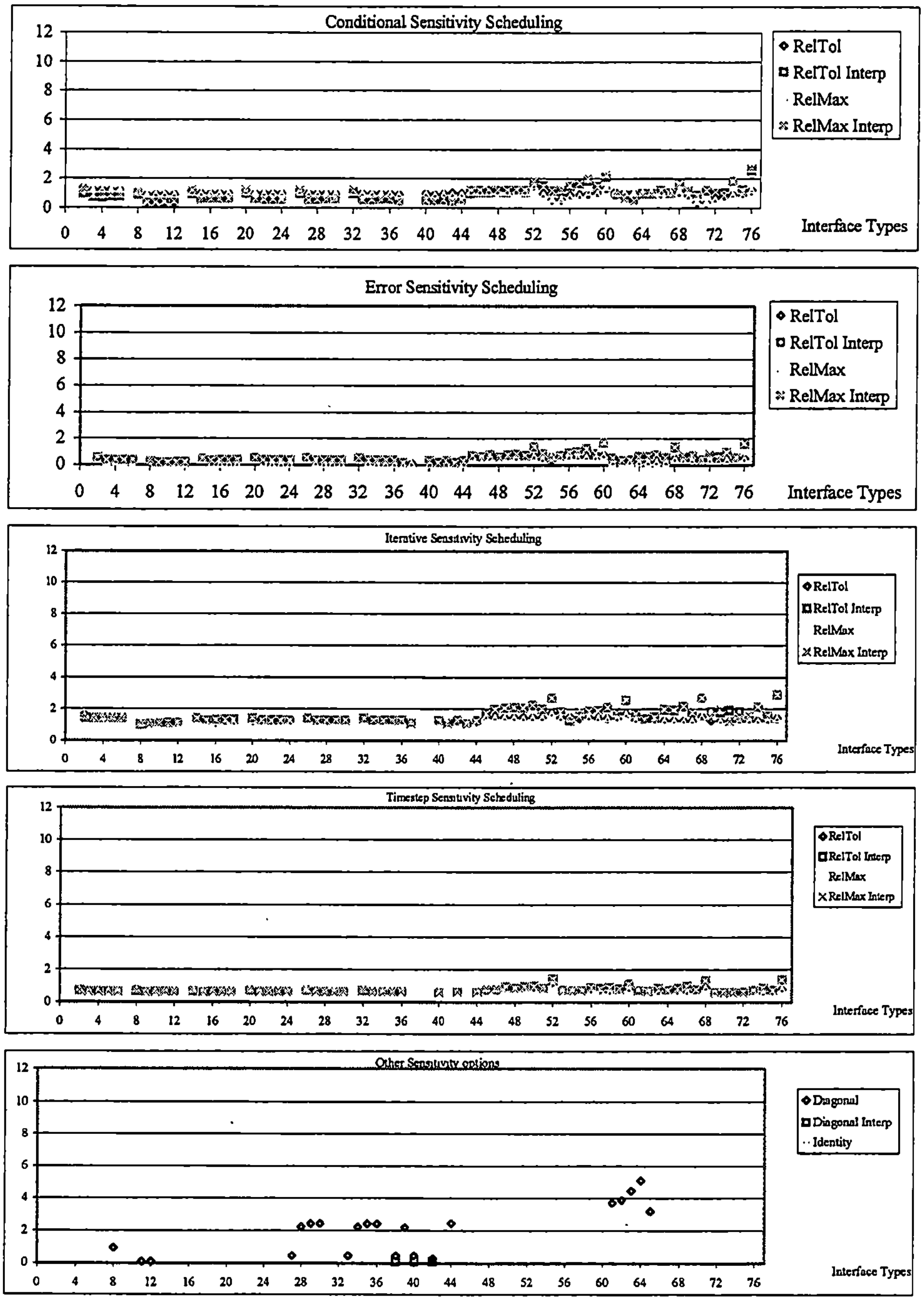


Figure D-11. Iteration results for E2E_NF_LINEAR2a

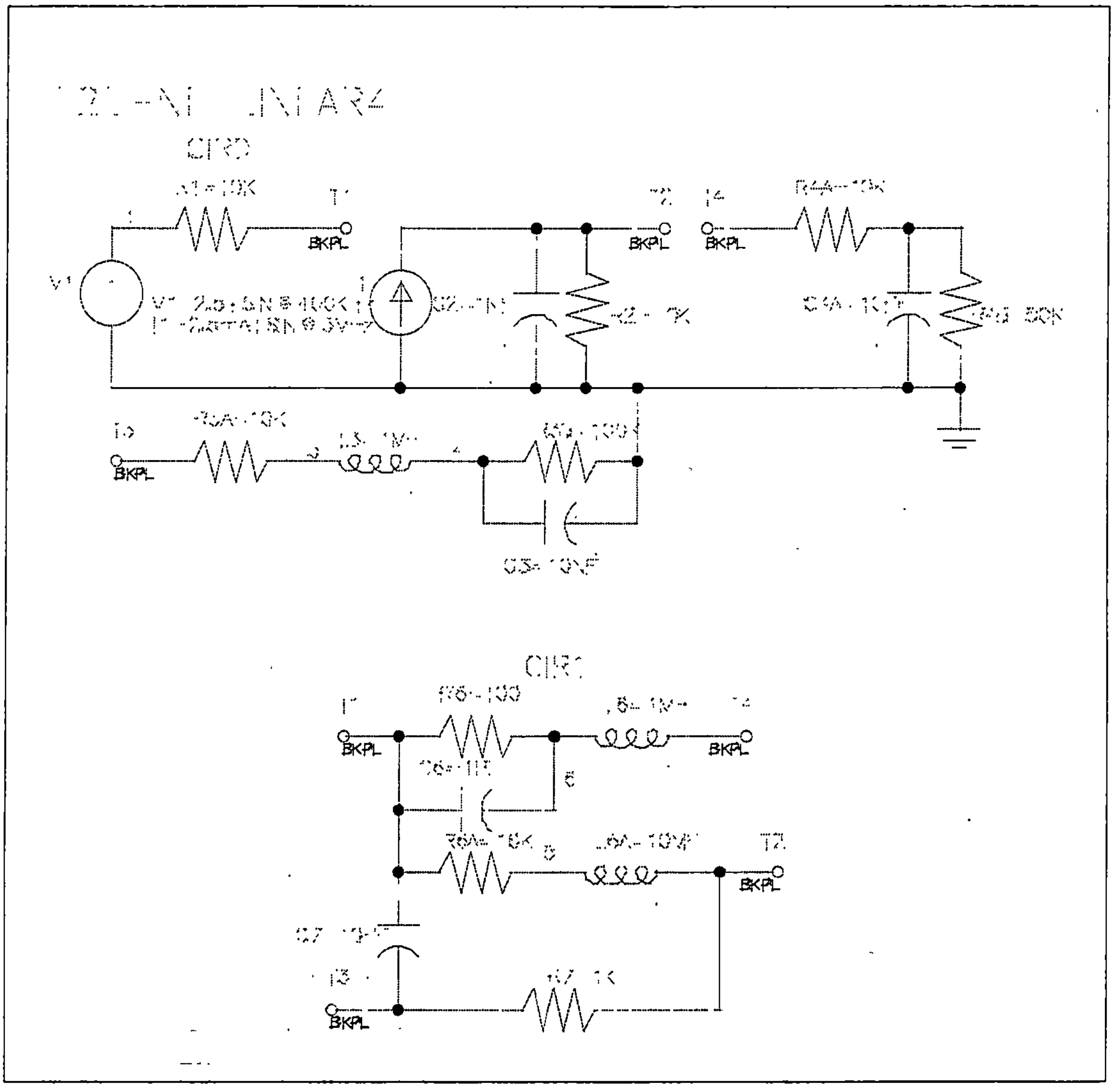


Figure D-12. E2E 4-terminal linear example

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

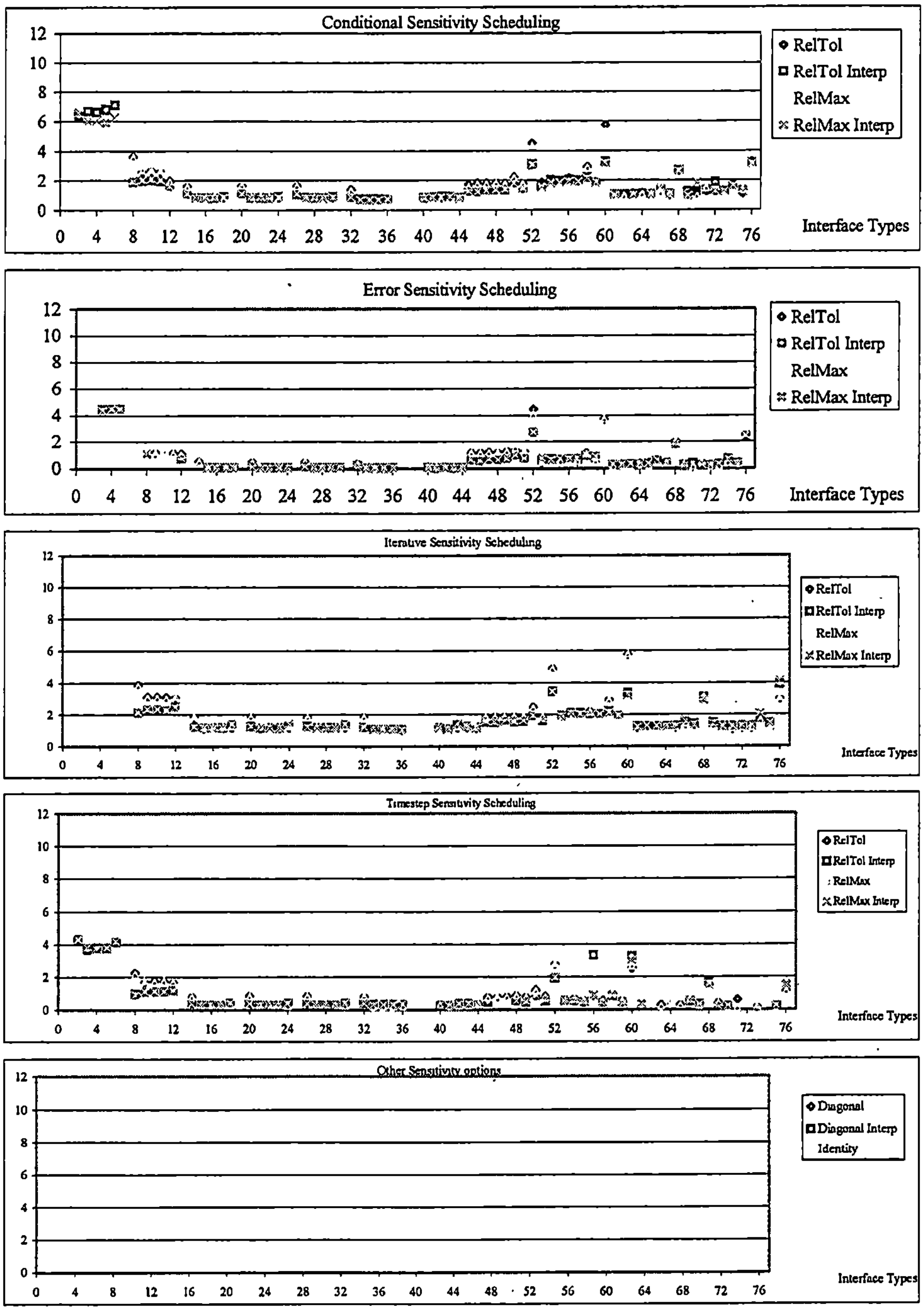


Figure D-13. Iteration results for E2E_NF_LINEAR4

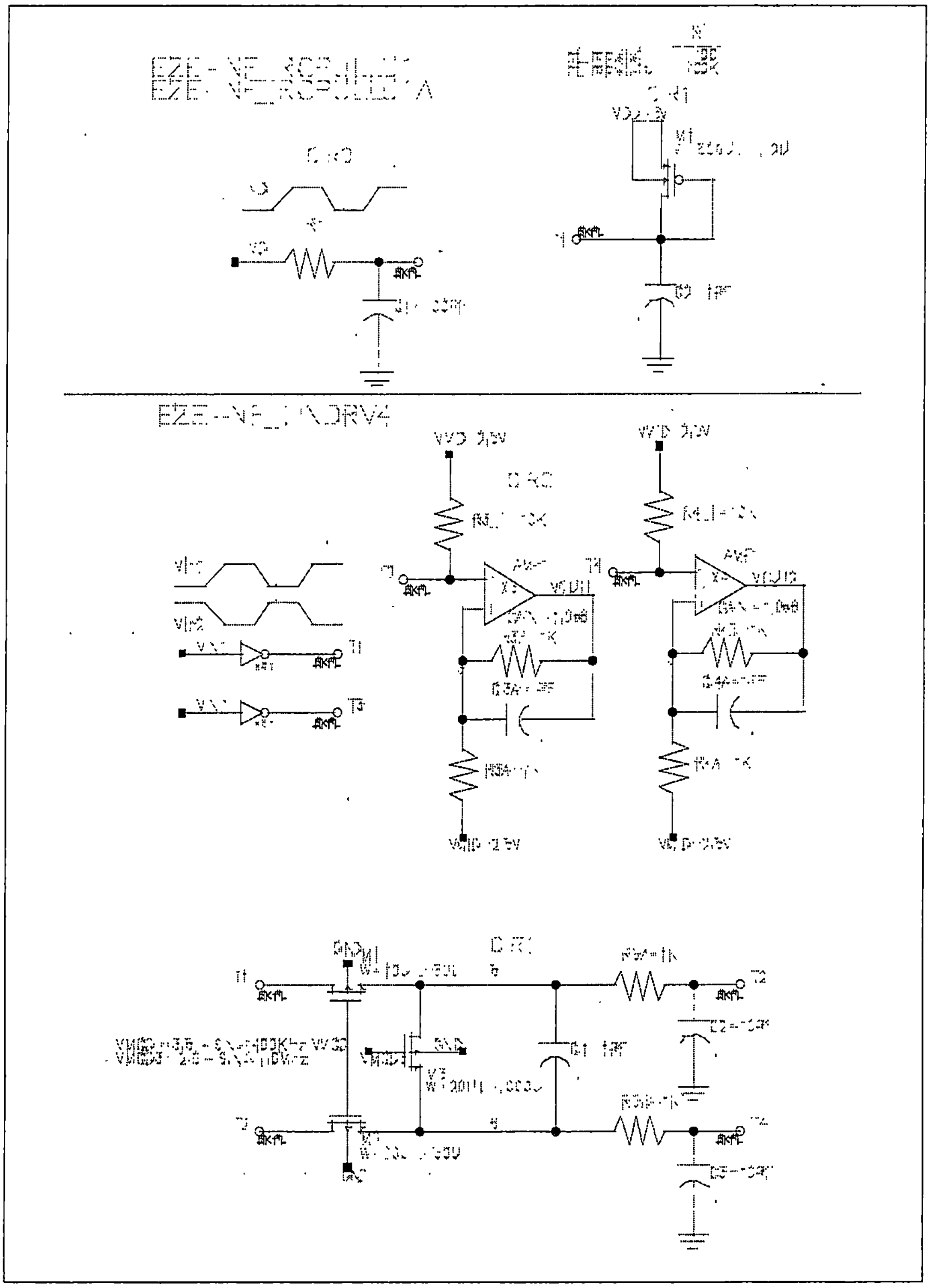


Figure D-14. E2E nonlinear no-feedback coupling tests

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

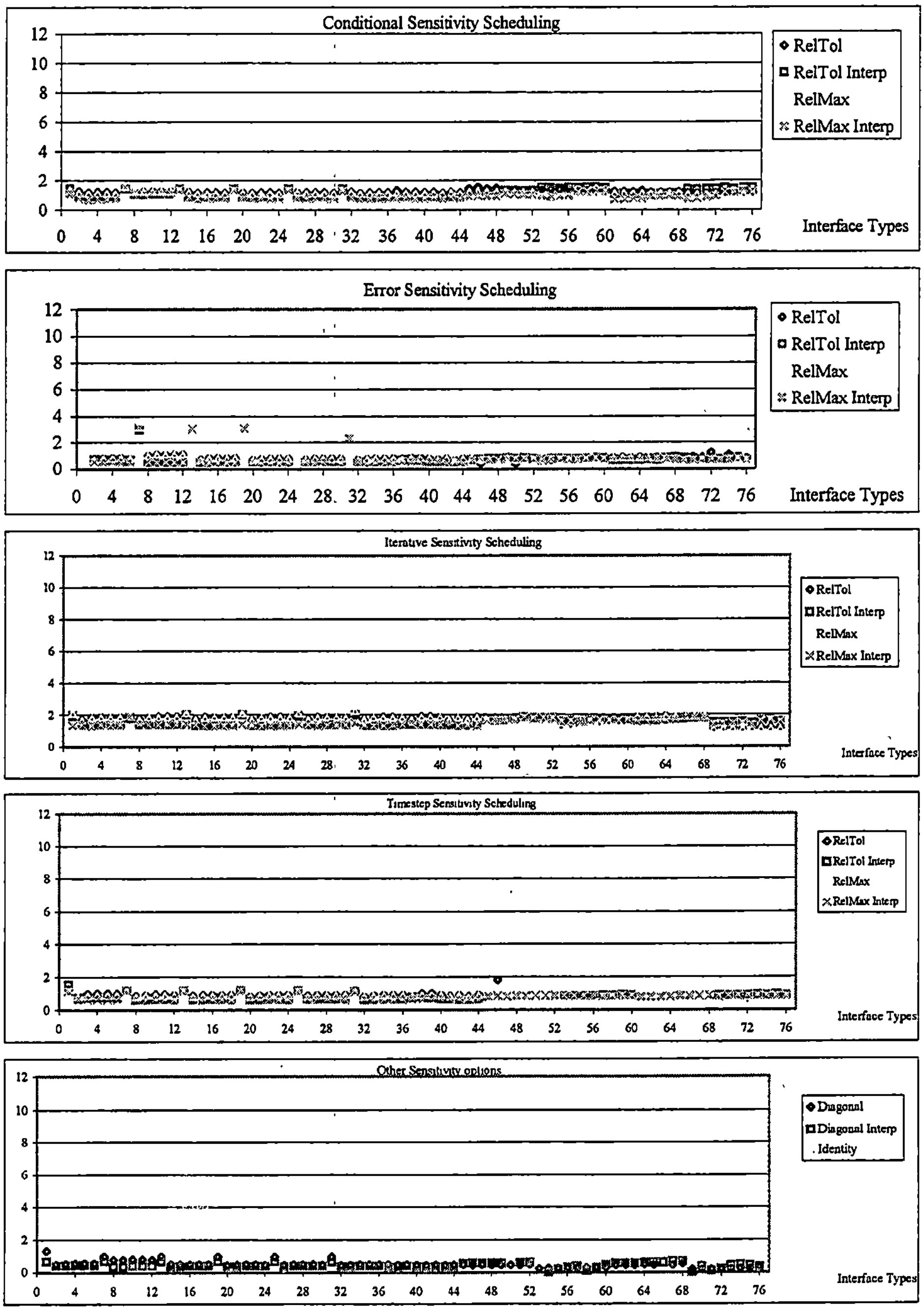


Figure D-15. Iteration results for E2E_NF_RCPULLU1

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

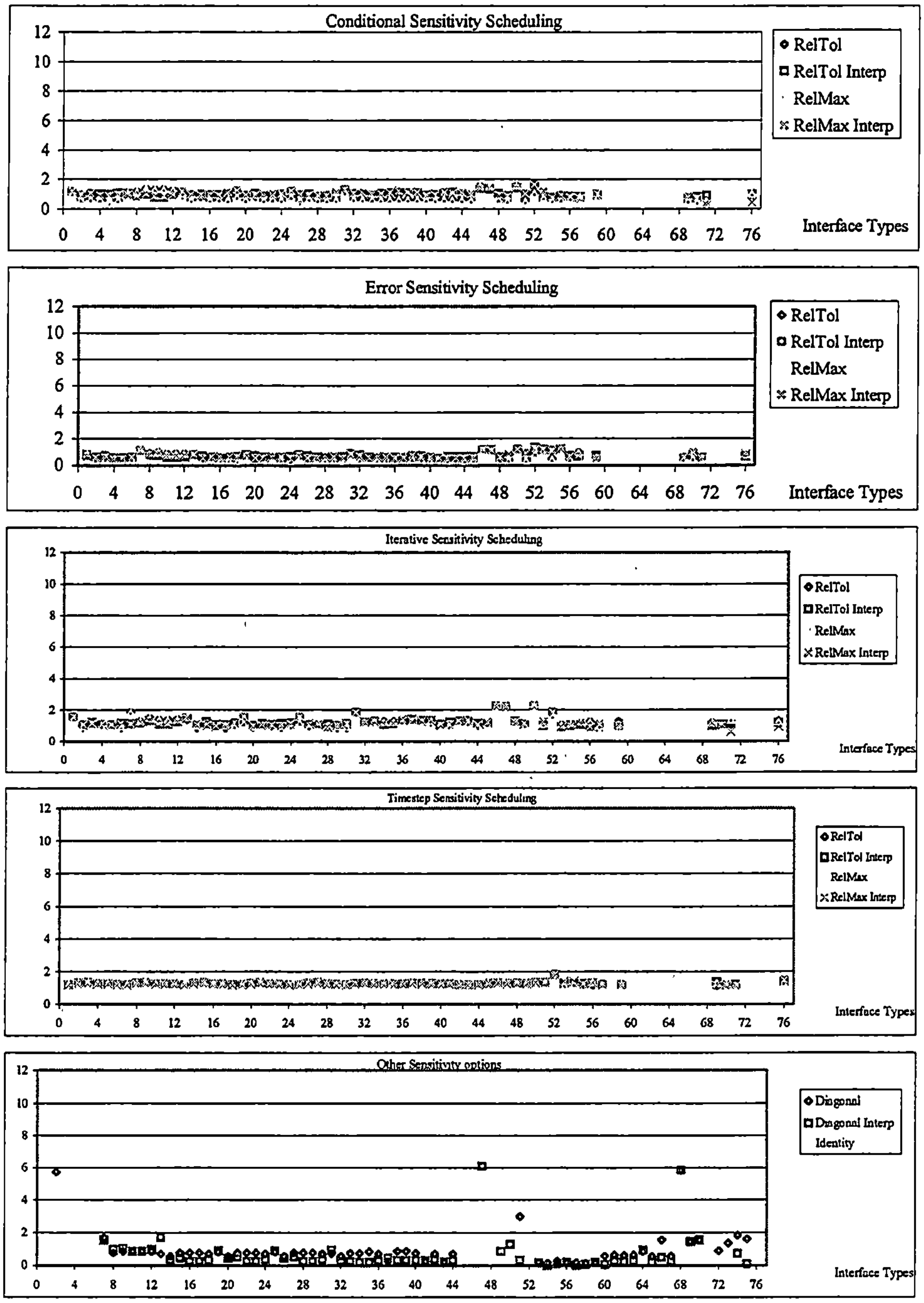


Figure D-16. Iteration results for E2E_NF_RCPULLU1A

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

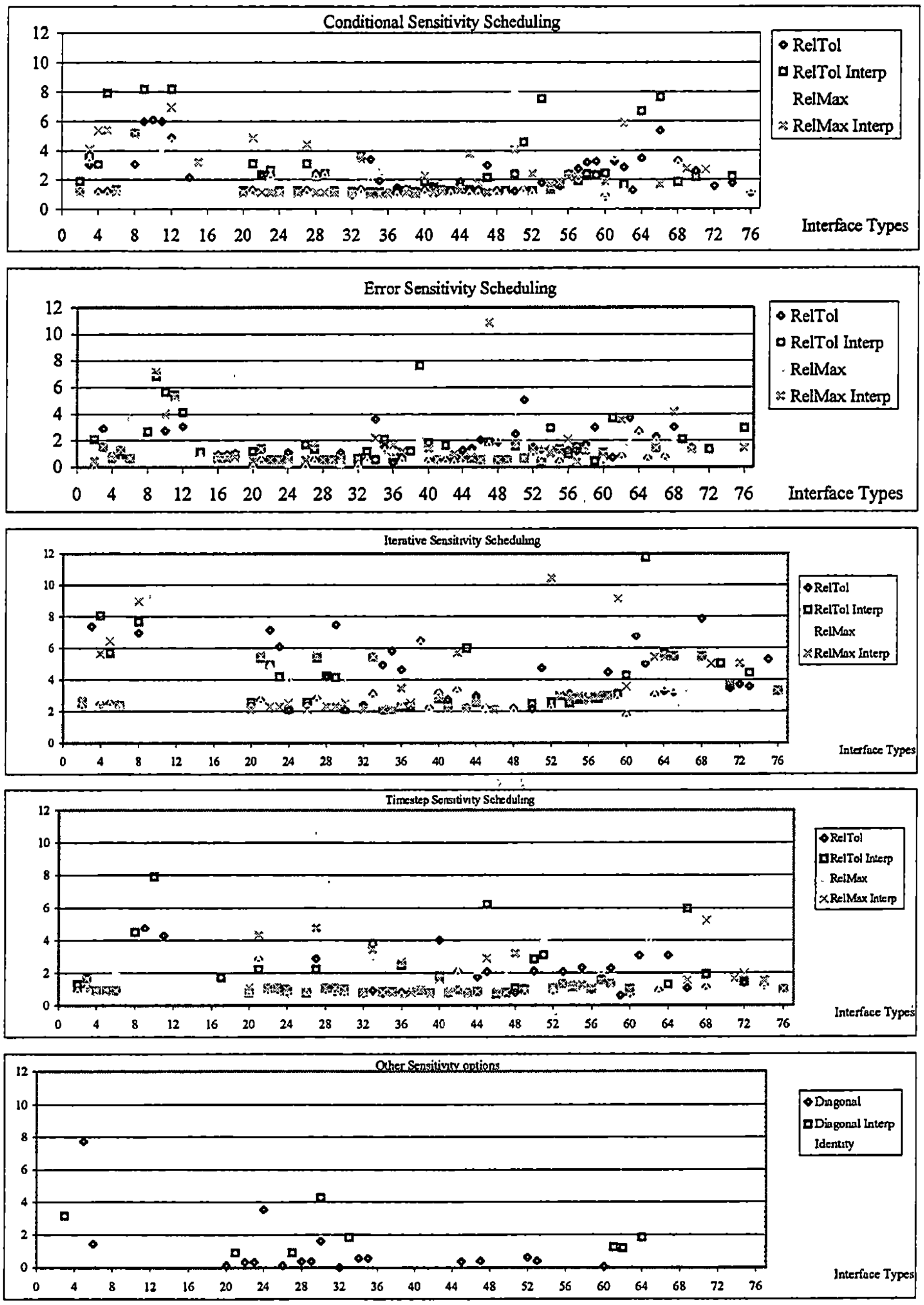


Figure D-17. Iteration results for E2E_NF_LINDRV4

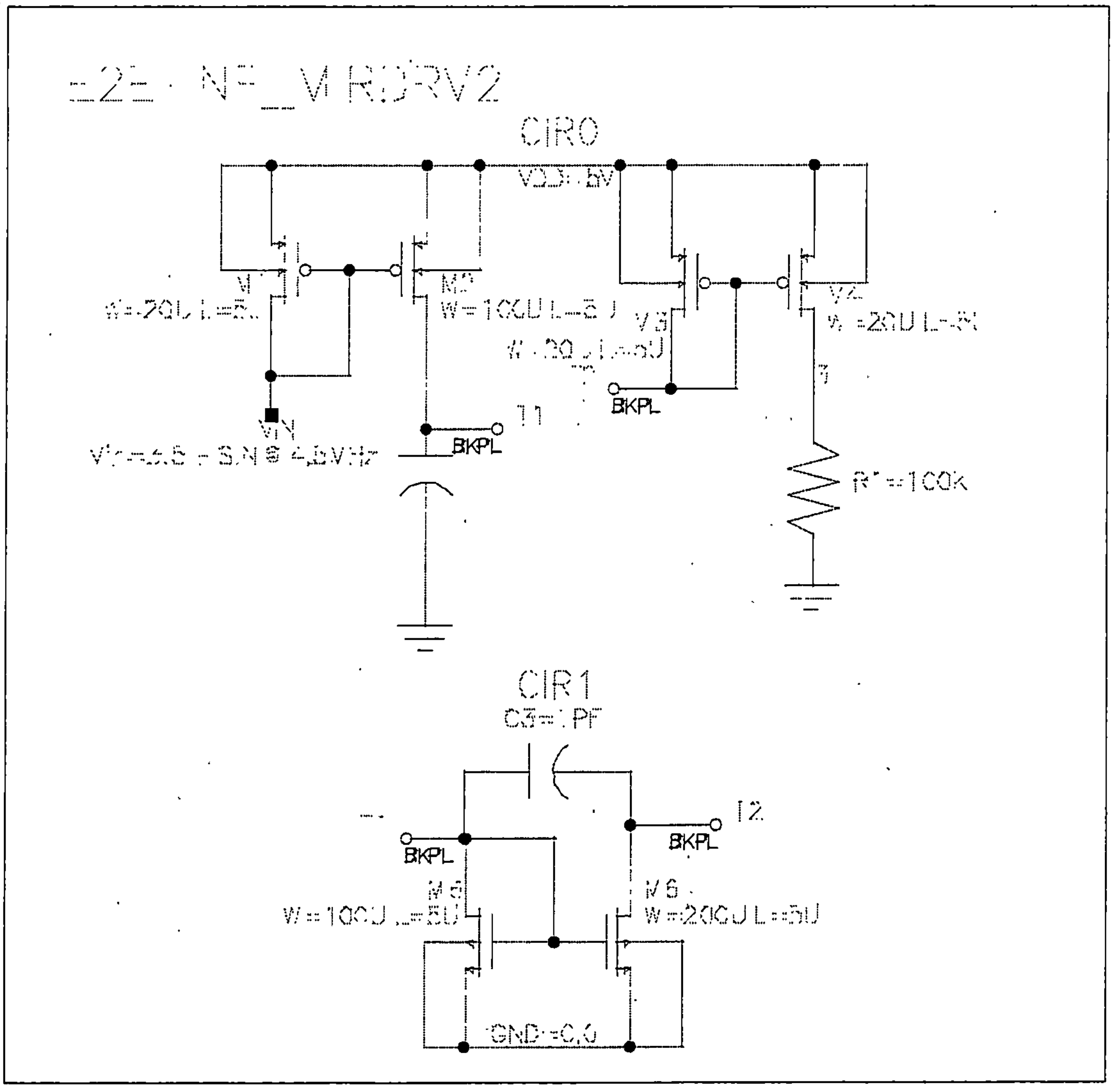


Figure D-18. E2E 2-terminal nonlinear current mirror

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

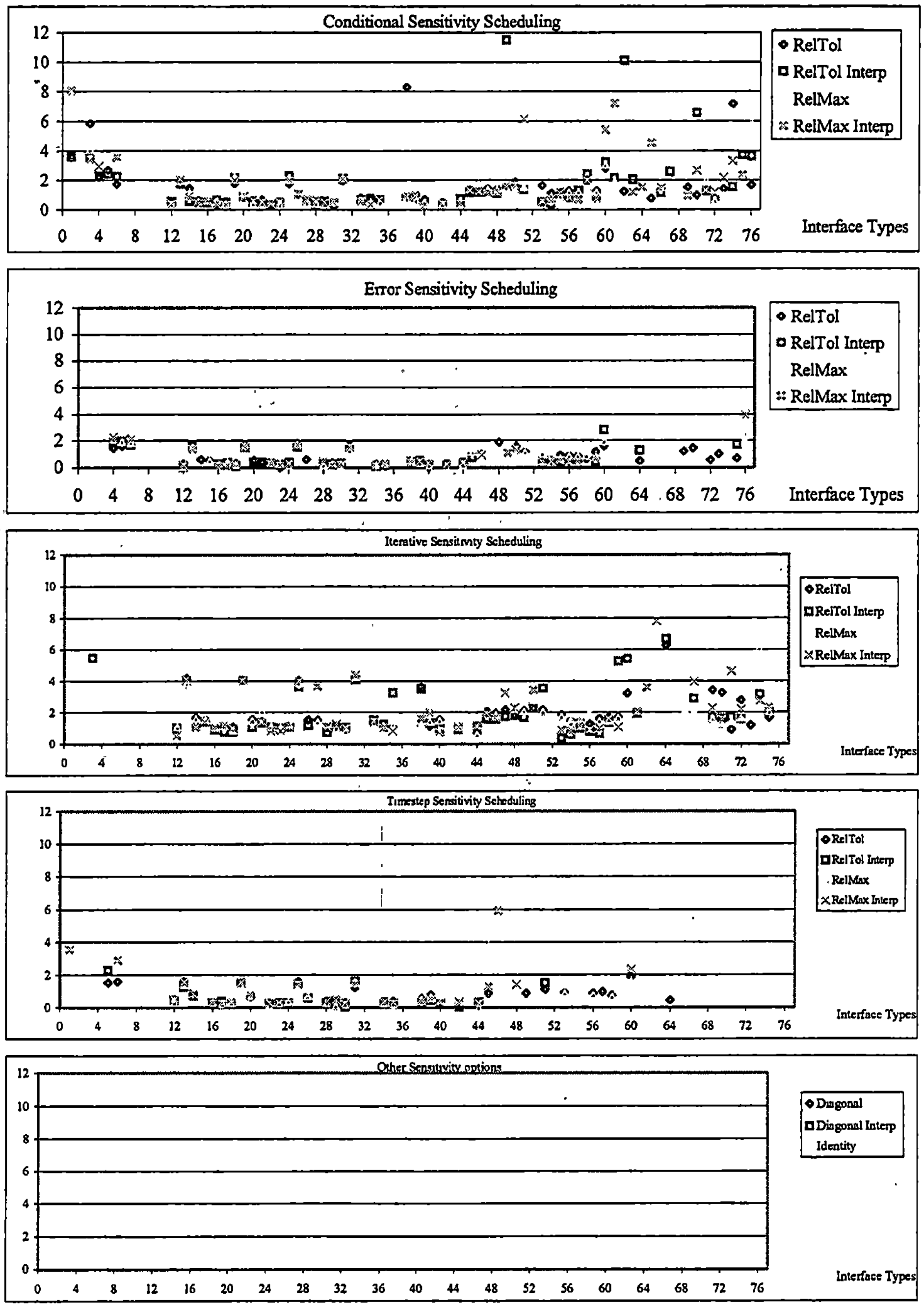


Figure D-19. Iteration results for E2E_NF_MIRDRV2

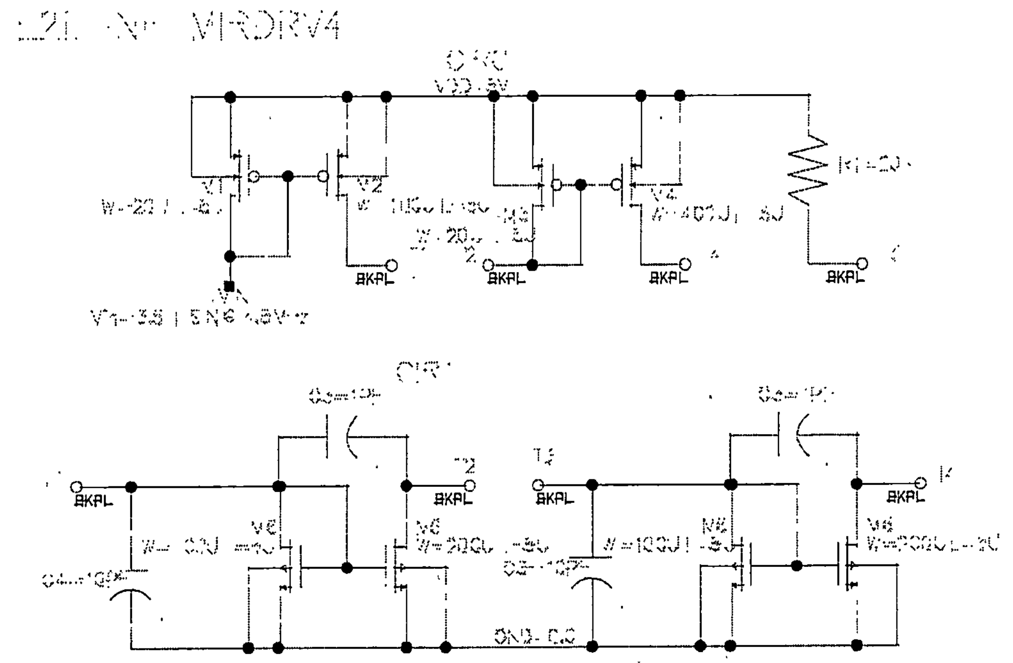


Figure D-20. E2E 4-terminal nonlinear current mirror

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

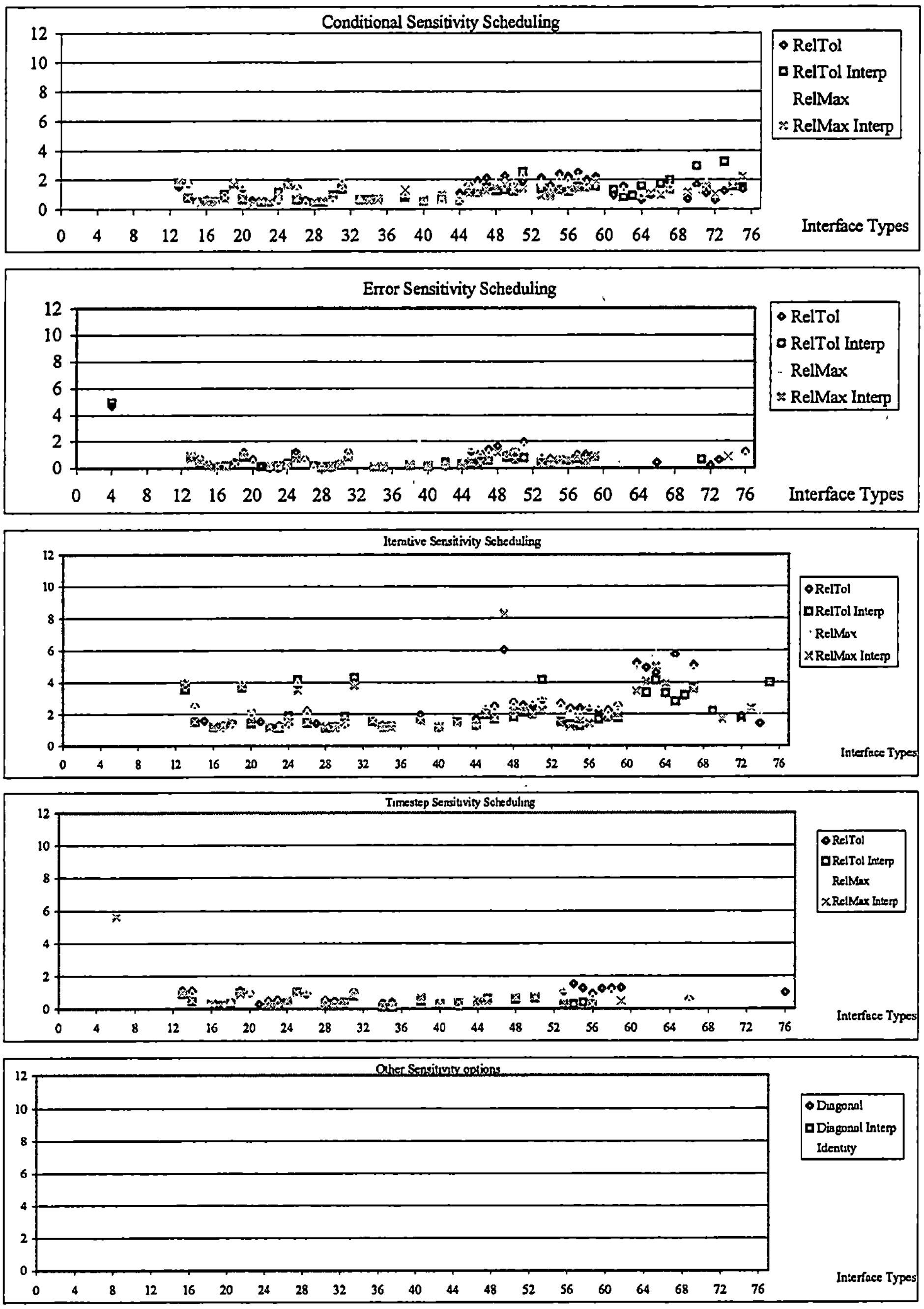


Figure D-21. Iteration results for E2E_NF_MIRDRV4

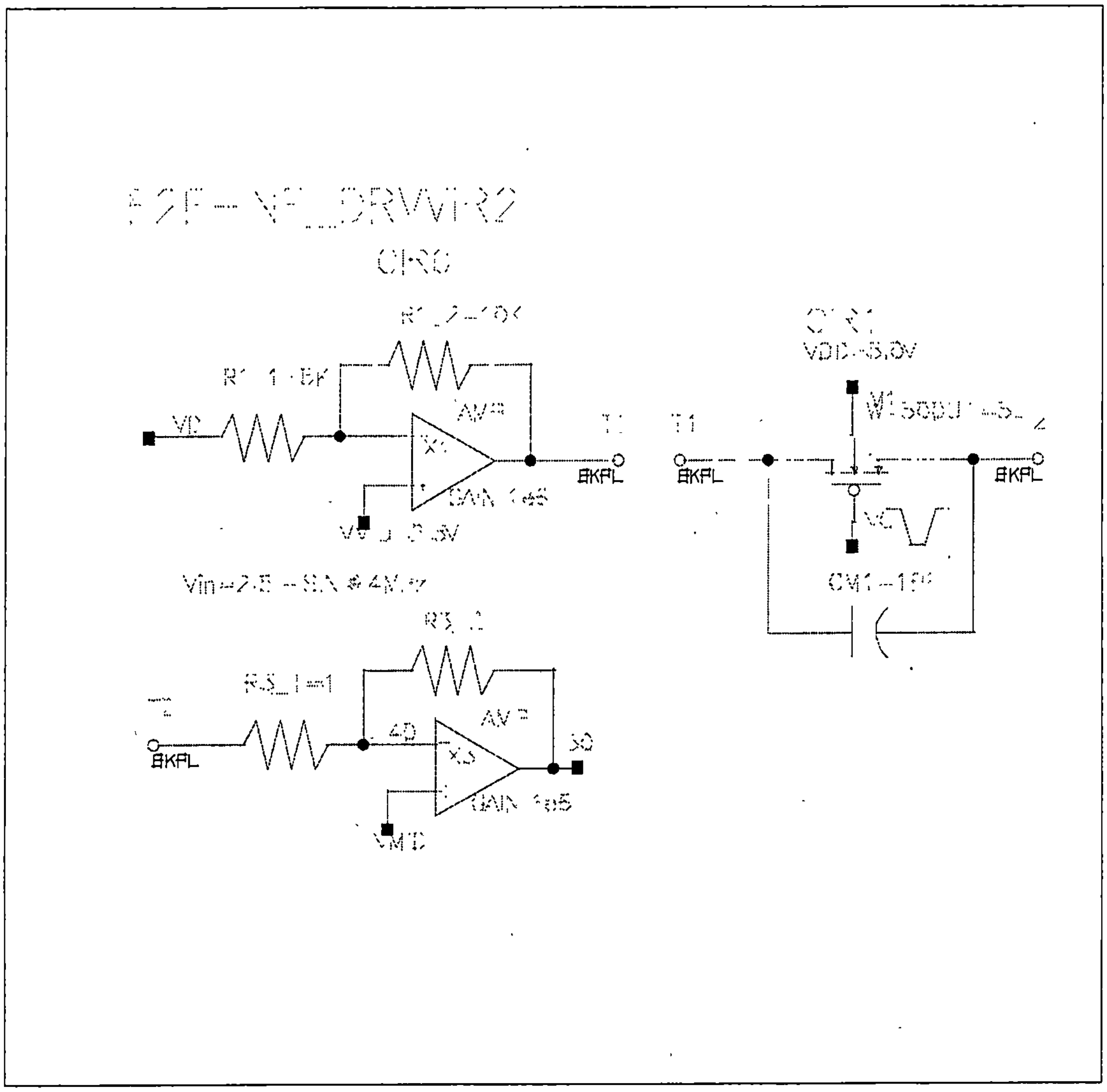


Figure D-22. E2E 2-terminal nonlinear driver

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

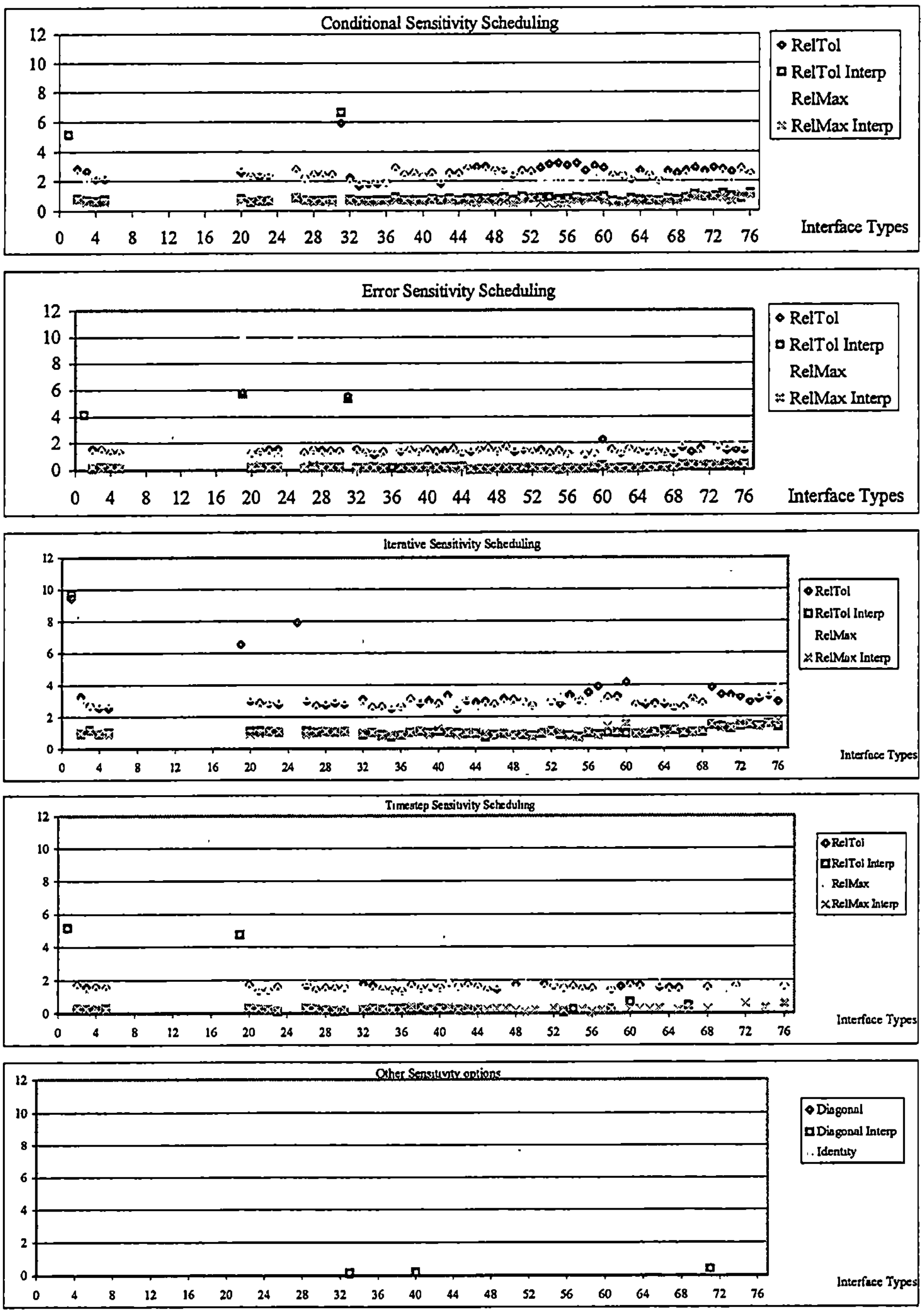


Figure D-23. Iteration results for E2E_NF_DRVVIR2

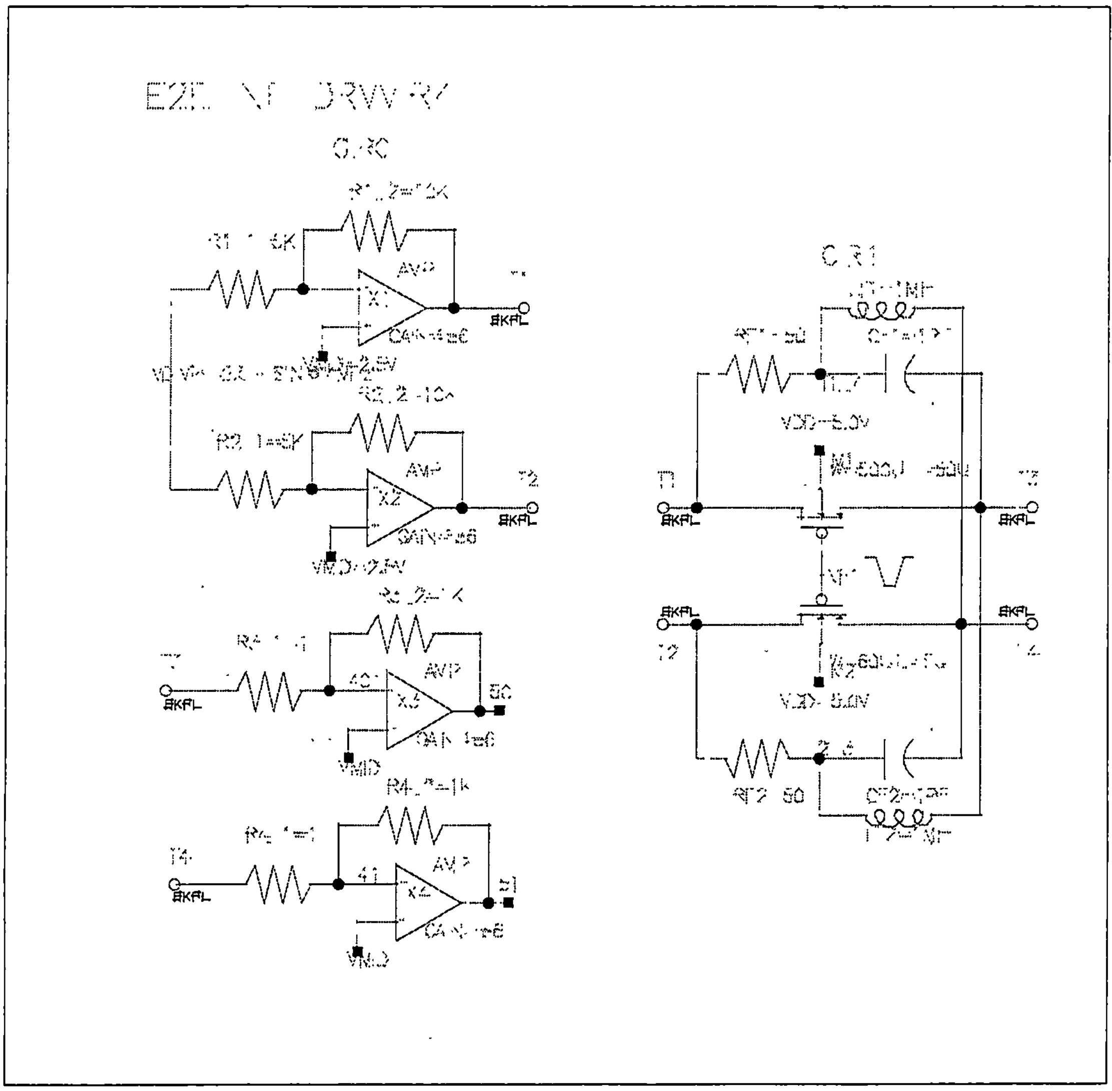


Figure D-24. E2E 4-terminal nonlinear driver

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

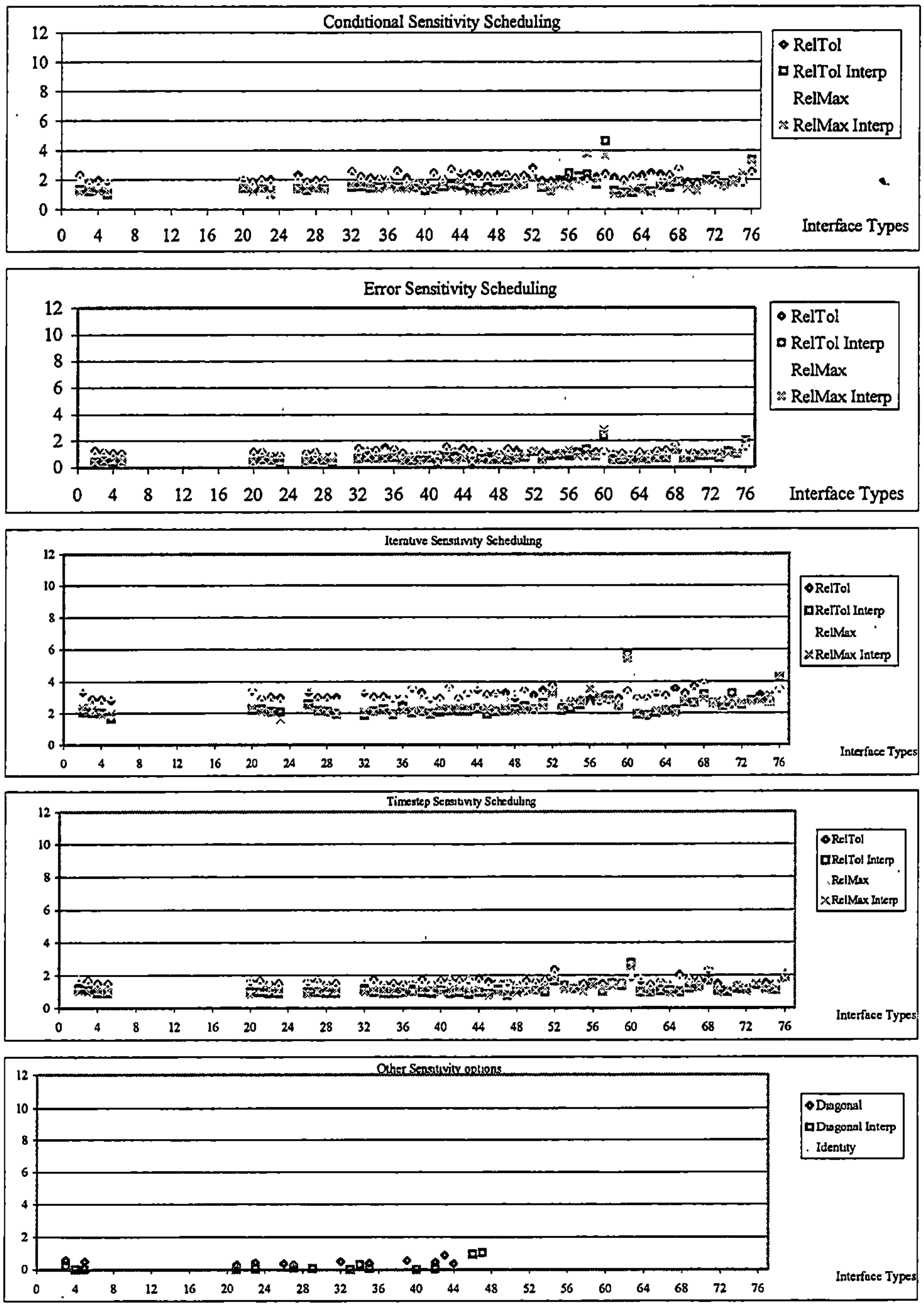


Figure D-25. Iteration results for E2E_NF_DRVVIR4

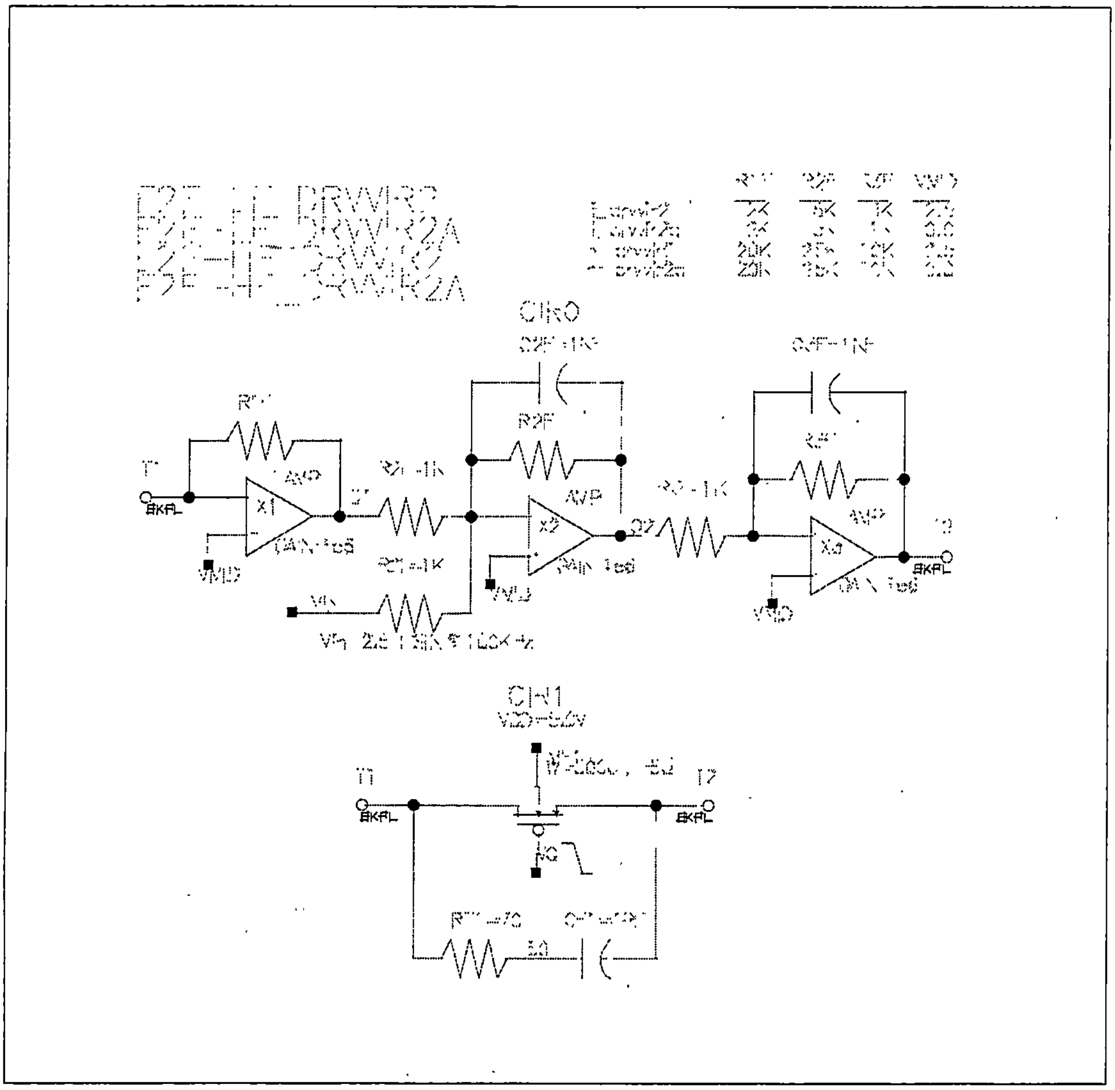


Figure D-26. E2E 2-terminal virtual driver example

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

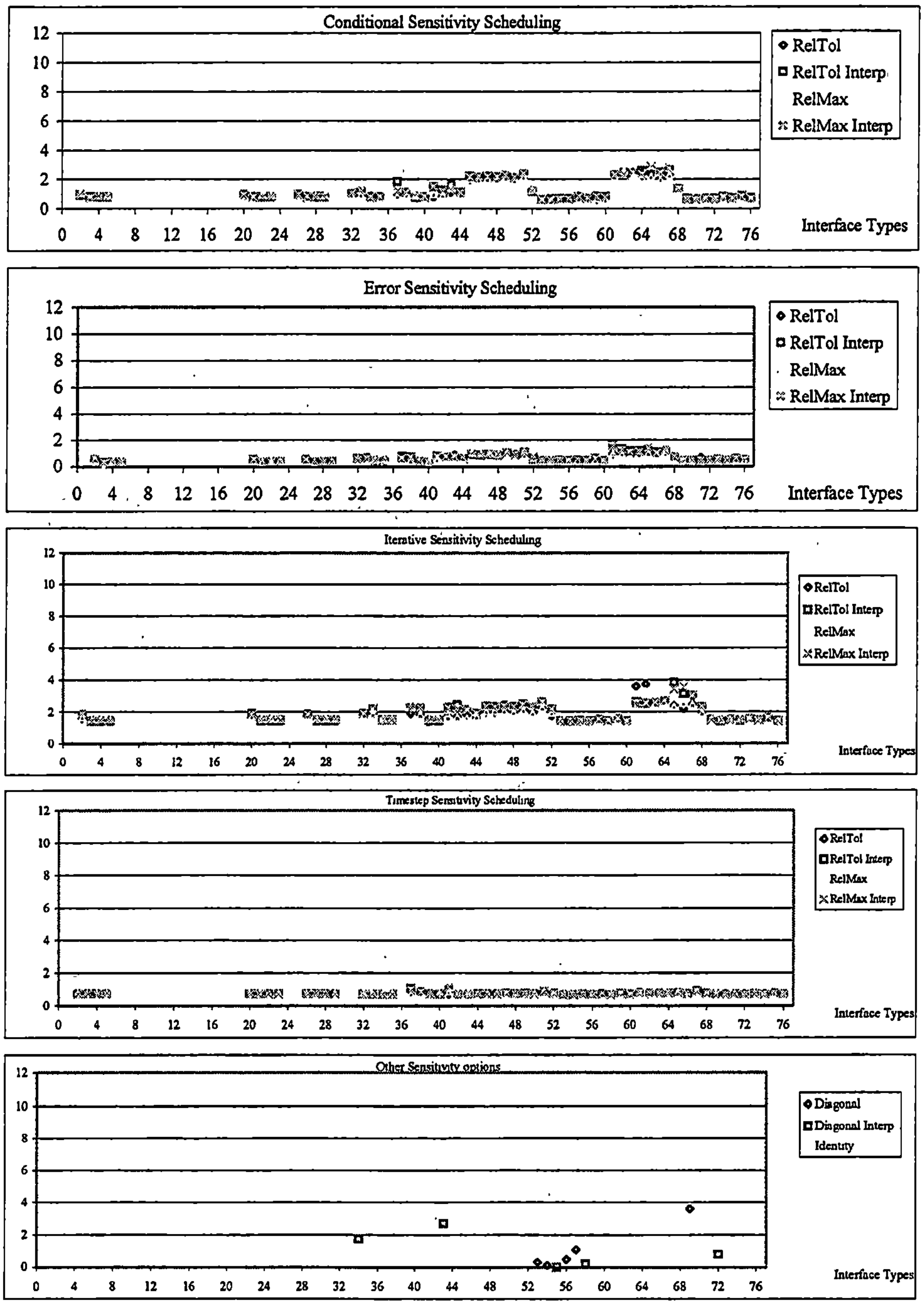


Figure D-27. Iteration results for E2E_LF_DRVVIR2

1-36: Conventional
37-44: Dynamic interfacing
45-52: Parallel Correction (ESRC)
53-60: Sequential Correction (ESRC)
61-68: Parallel Correction (ESRC_SENS)
69-76: Sequential Correction (ESRC_SENS)

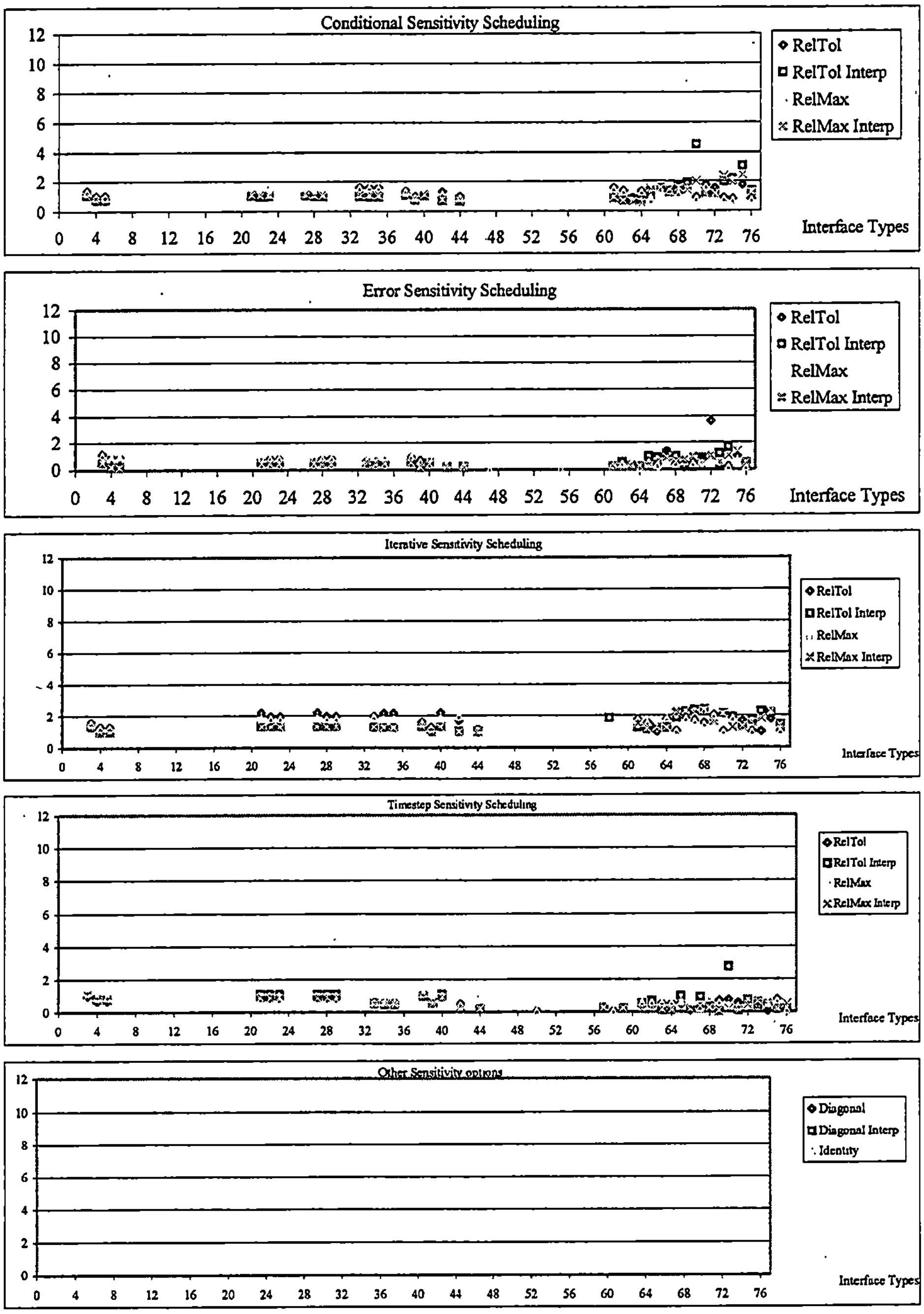


Figure D-28. Iteration results for E2E_LF_DRVVIR2a

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

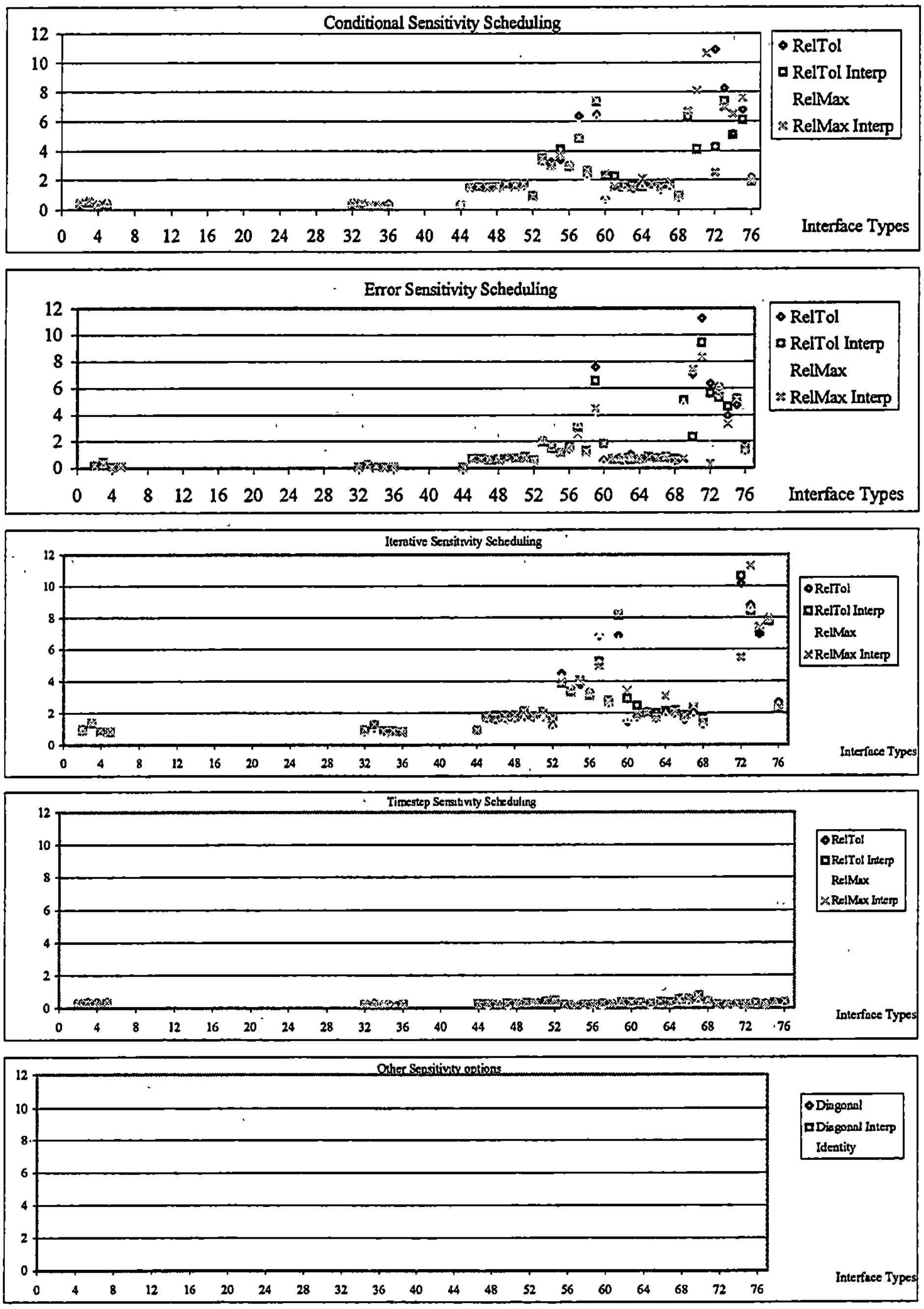


Figure D-29. Iteration results for E2E_HF_DRVVIR2

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

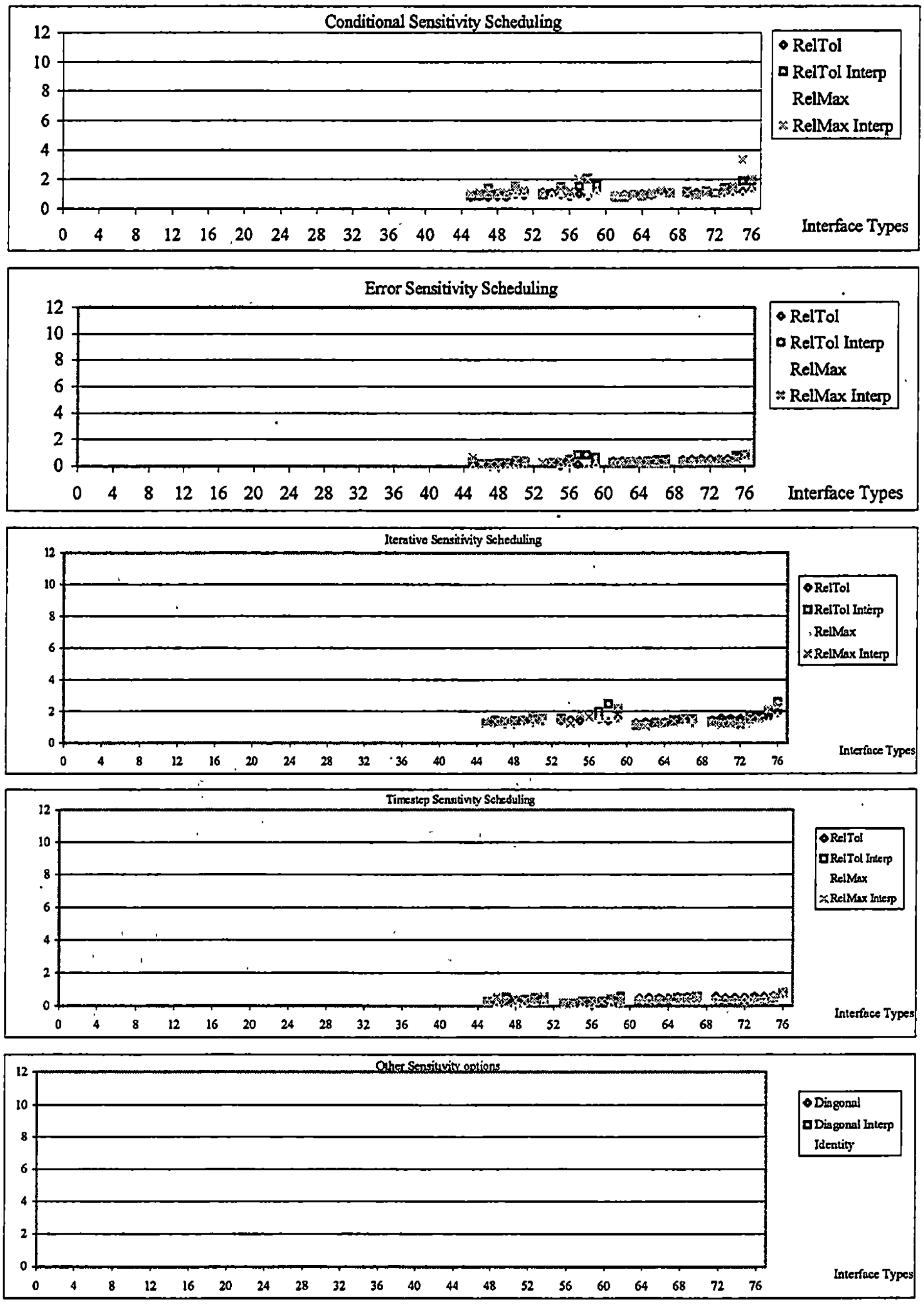


Figure D-30. Iteration results for E2E_HF_DRVVIR2a

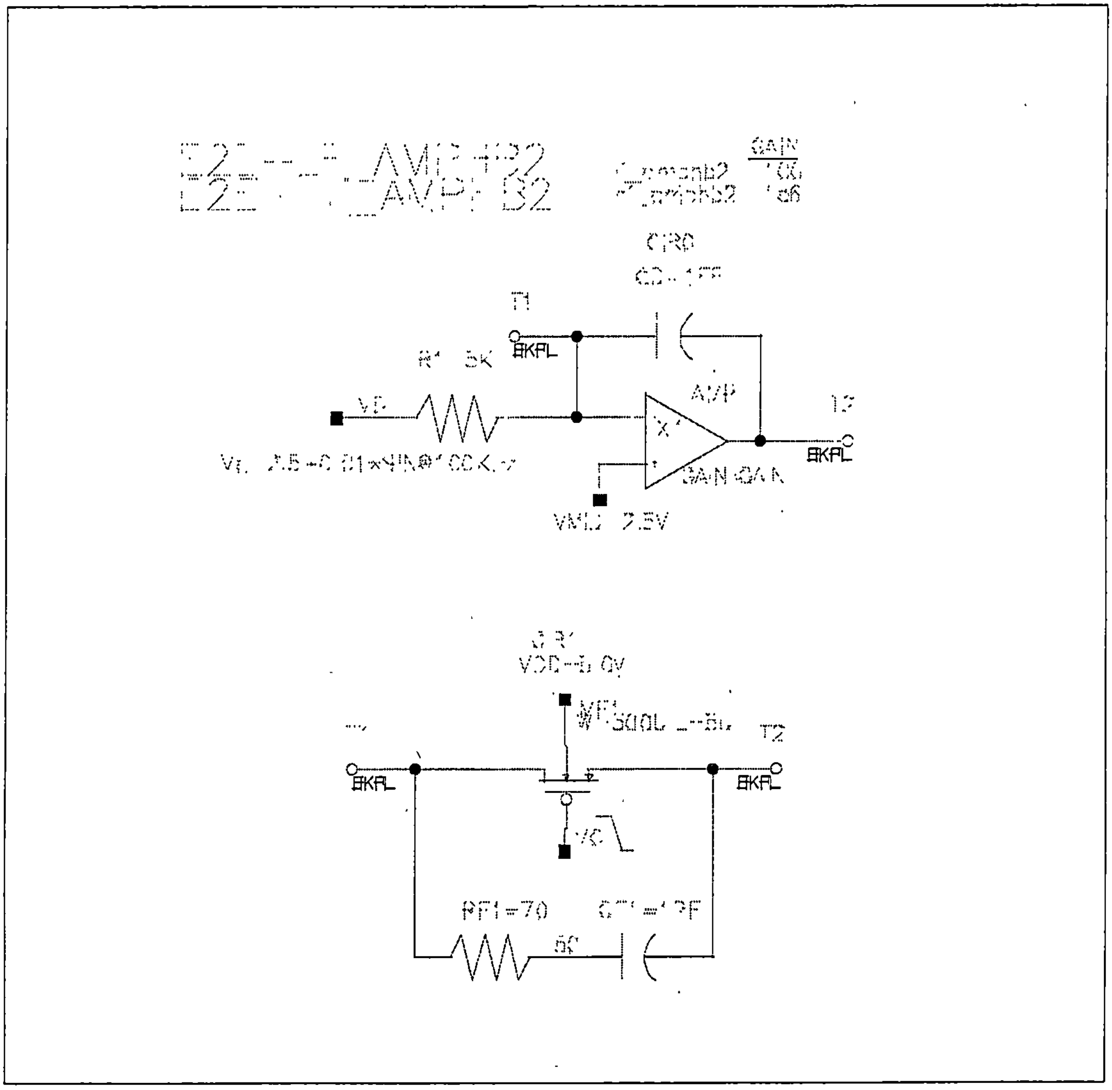


Figure D-31. E2E broken amplifier 2-terminal coupling tests

1-36: Conventional
37-44: Dynamic interfacing
45-52: Parallel Correction (ESRC)
53-60: Sequential Correction (ESRC)
61-68: Parallel Correction (ESRC_SENS)
69-76: Sequential Correction (ESRC_SENS)

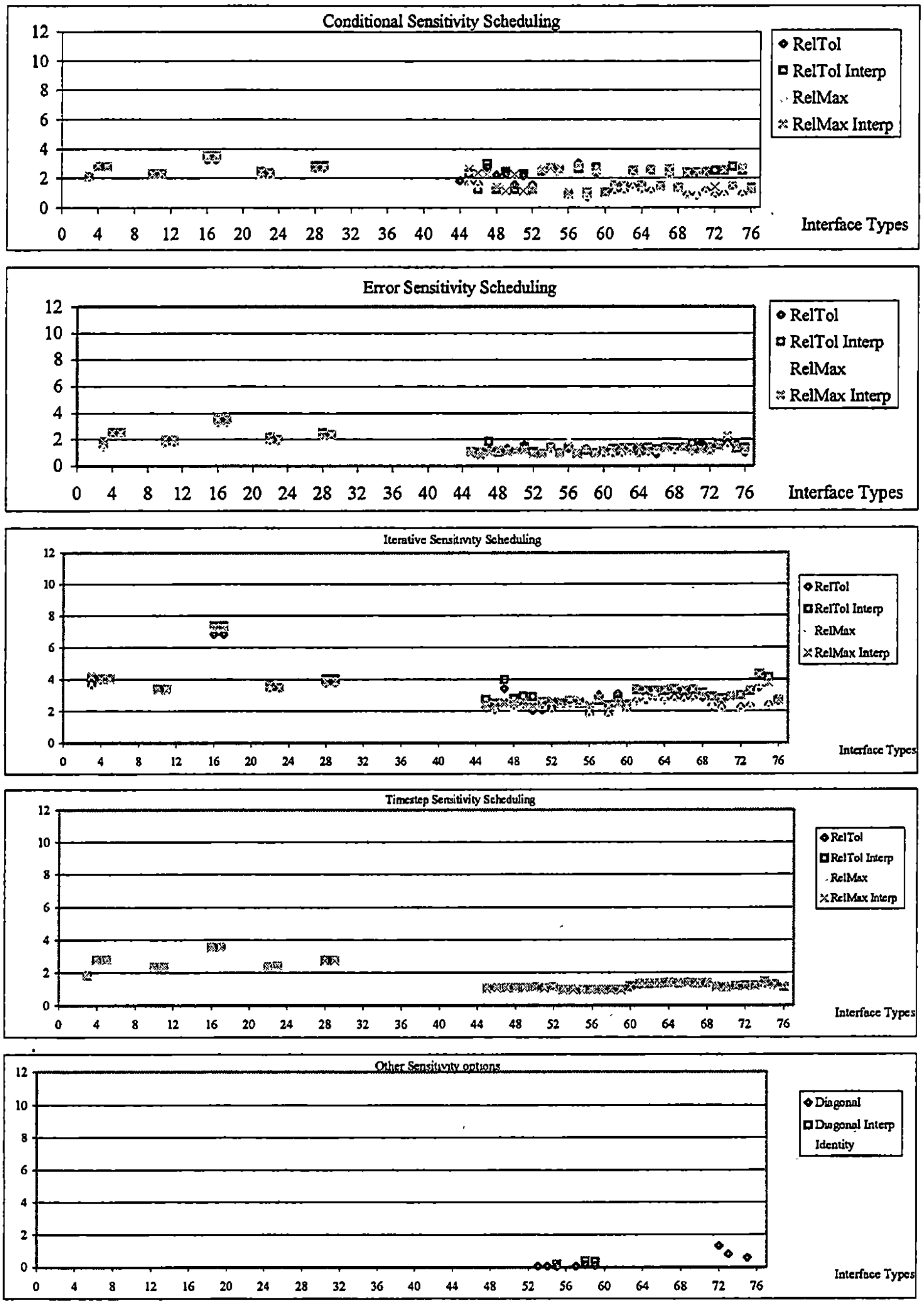


Figure D-32. Iteration results for E2E_LF_AMPHB2

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

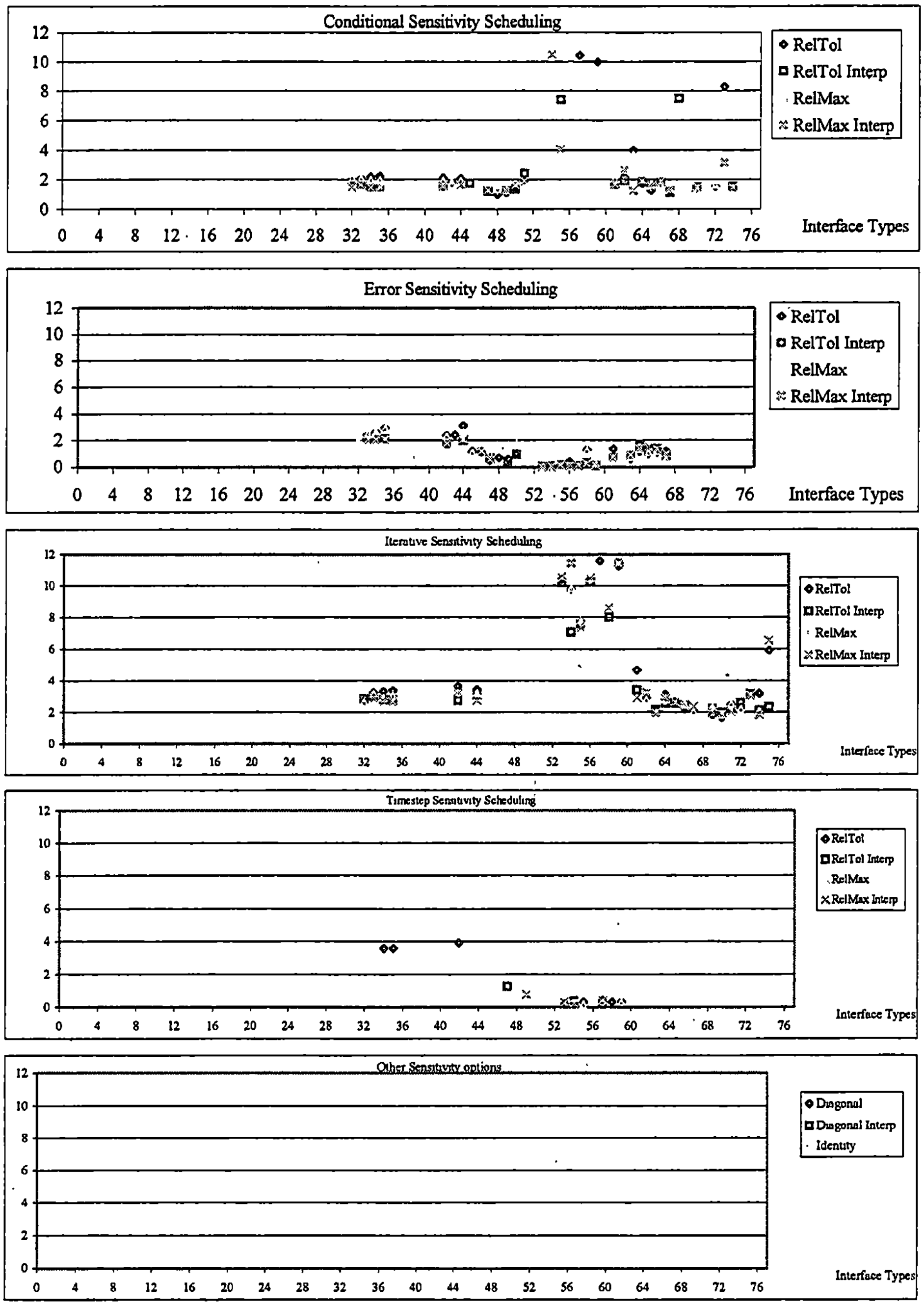


Figure D-33. Iteration results for E2E_HF_AMPHB2

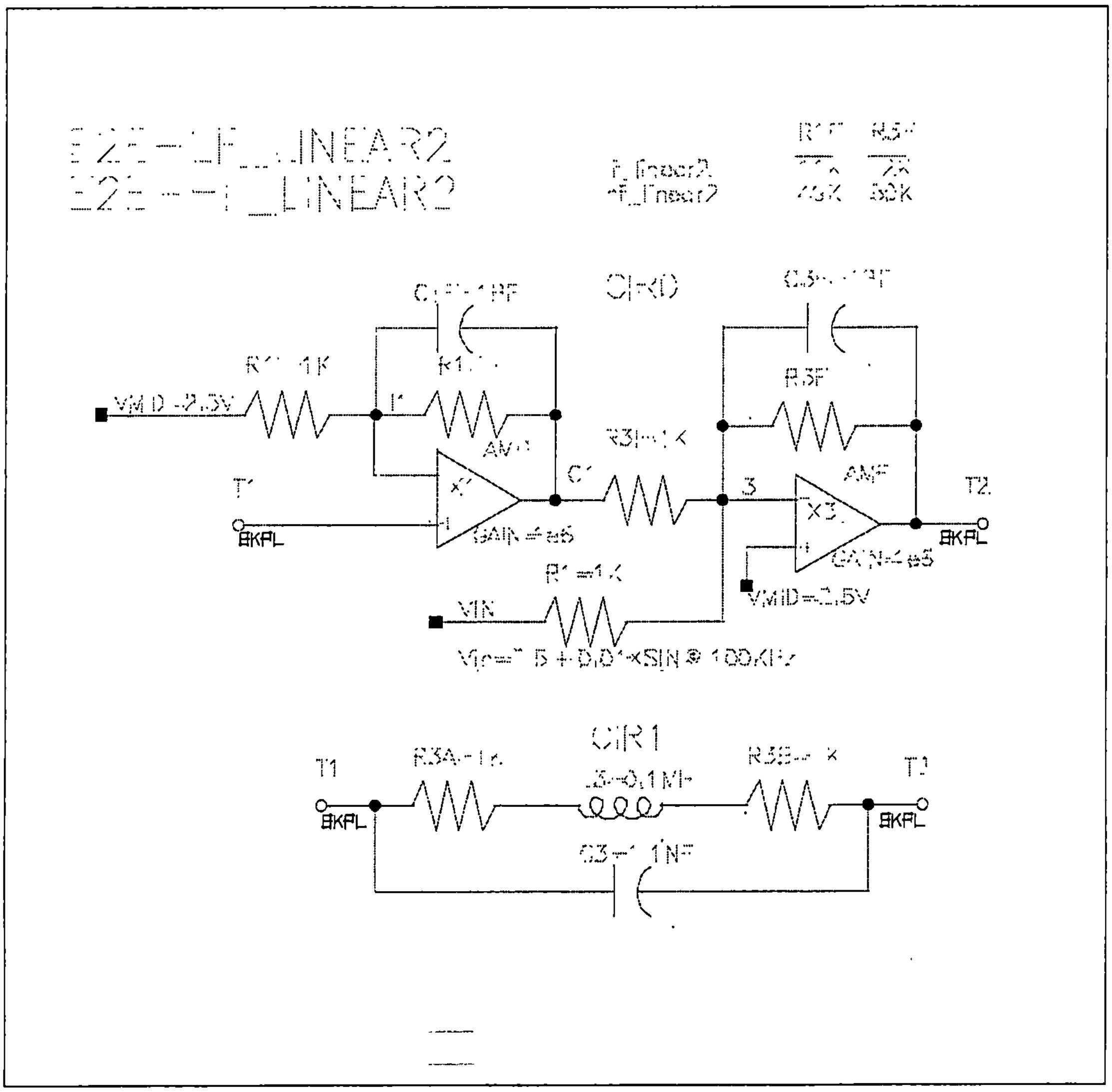


Figure D-34. E2E 2-terminal linear virtual driver example

1-36: Conventional
37-44: Dynamic interfacing
45-52: Parallel Correction (ESRC)
53-60: Sequential Correction (ESRC)
61-68: Parallel Correction (ESRC_SENS)
69-76: Sequential Correction (ESRC_SENS)

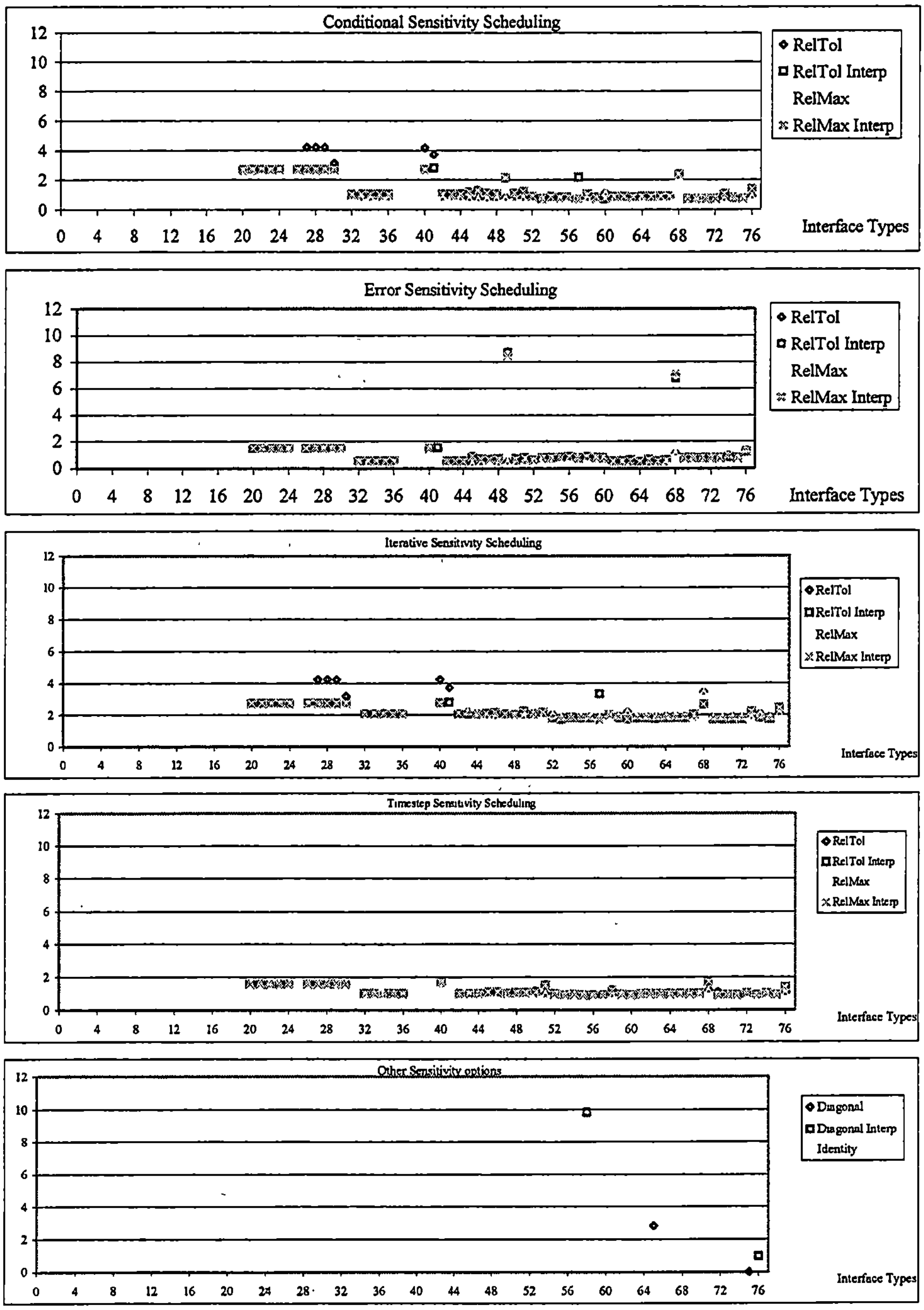


Figure D-35. Iteration results for E2E_LF_LINEAR2

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

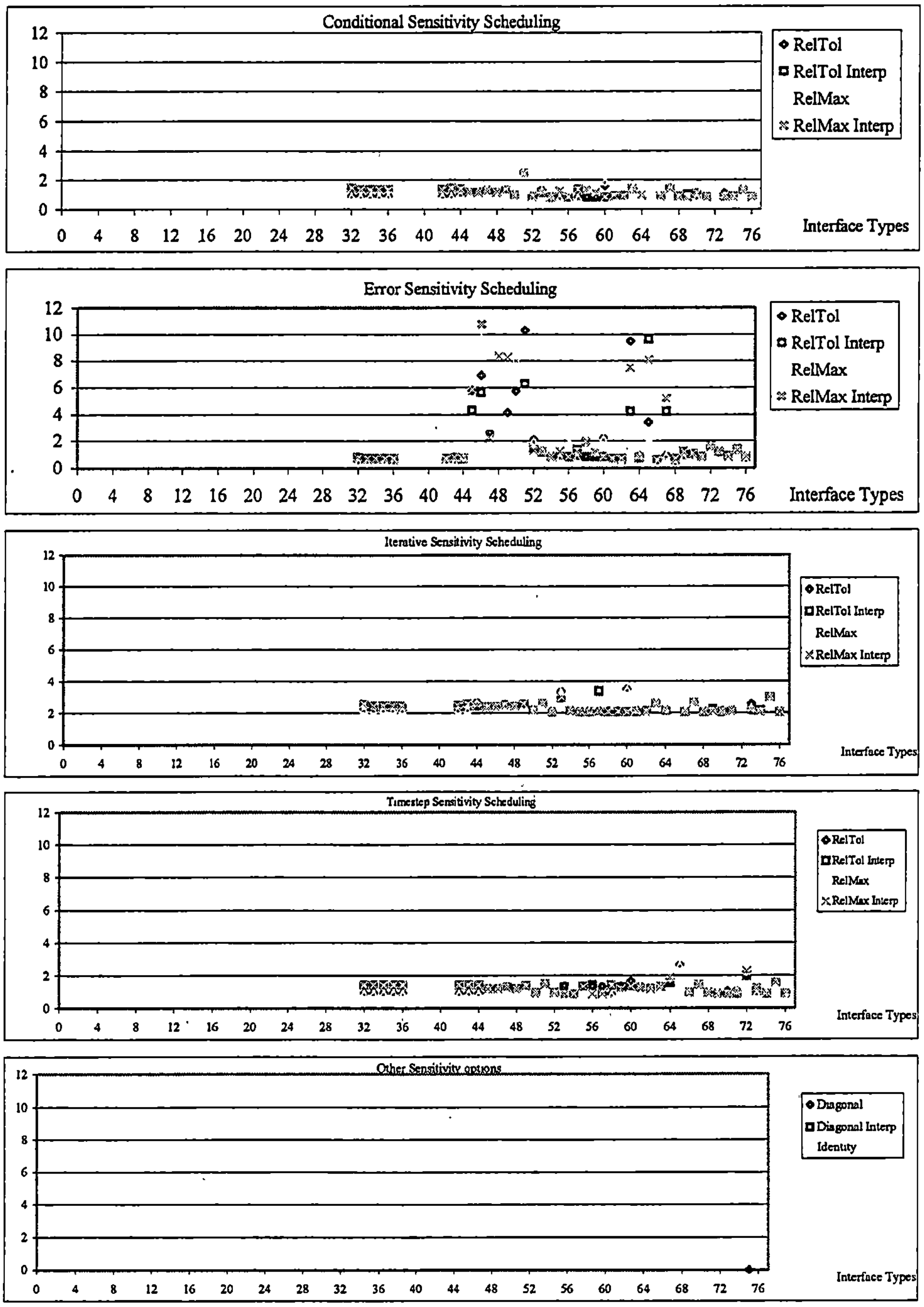


Figure D-36. Iteration results for E2E_HF_LINEAR2

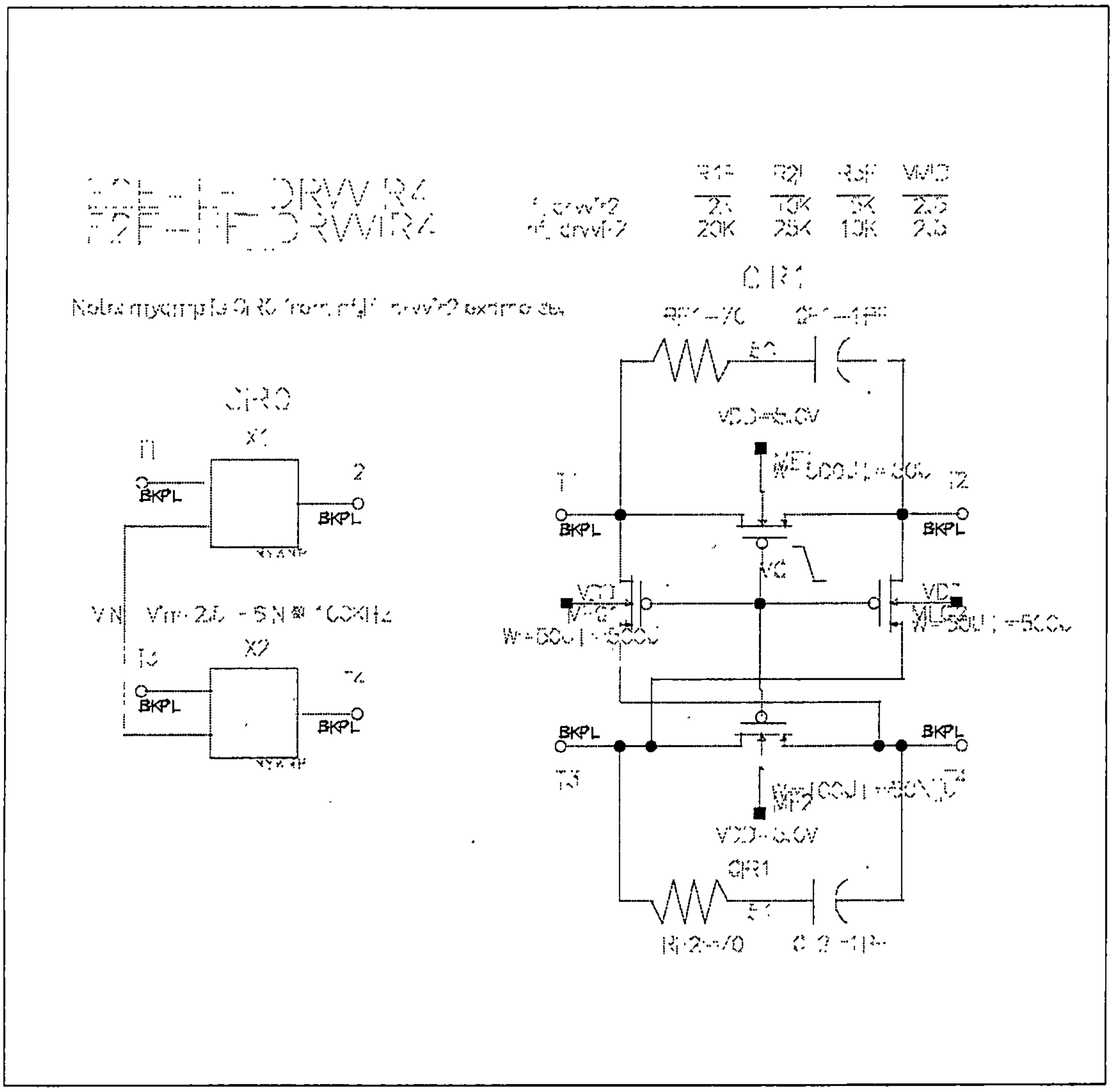
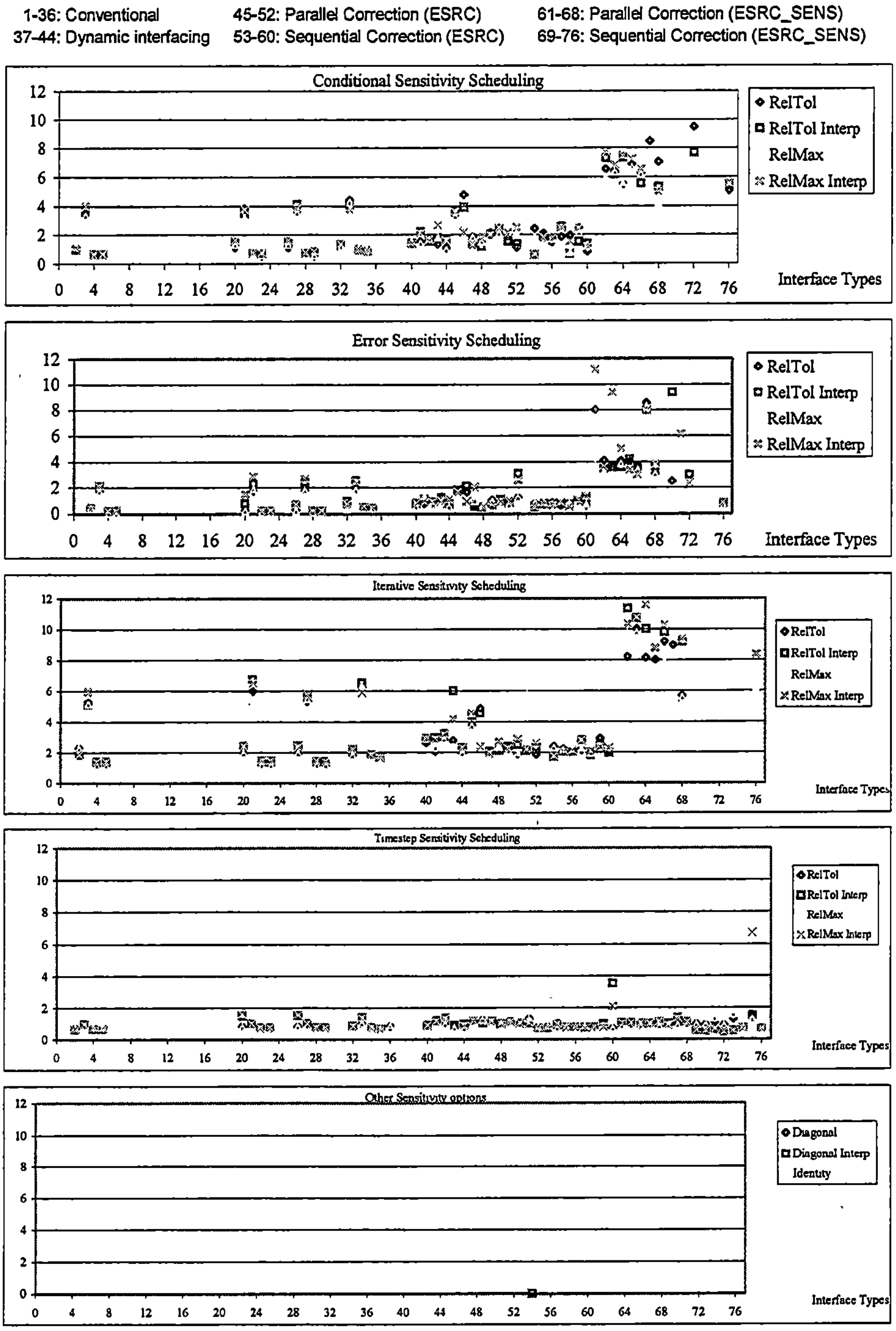


Figure D-37. E2E 4-terminal virtual driver example



1-36: Conventional
37-44: Dynamic interfacing
45-52: Parallel Correction (ESRC)
53-60: Sequential Correction (ESRC)
61-68: Parallel Correction (ESRC_SENS)
69-76: Sequential Correction (ESRC_SENS)

Figure D-38. Iteration results for E2E_LF_DRVVIR4

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

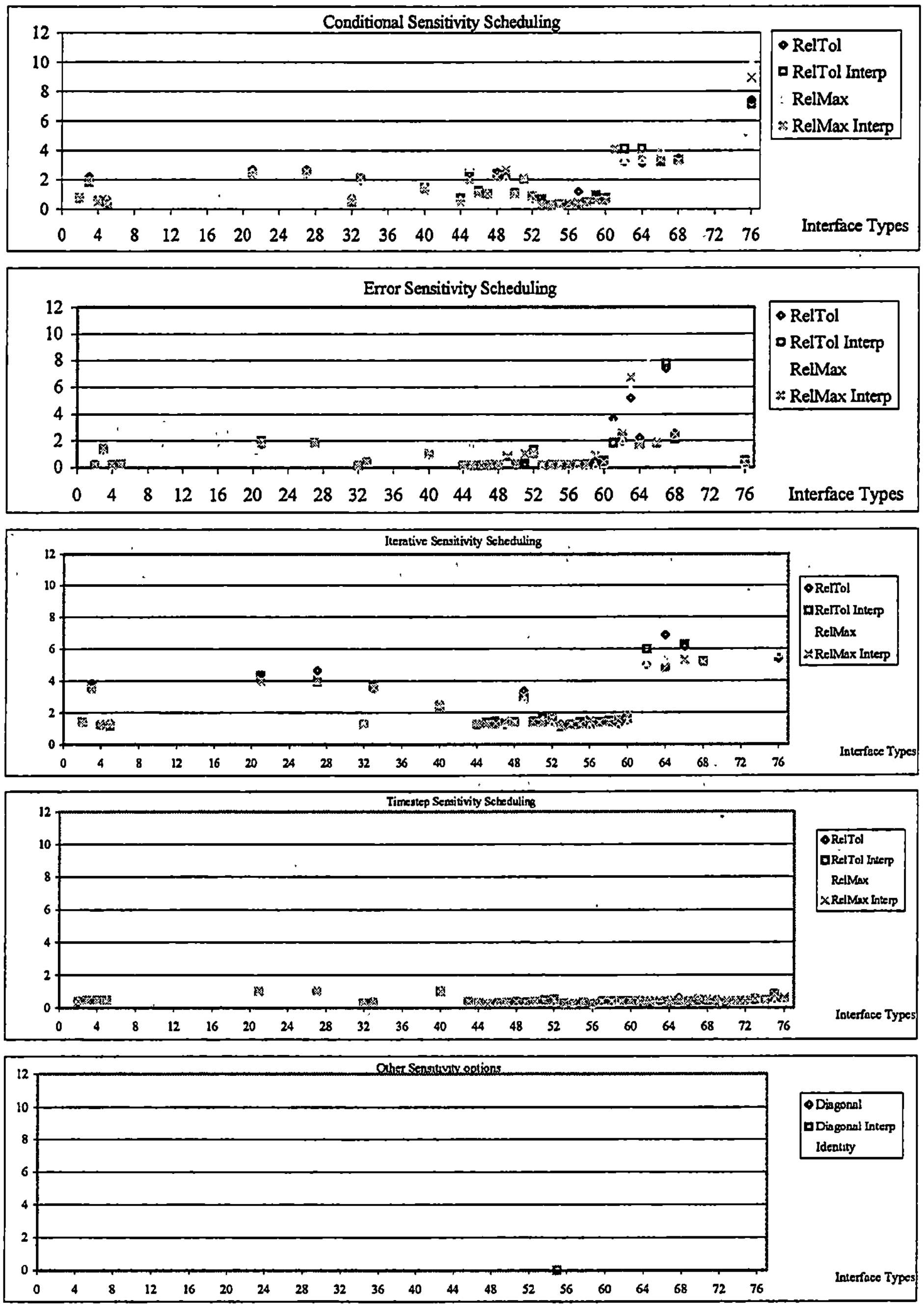


Figure D-39. Iteration results for E2E_HF_DRVVIR4

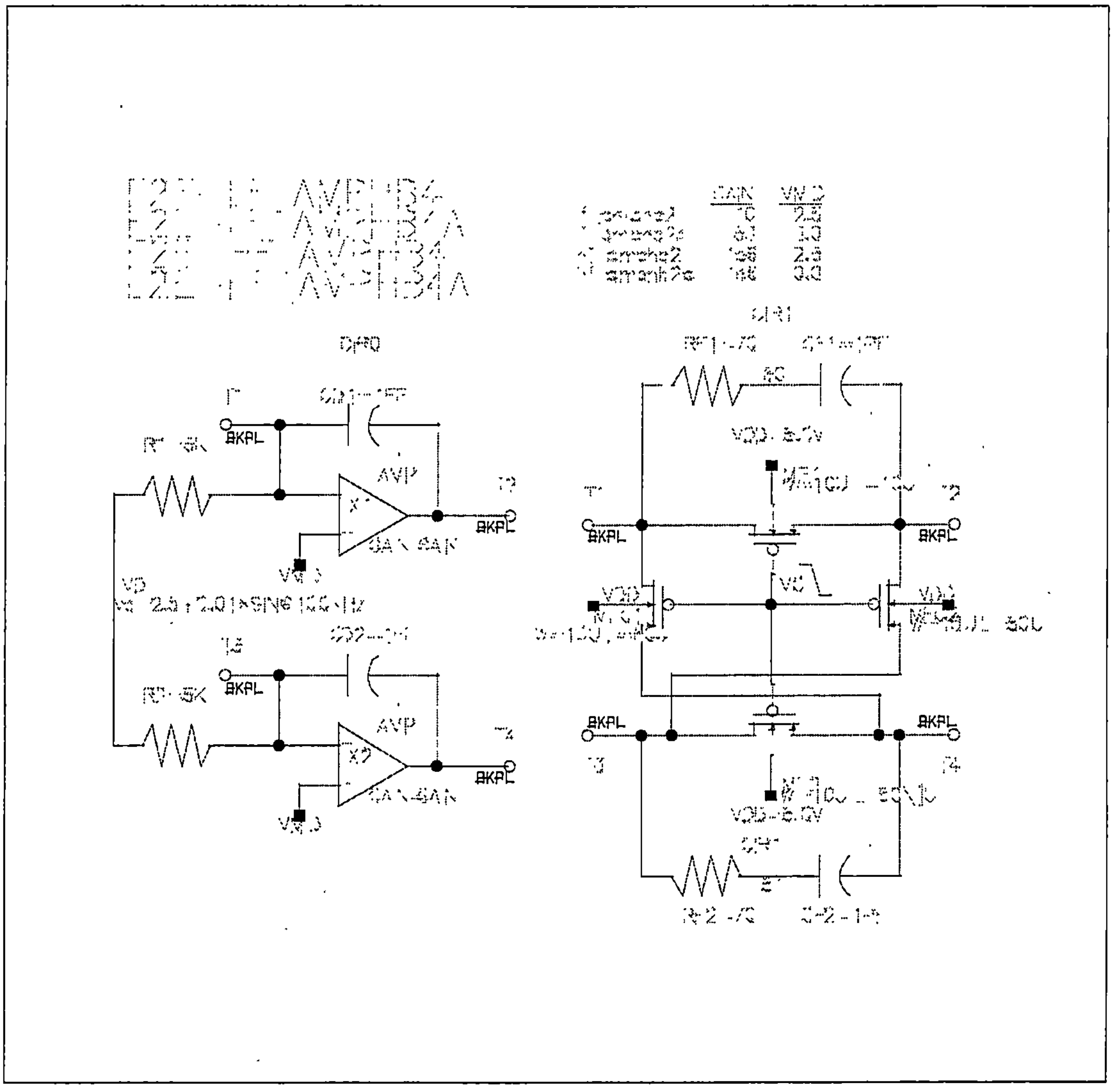


Figure D-40. E2E broken-amplifier 4-terminal tests

1-36: Conventional
37-44: Dynamic interfacing
45-52: Parallel Correction (ESRC)
53-60: Sequential Correction (ESRC)
61-68: Parallel Correction (ESRC_SENS)
69-76: Sequential Correction (ESRC_SENS)

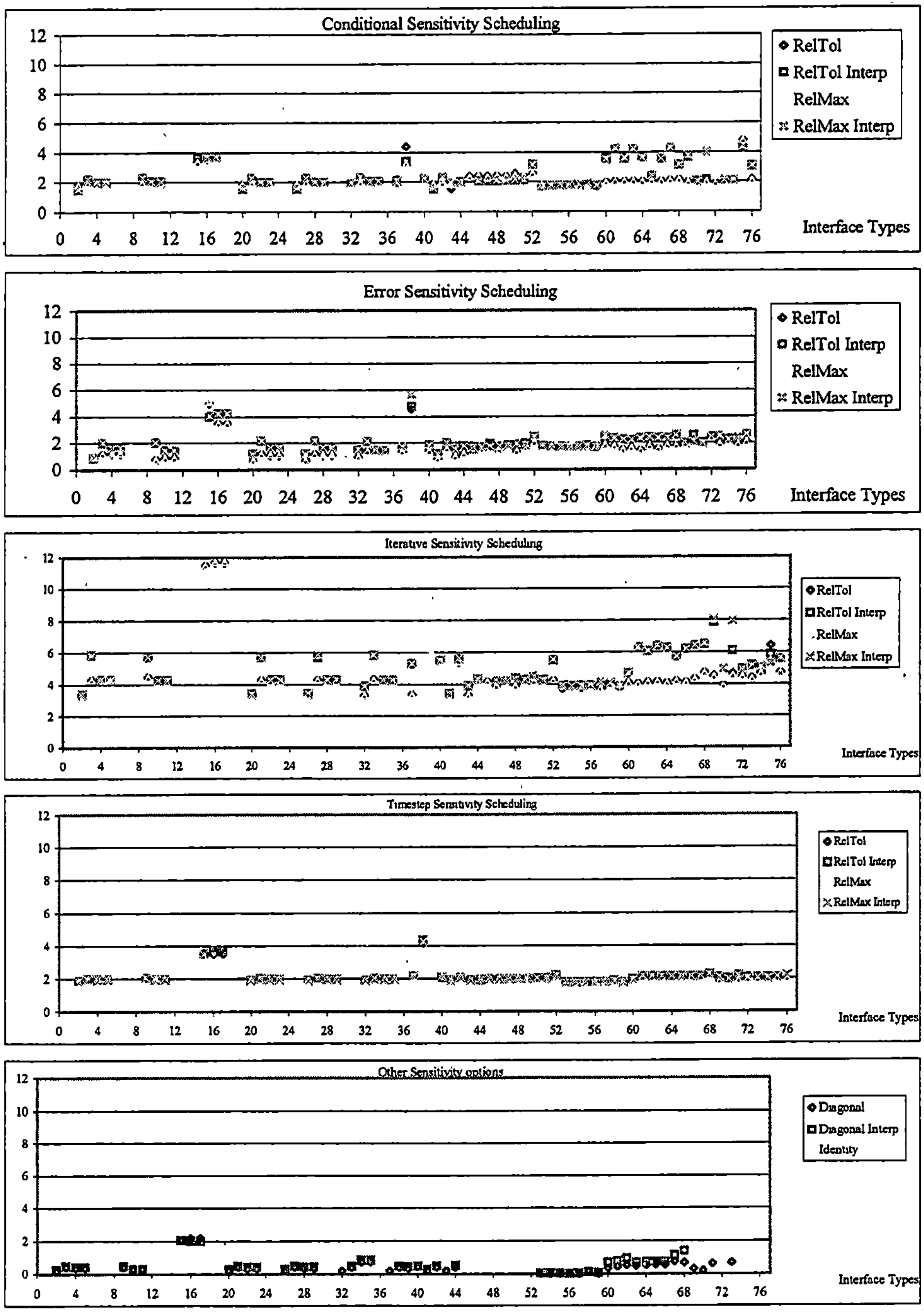


Figure D-41. Iteration results for E2E_LF_AMPHB4

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

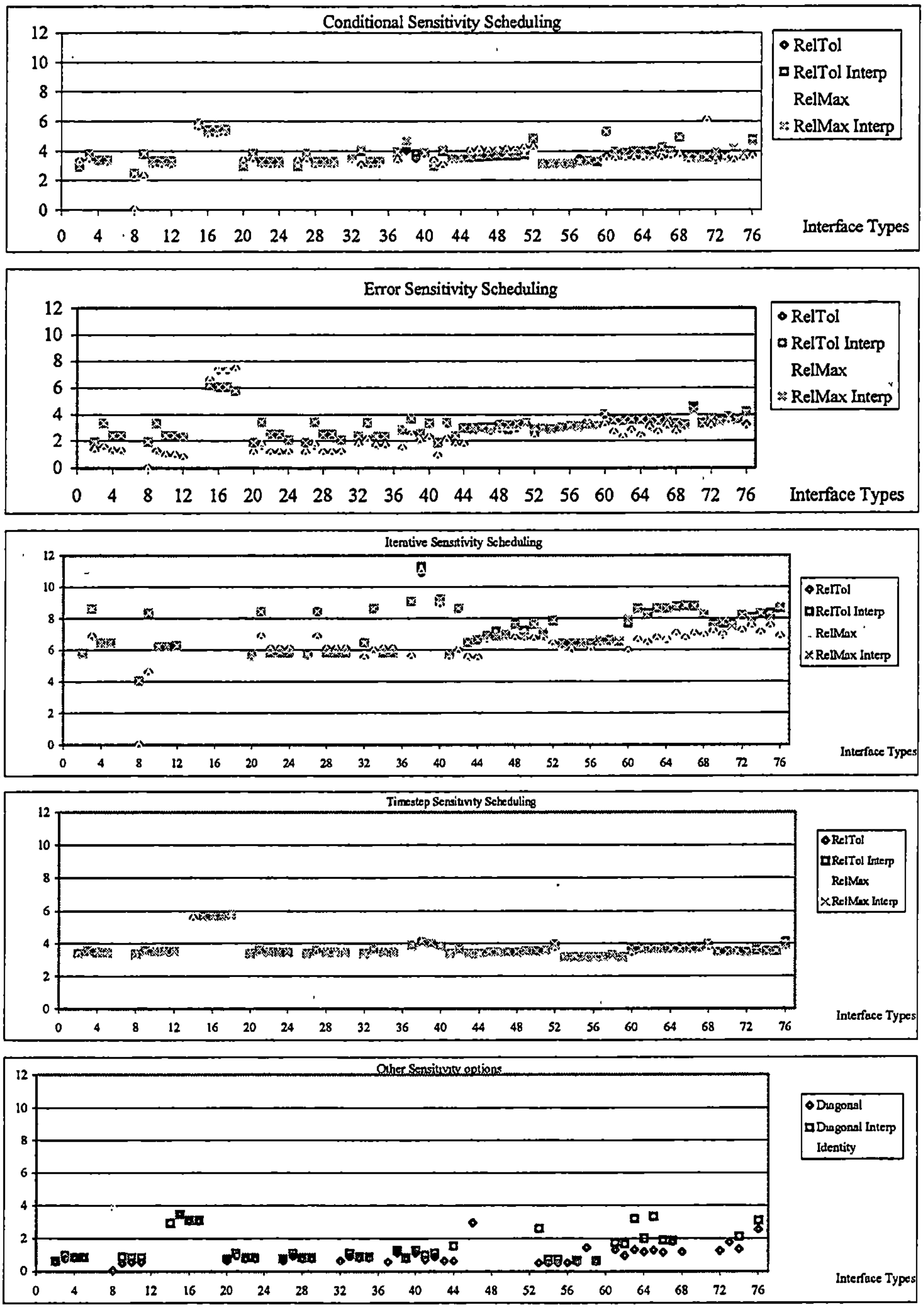


Figure D-42. Iteration results for E2E_LF_AMPHB4a

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

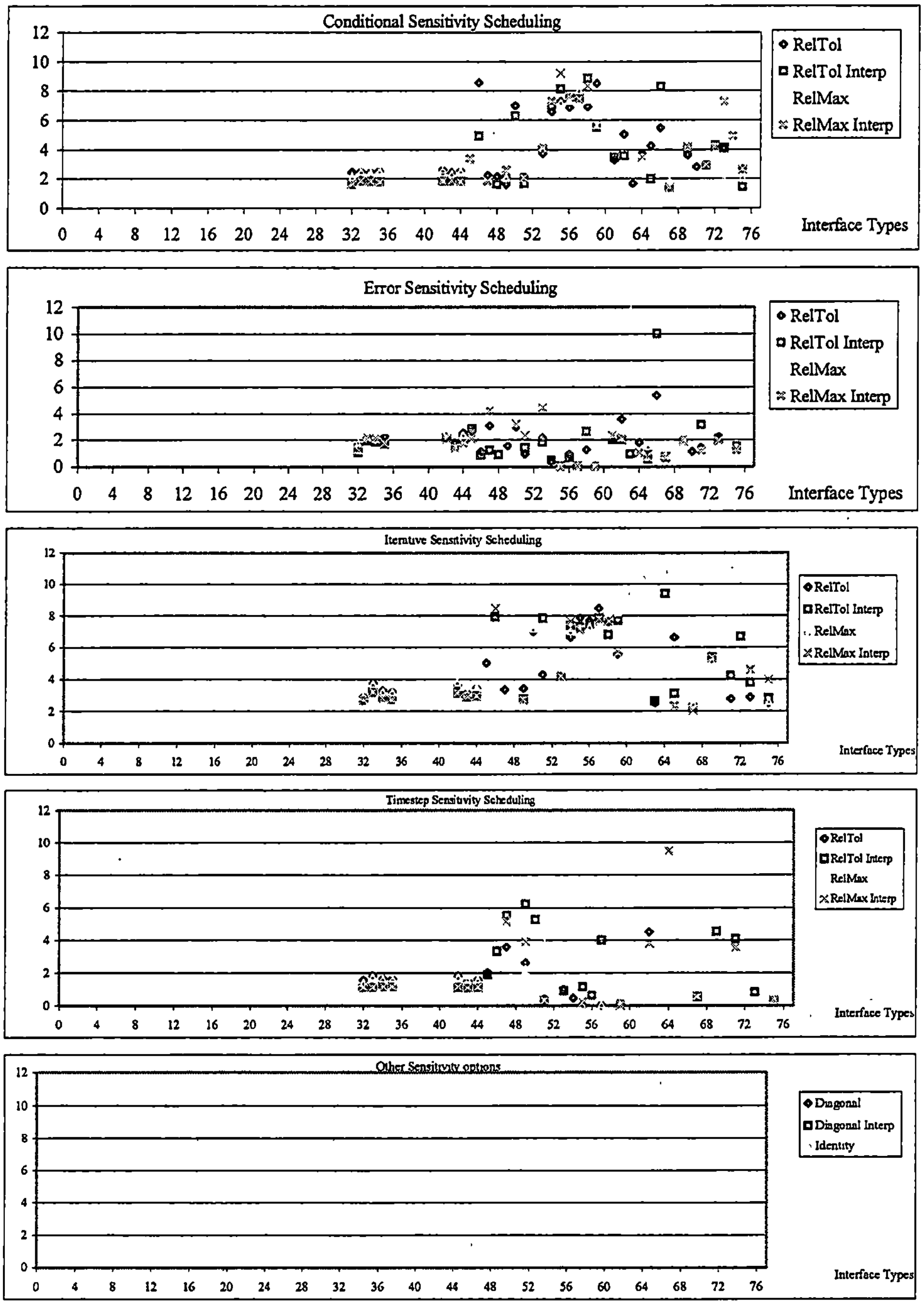


Figure D-43. Iteration results for E2E_HF_AMPHB4

1-36: Conventional
37-44: Dynamic interfacing
45-52: Parallel Correction (ESRC)
53-60: Sequential Correction (ESRC)
61-68: Parallel Correction (ESRC_SENS)
69-76: Sequential Correction (ESRC_SENS)

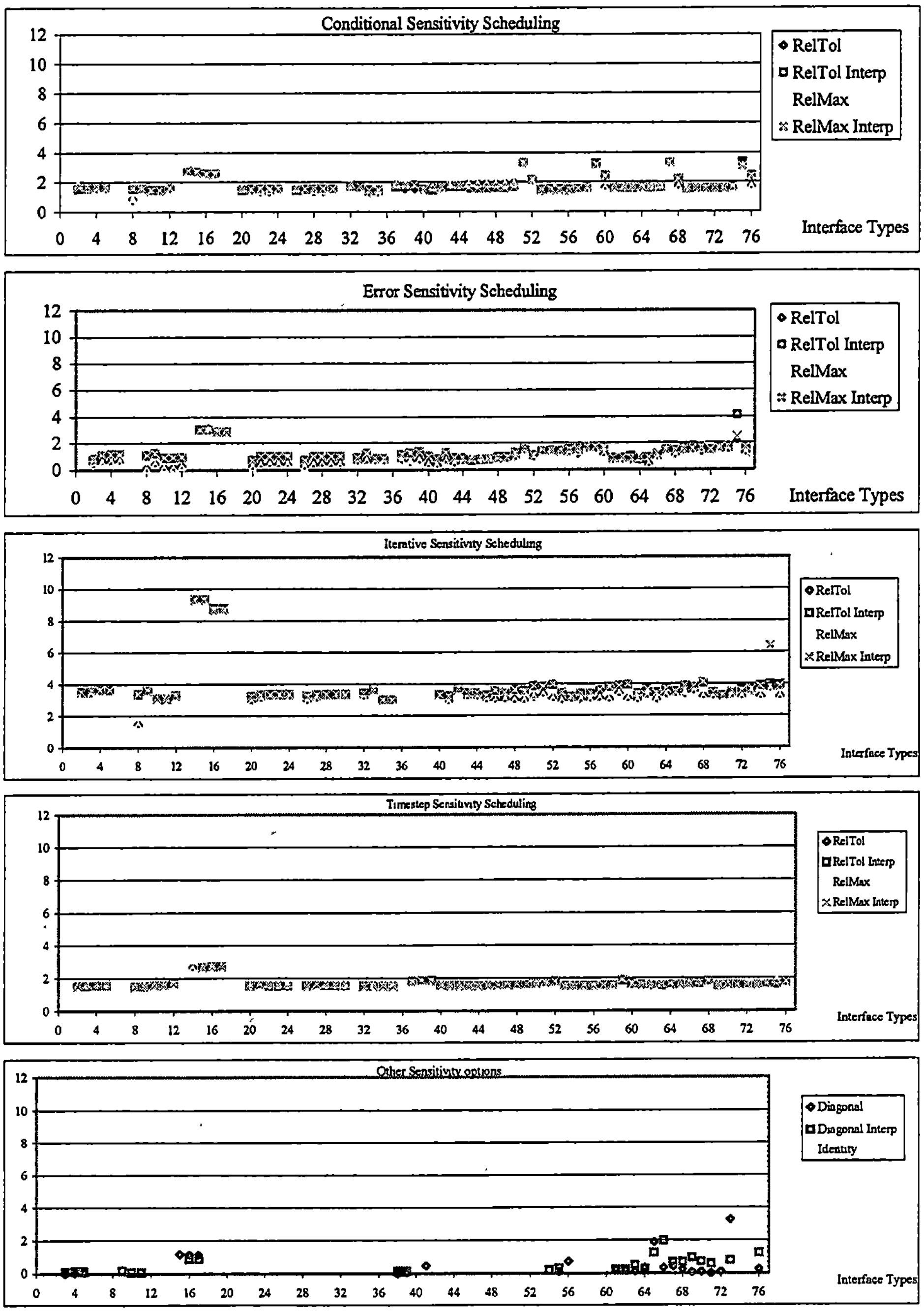


Figure D-44. Iteration results for E2E_HF_AMPHB4a

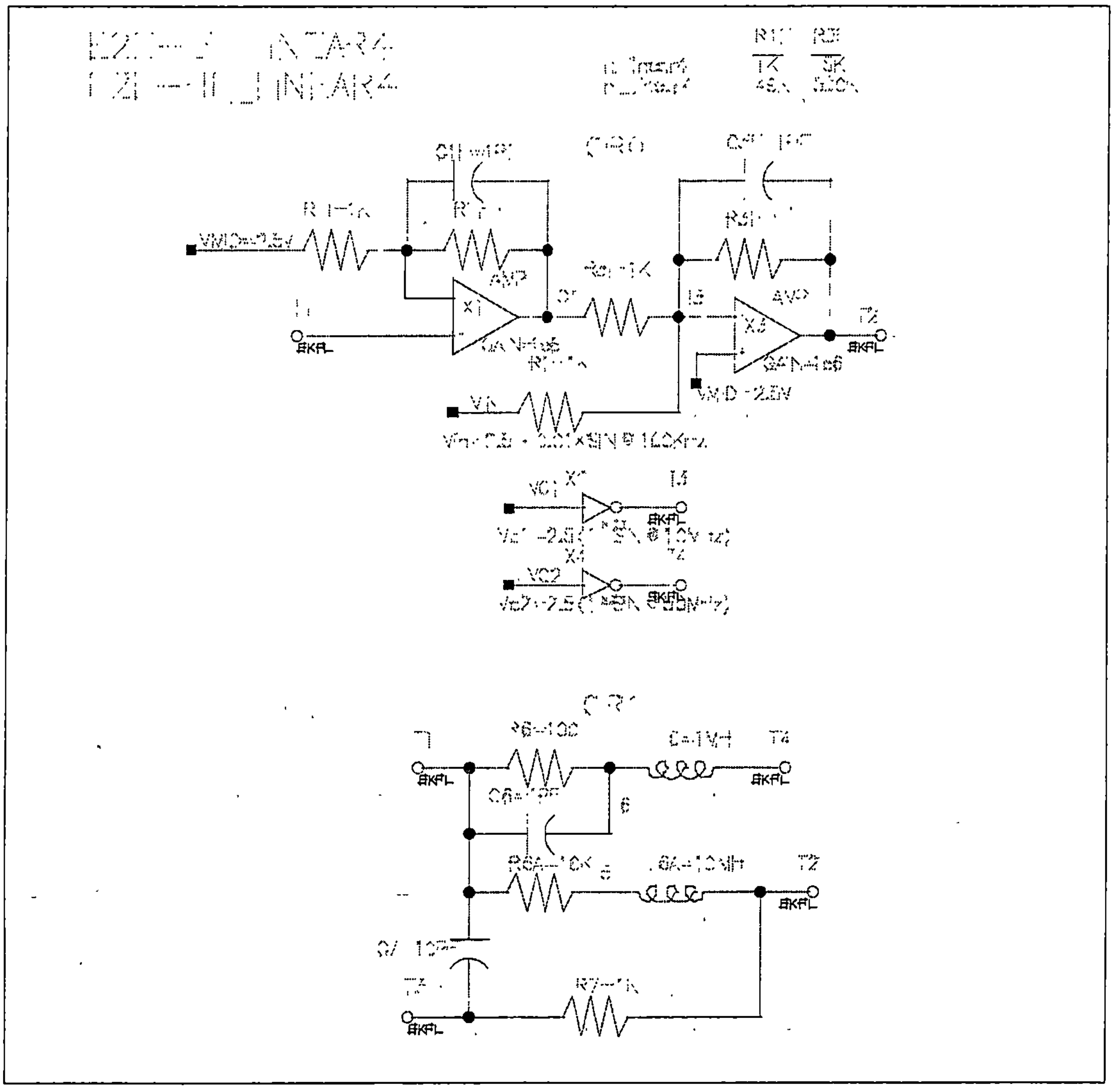


Figure D-45. E2E 4-terminal linear virtual driver example

1-36: Conventional
37-44: Dynamic interfacing

45-52: Parallel Correction (ESRC)
53-60: Sequential Correction (ESRC)

61-68: Parallel Correction (ESRC_SENS)
69-76: Sequential Correction (ESRC_SENS)

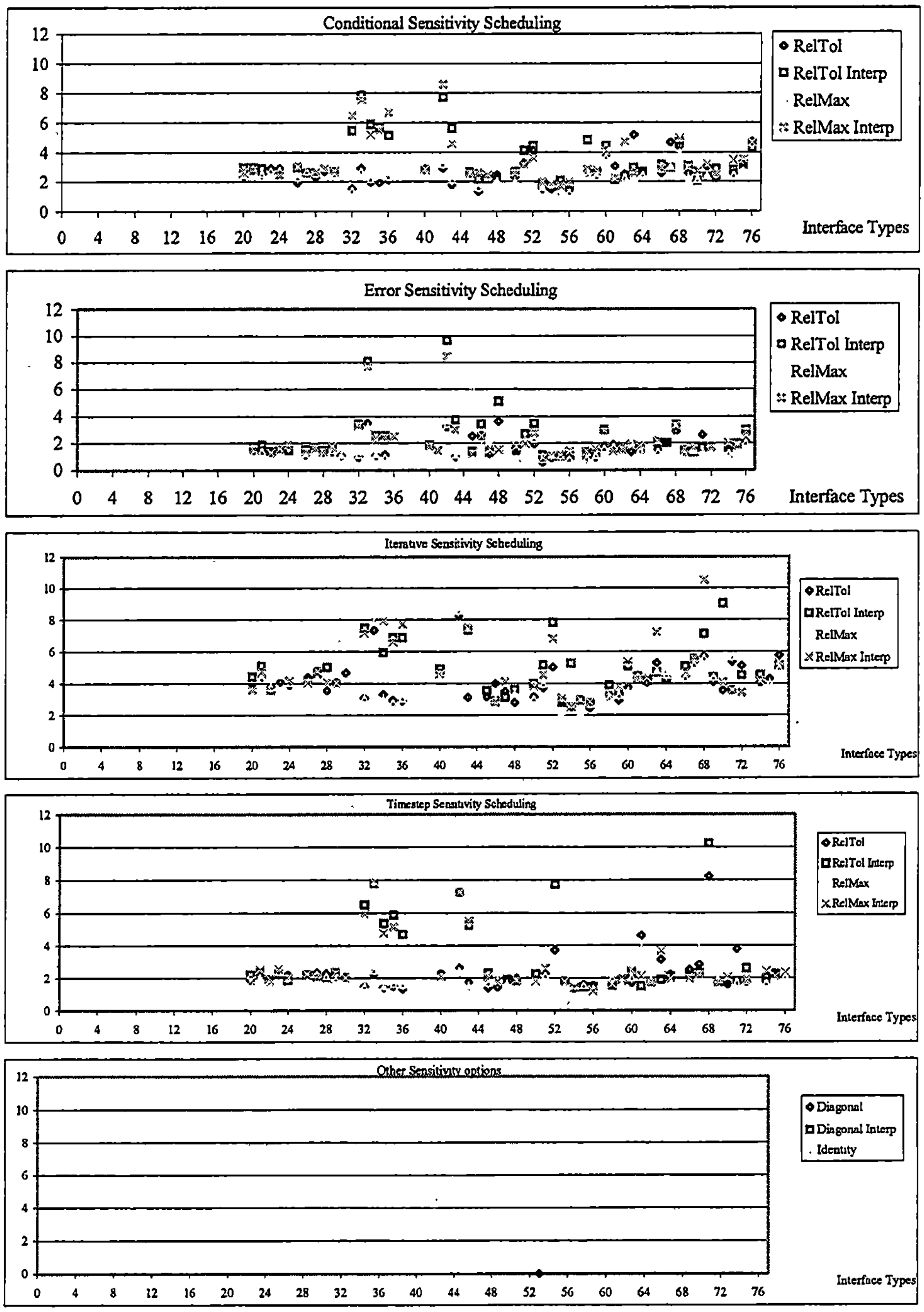


Figure D-46. Iteration results for E2E_LF_LINEAR4

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

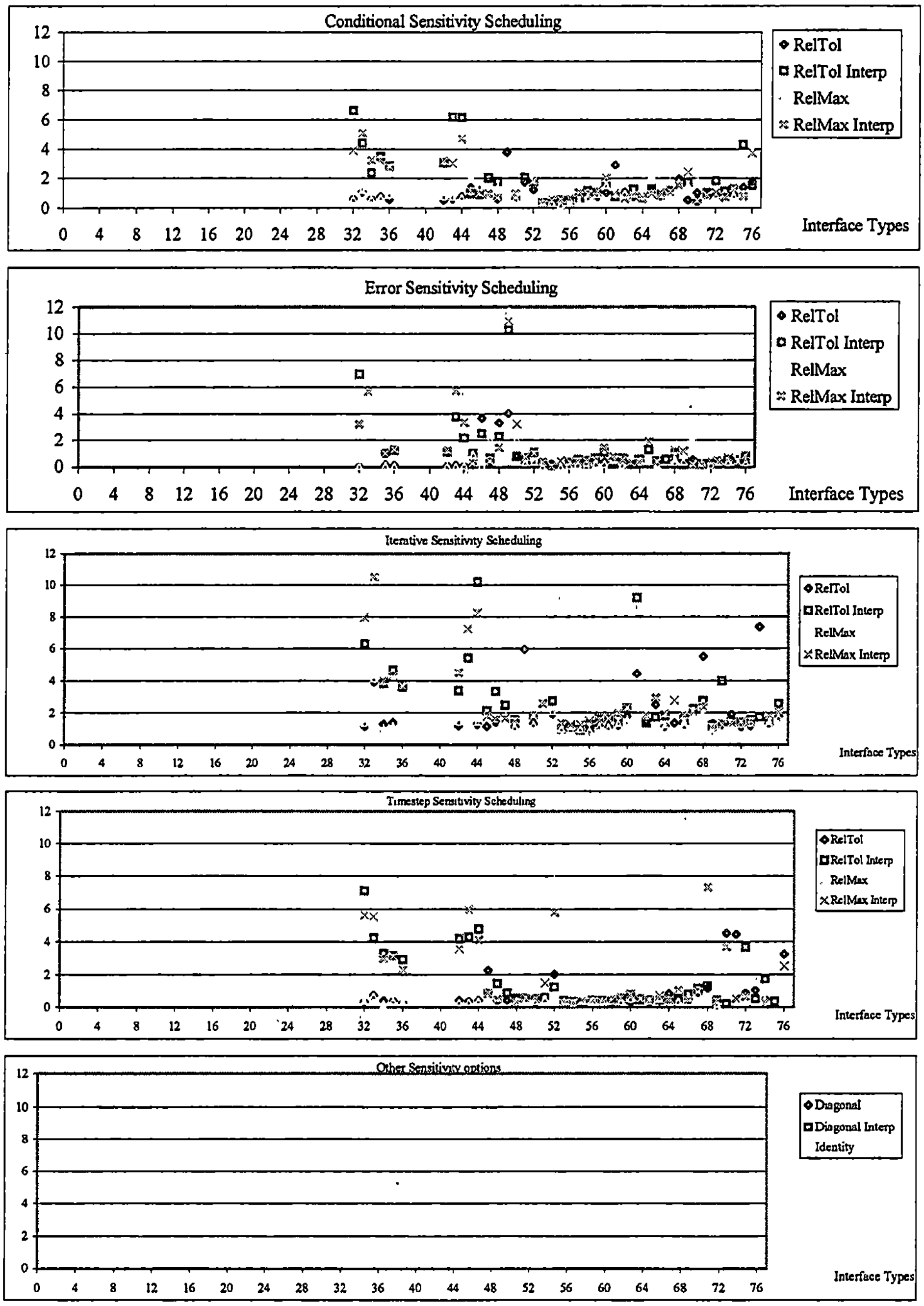


Figure D-47. Iteration results for E2E_HF_LINEAR4

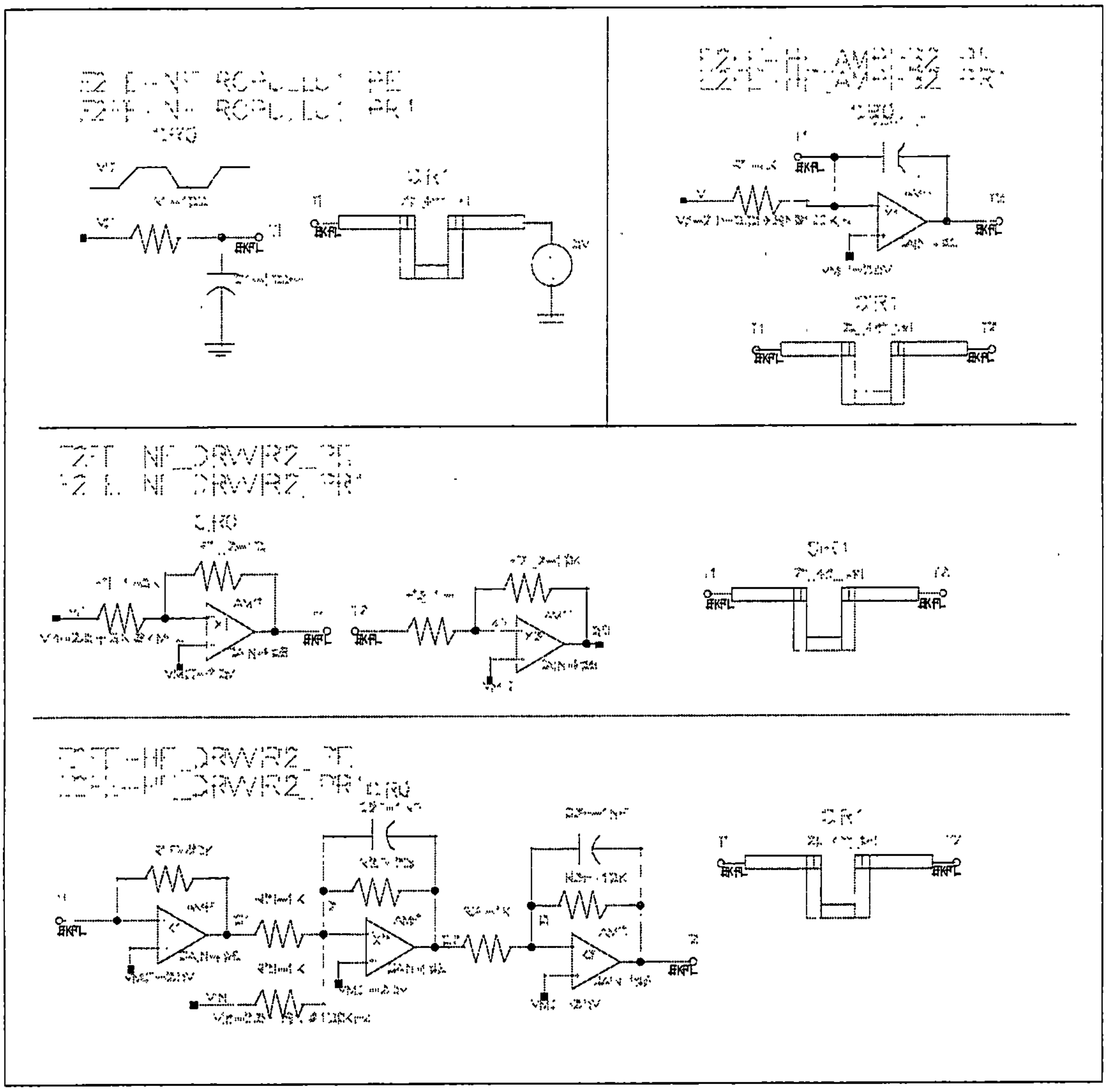


Figure D-48. E2FE 1and 2-terminal coupling tests

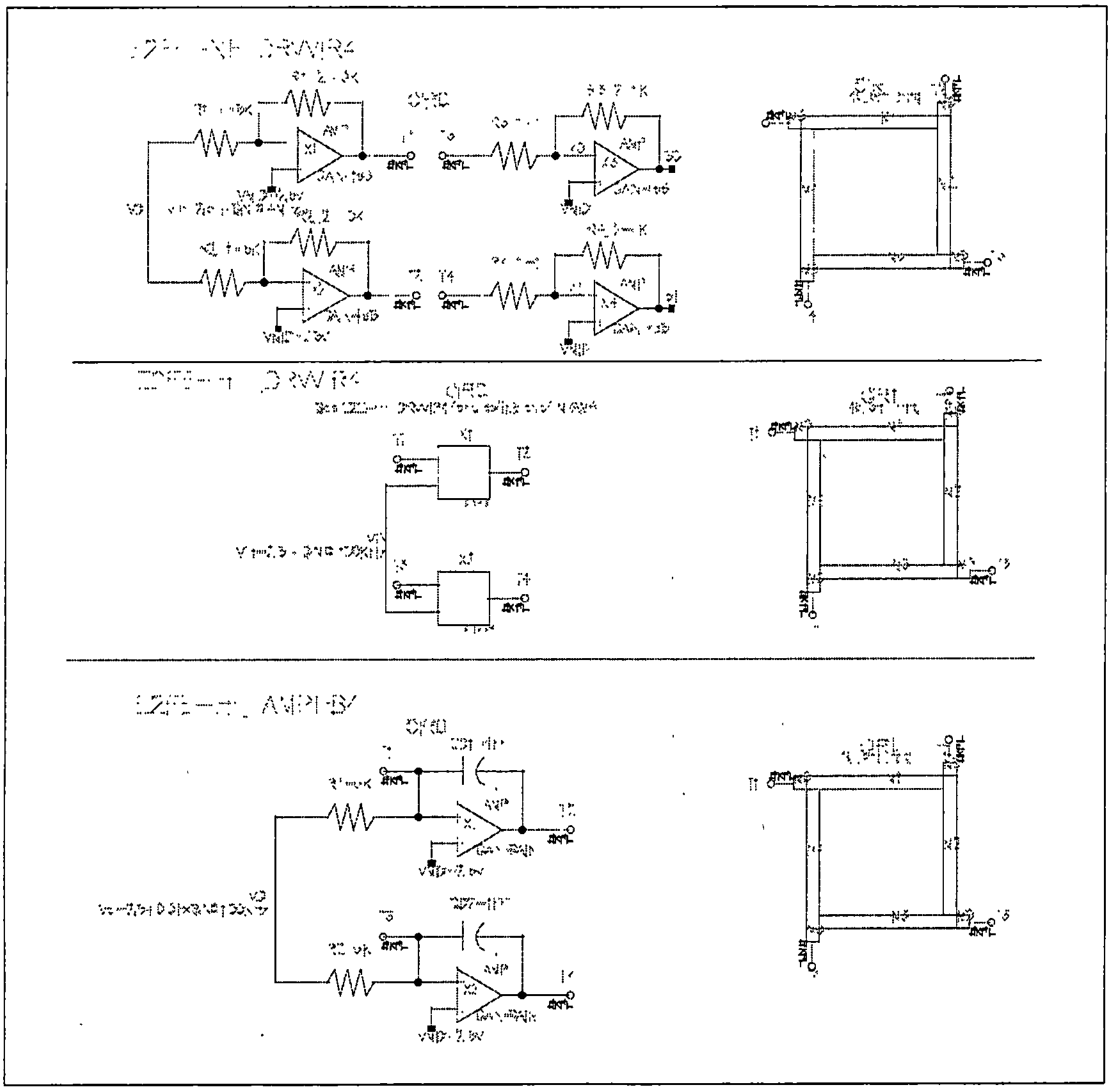


Figure D-49. E2FE 4-terminal coupling tests

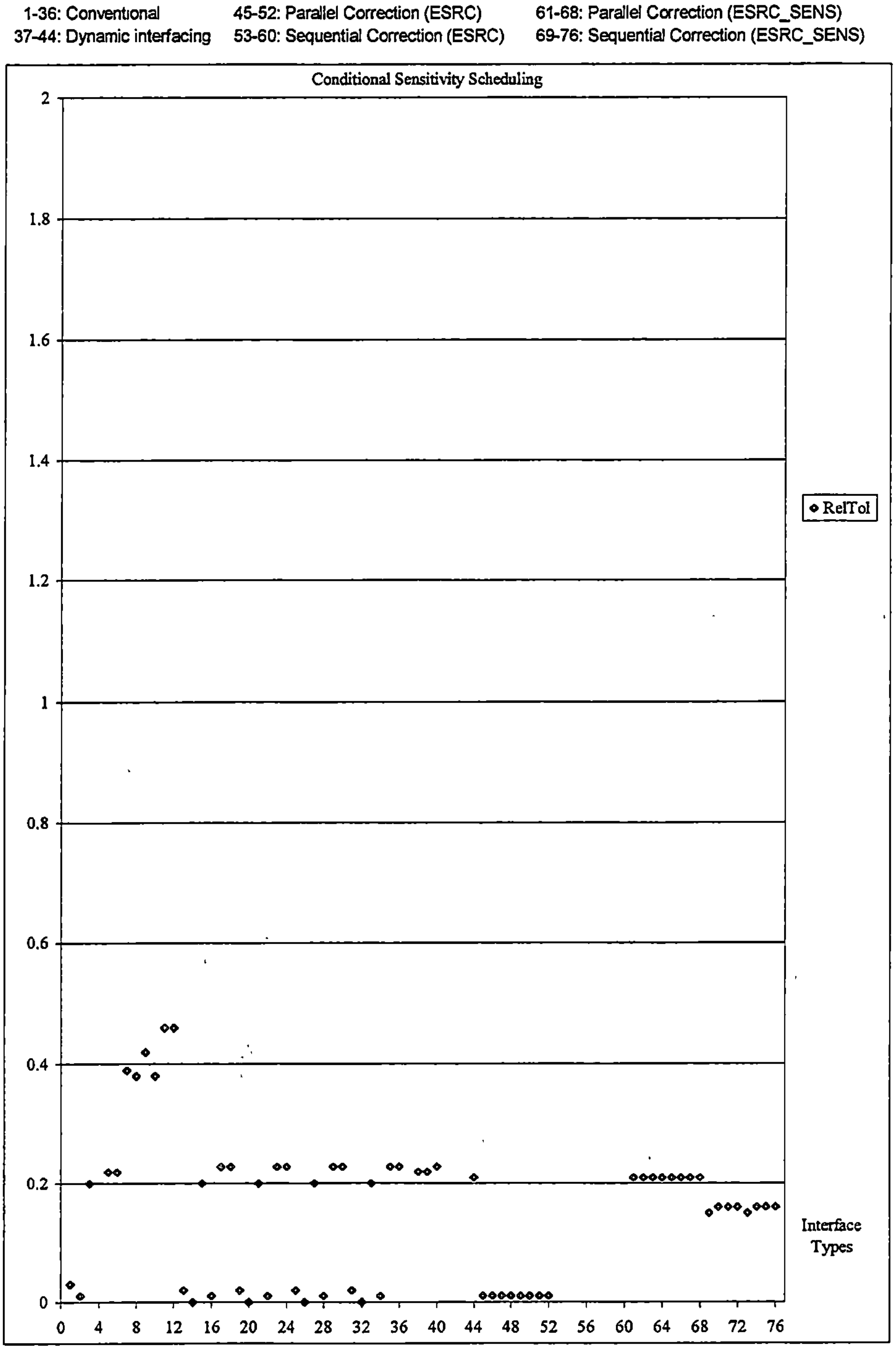


Figure D-50. Iteration results for E2FE_NF_RCPULLU1_PE

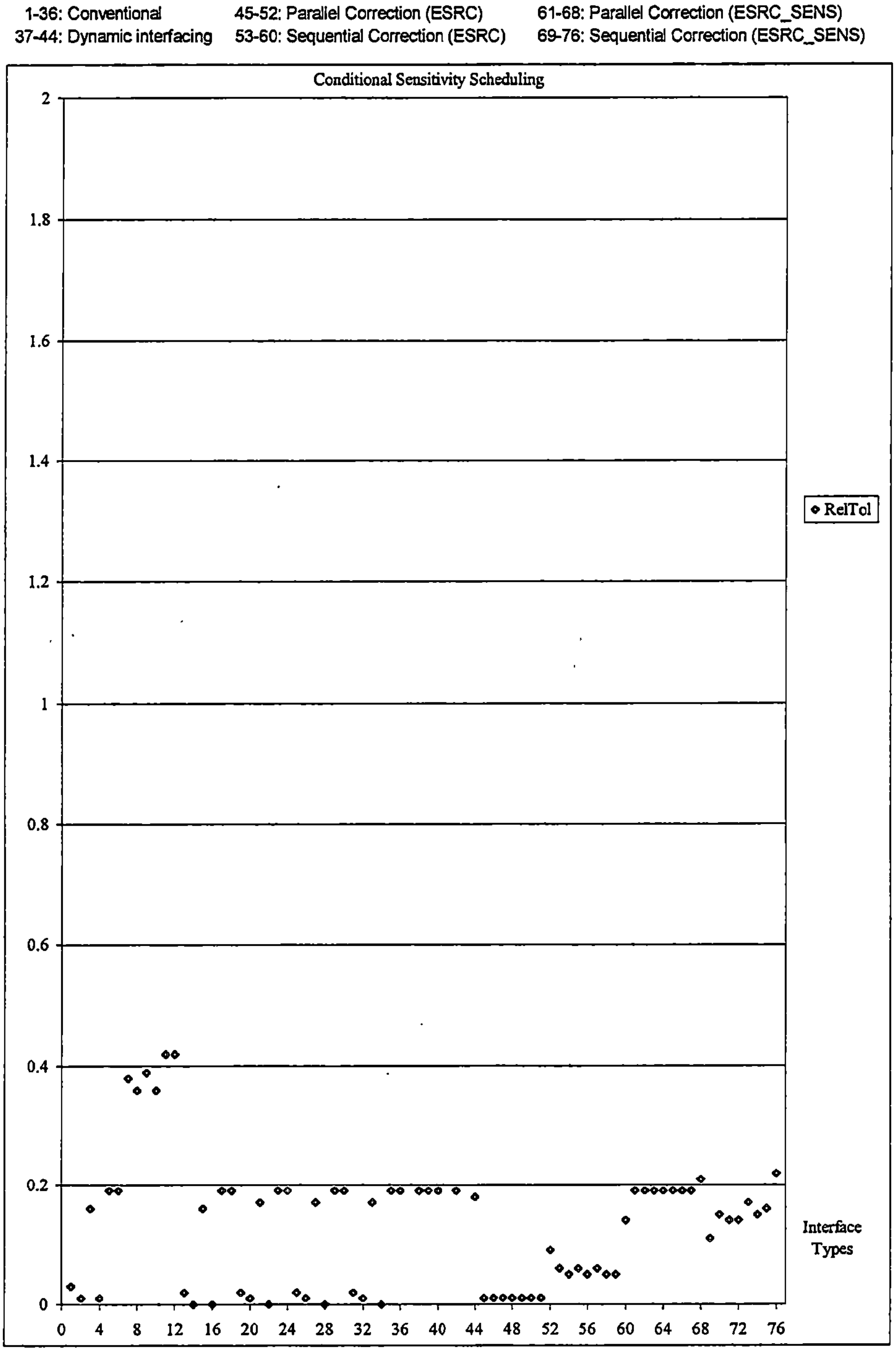


Figure D-51. Iteration results for E2FE_NF_RCPULLU1_PR1

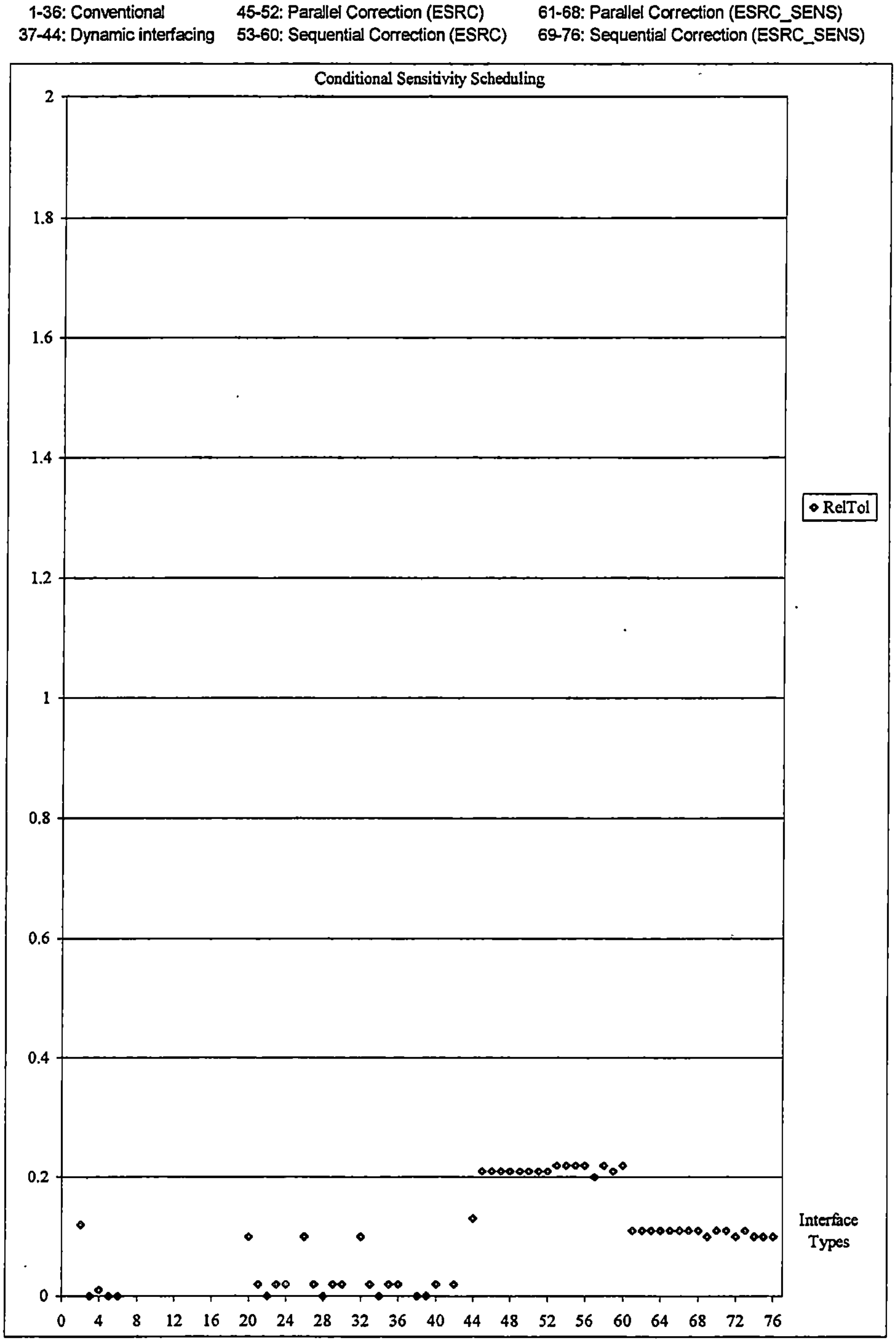


Figure D-52. Iteration results for E2FE_NF_DRVVIR2_PE

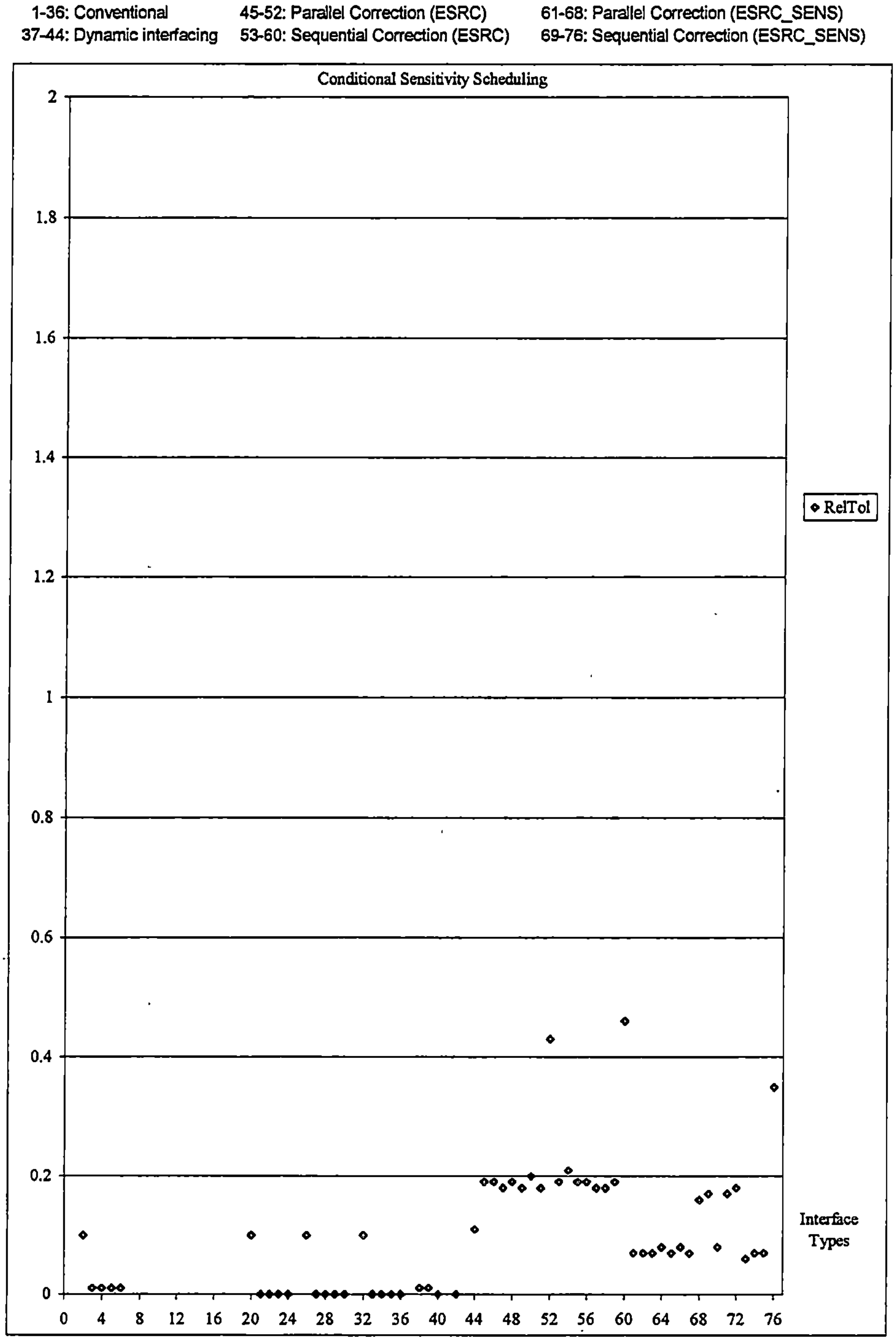


Figure D-53. Iteration results for E2FE_NF_DRVVIR2_PR1

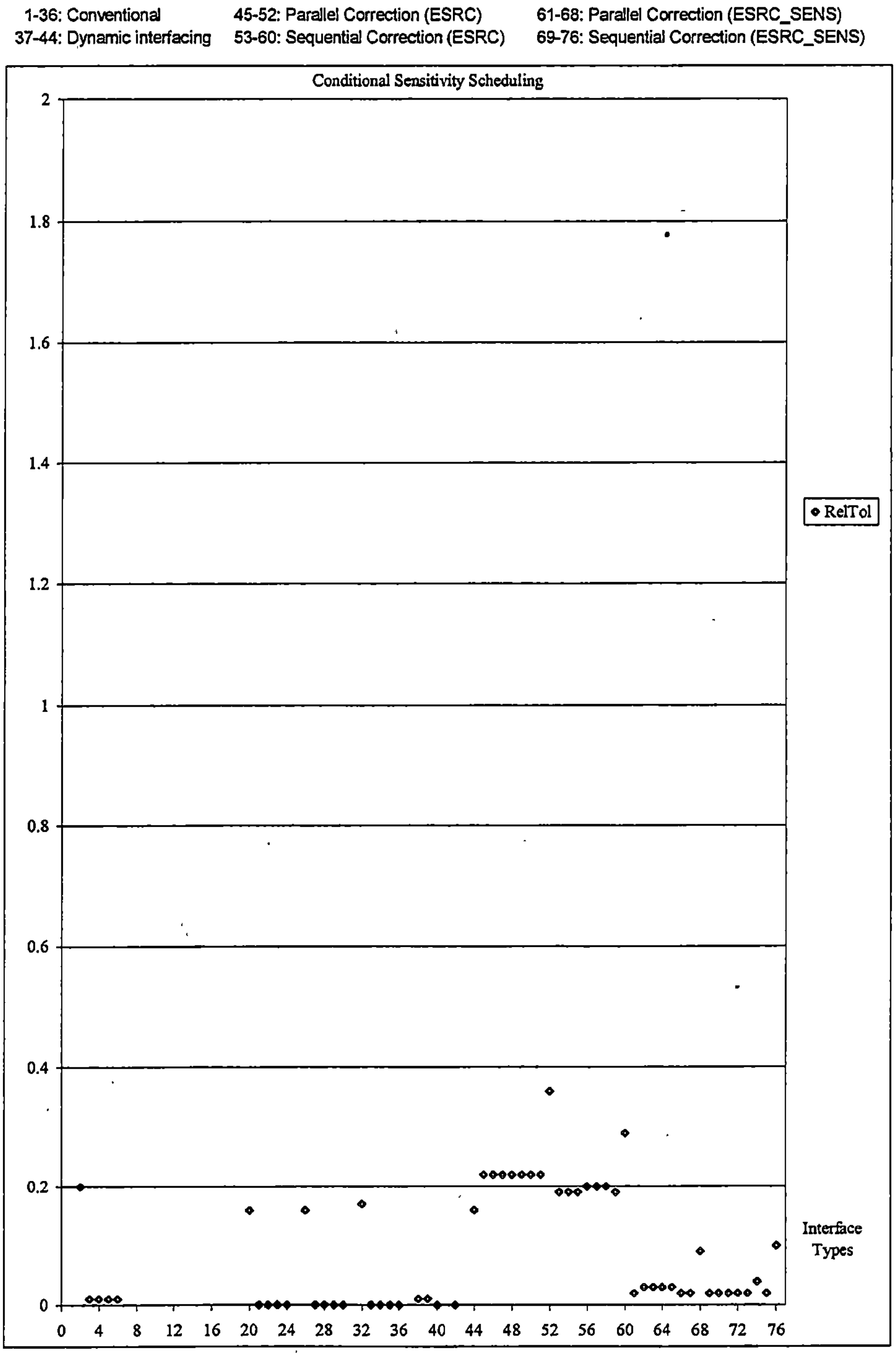


Figure D-54. Iteration results for E2FE_NF_DRVVIR4_PE

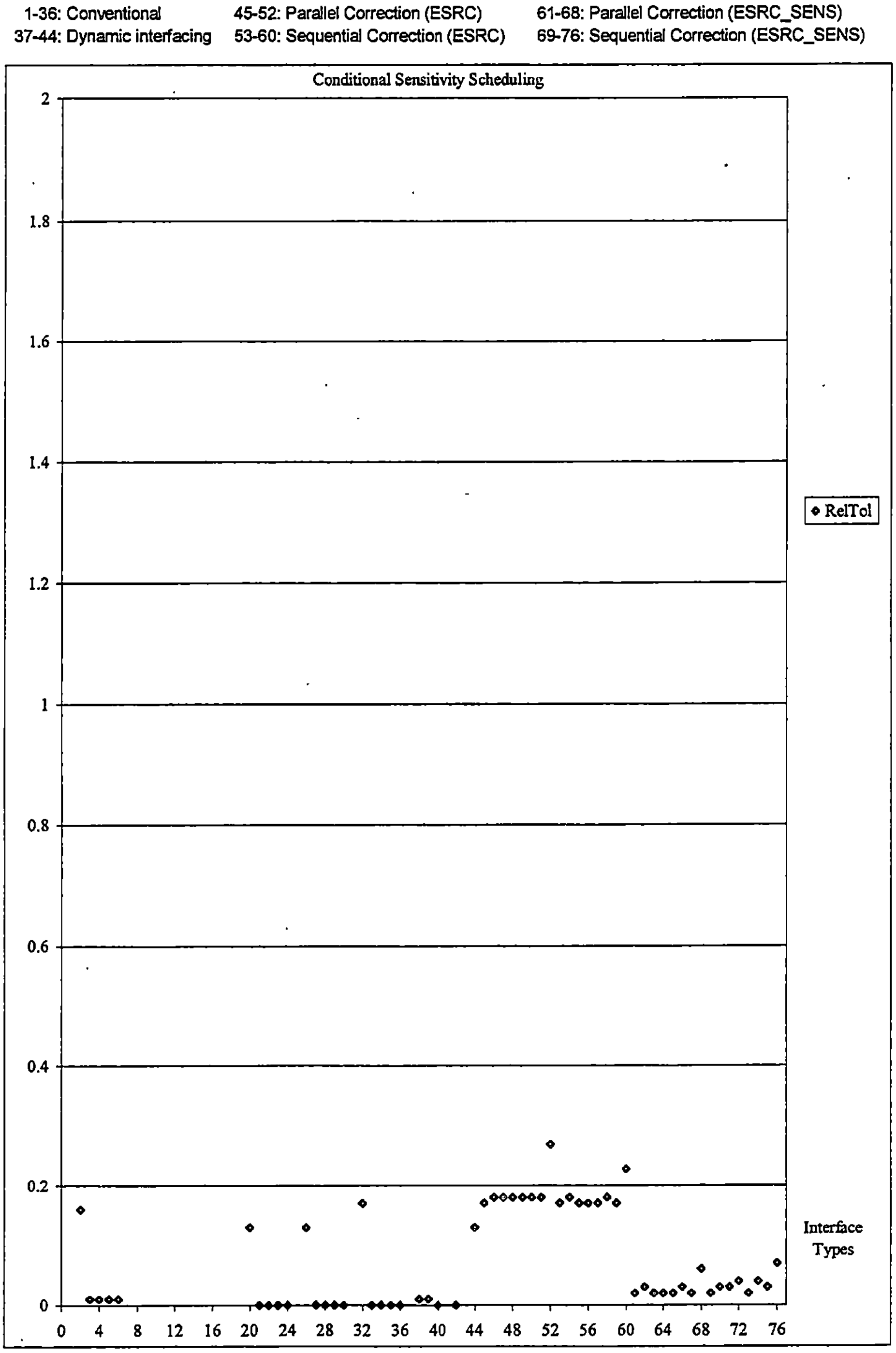


Figure D-55. Iteration results for E2FE_NF_DRVVIR4_PR1

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

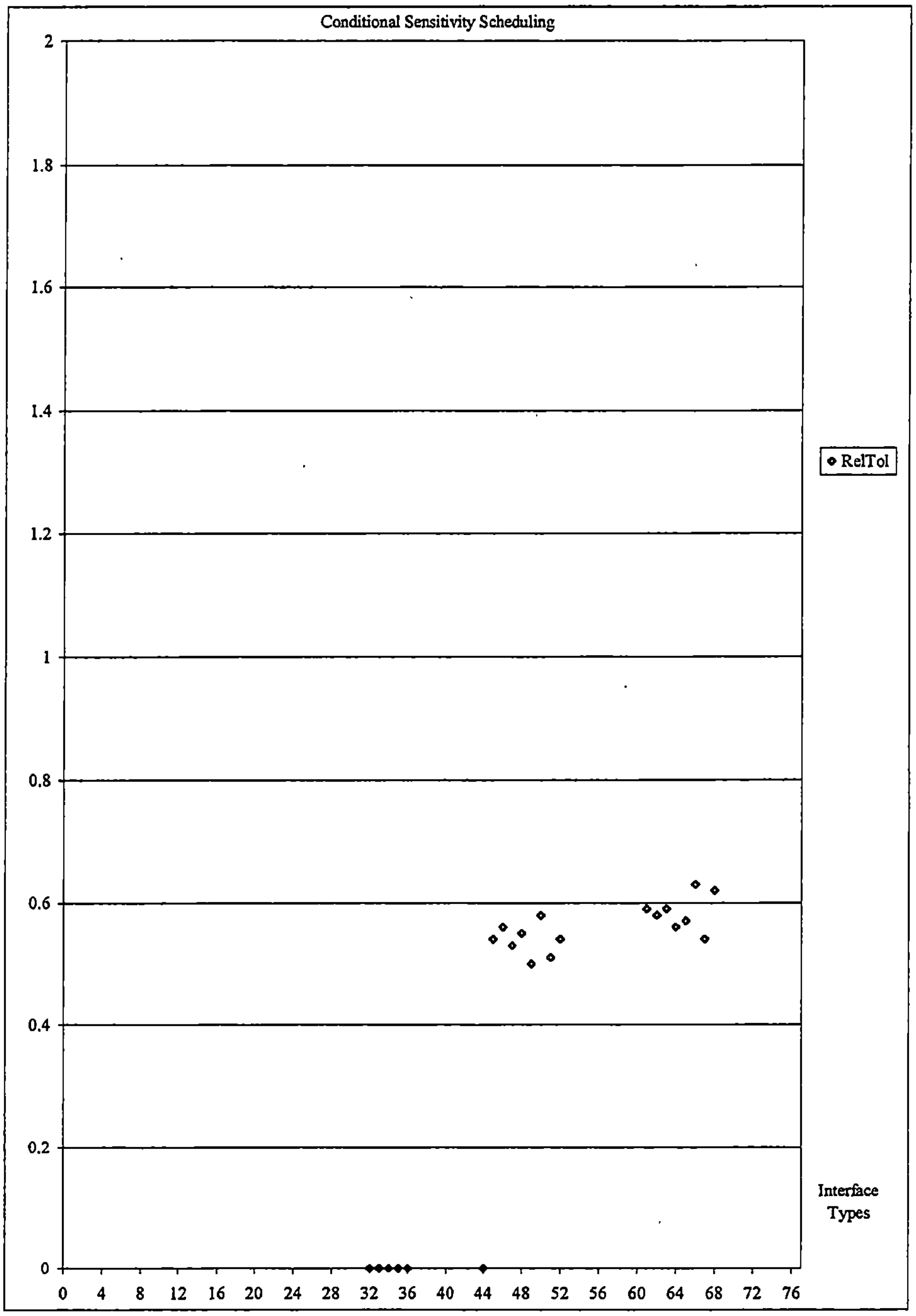


Figure D-56. Iteration results for E2FE_HF_AMPHB2_PE

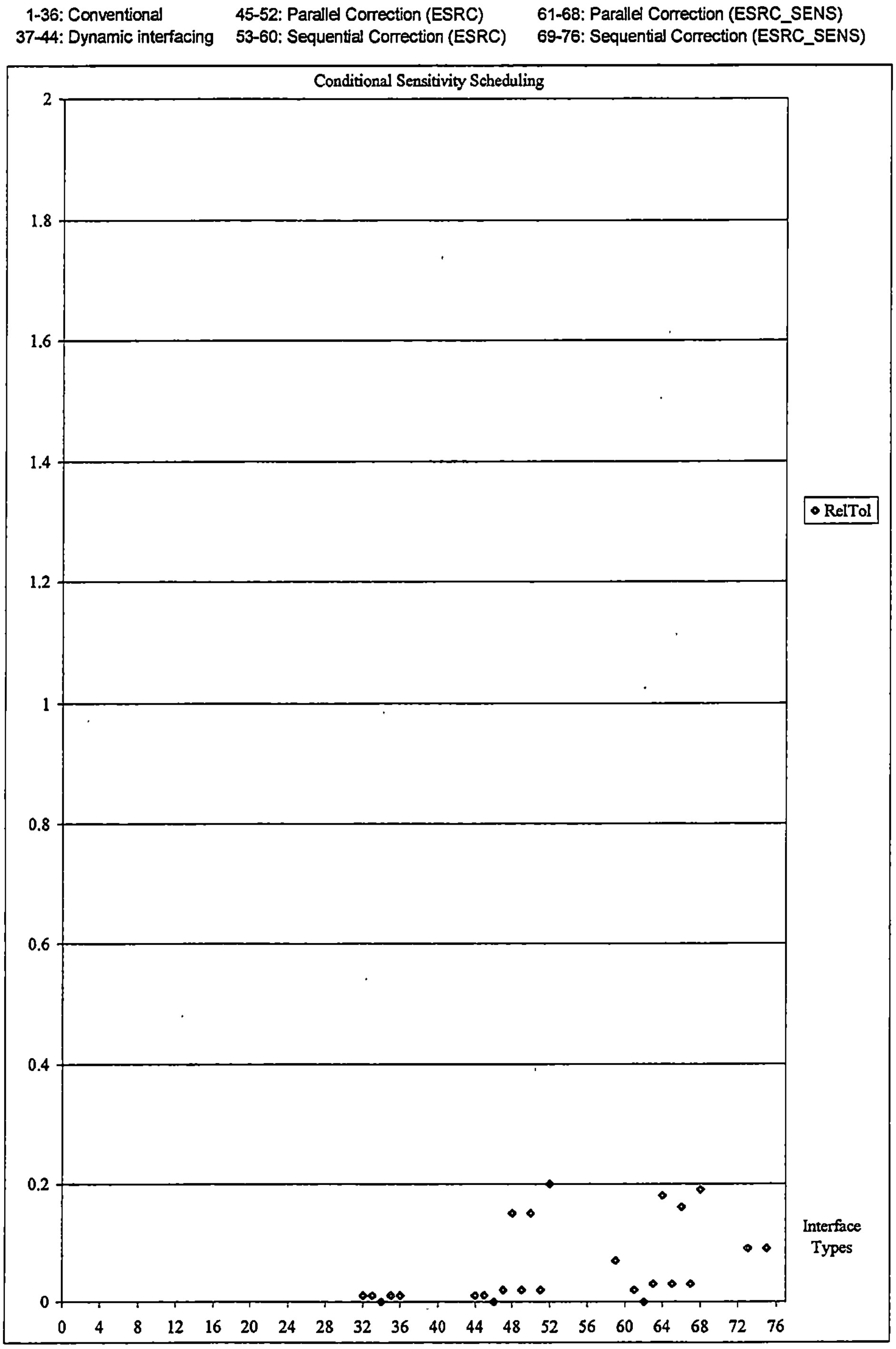


Figure D-57. Iteration results for E2FE_HF_AMPHB2_PR1

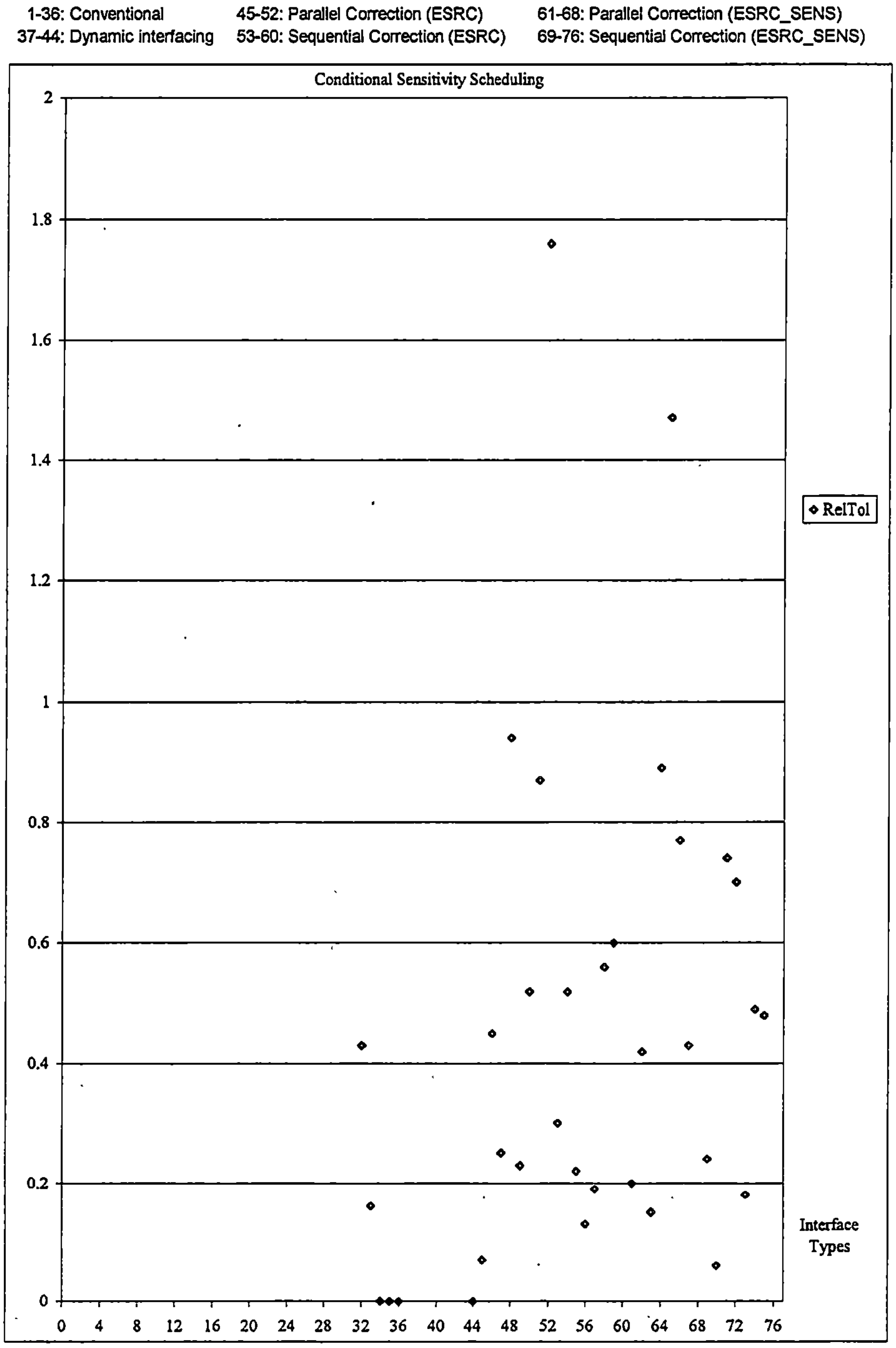


Figure D-58. Iteration results for E2FE_HF_AMPHB4_PE

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

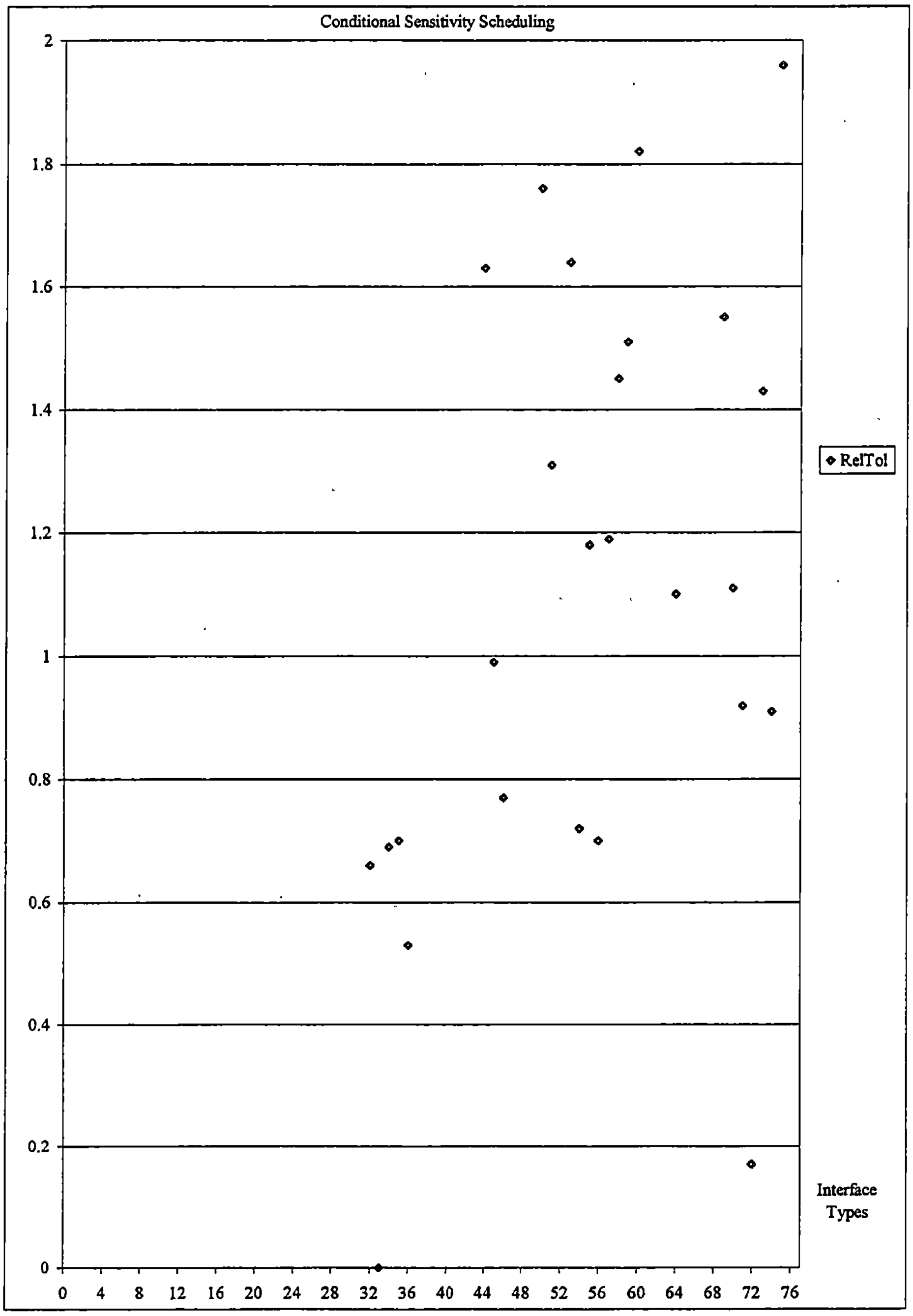


Figure D-59. Iteration results for E2FE_HF_AMPHB4_PR1

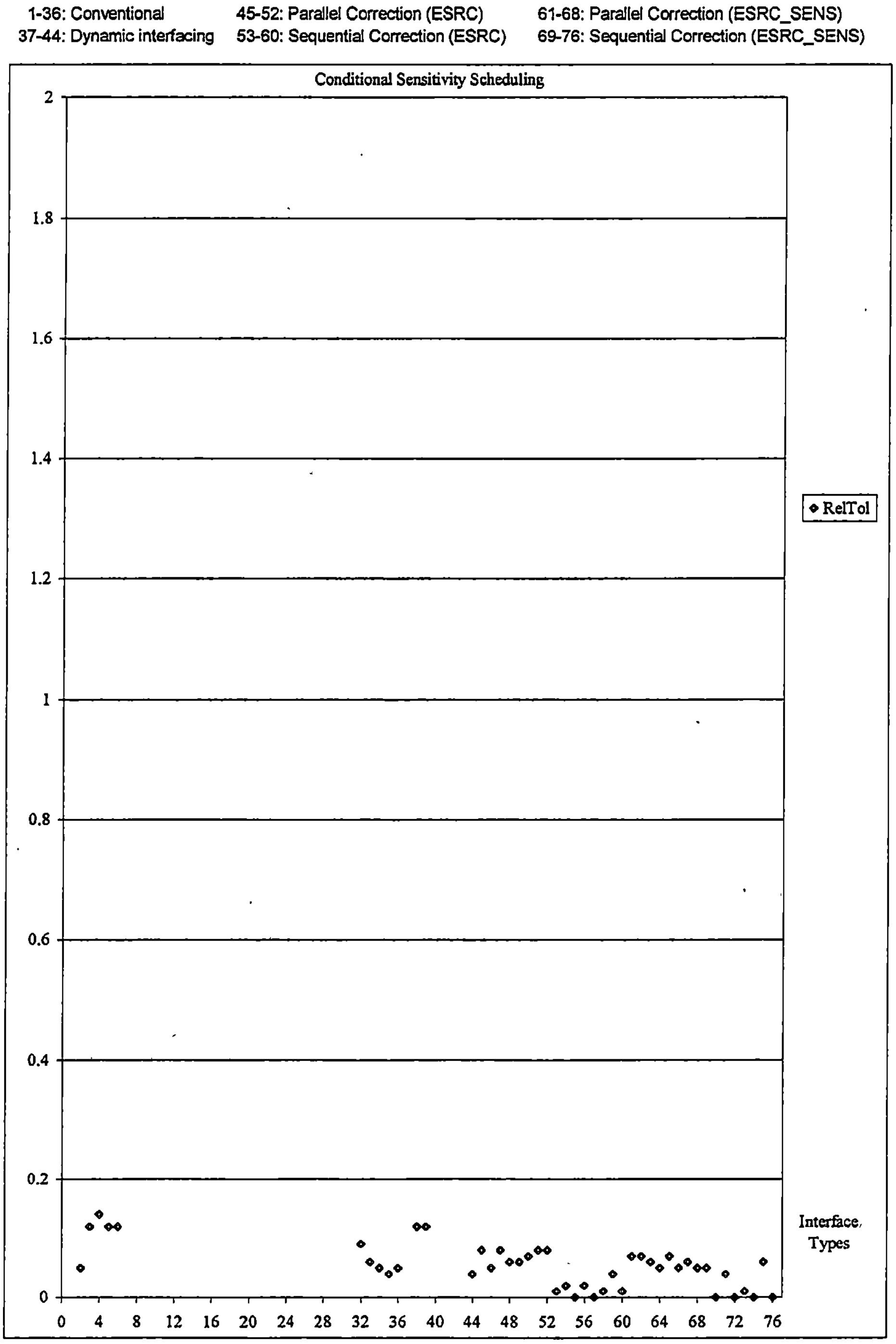


Figure D-60. Iteration results for E2FE_HF_DRVVIR2_PE

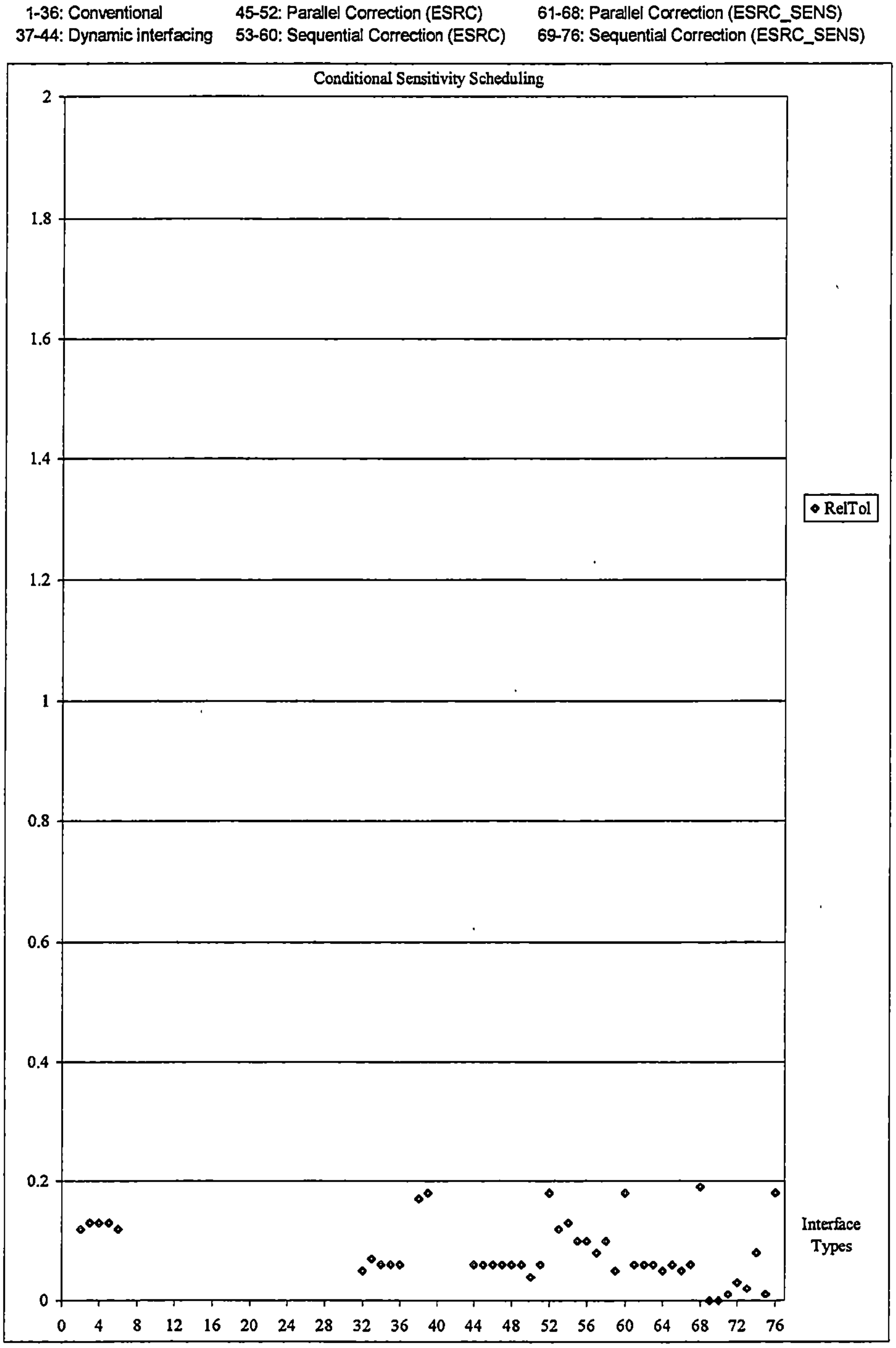


Figure D-61. Iteration results for E2FE_HF_DRVVIR2_PR1

1-36: Conventional 45-52: Parallel Correction (ESRC) 61-68: Parallel Correction (ESRC_SENS)
37-44: Dynamic interfacing 53-60: Sequential Correction (ESRC) 69-76: Sequential Correction (ESRC_SENS)

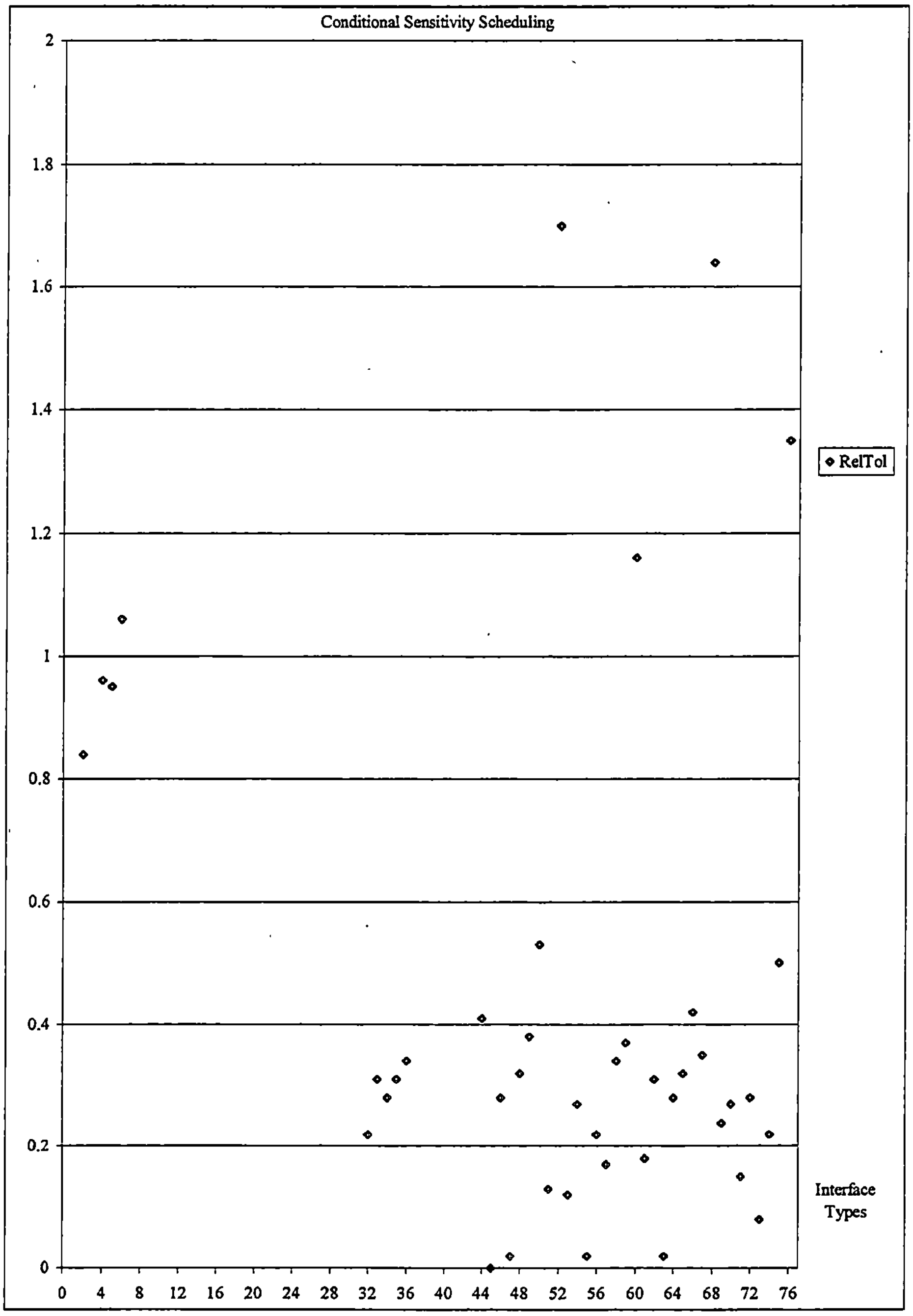


Figure D-62. Iteration results for E2FE_HF_DRVVIR4_PE

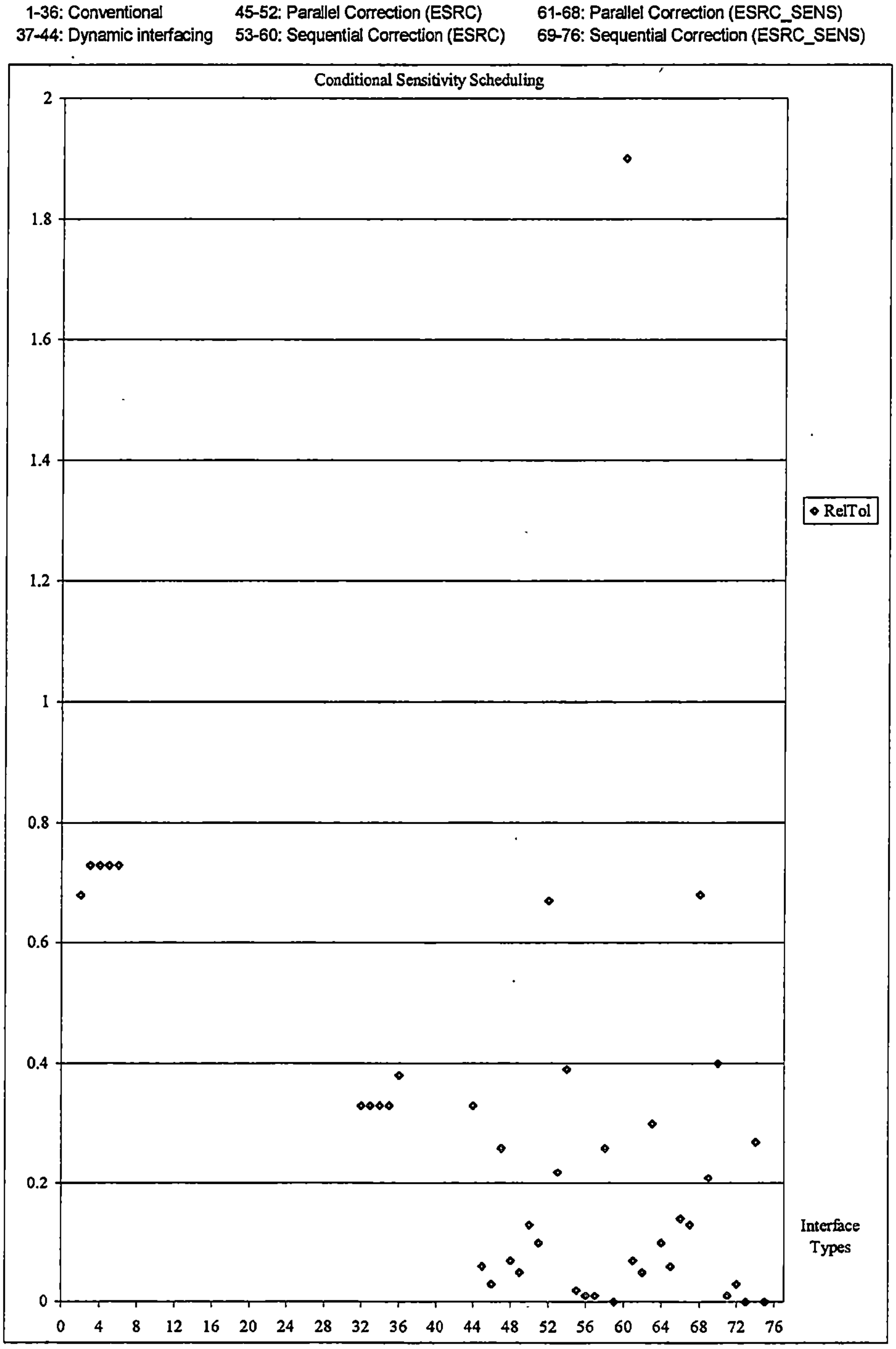


Figure D-63. Iteration results for E2FE_HF_DRVVIR4_PR1

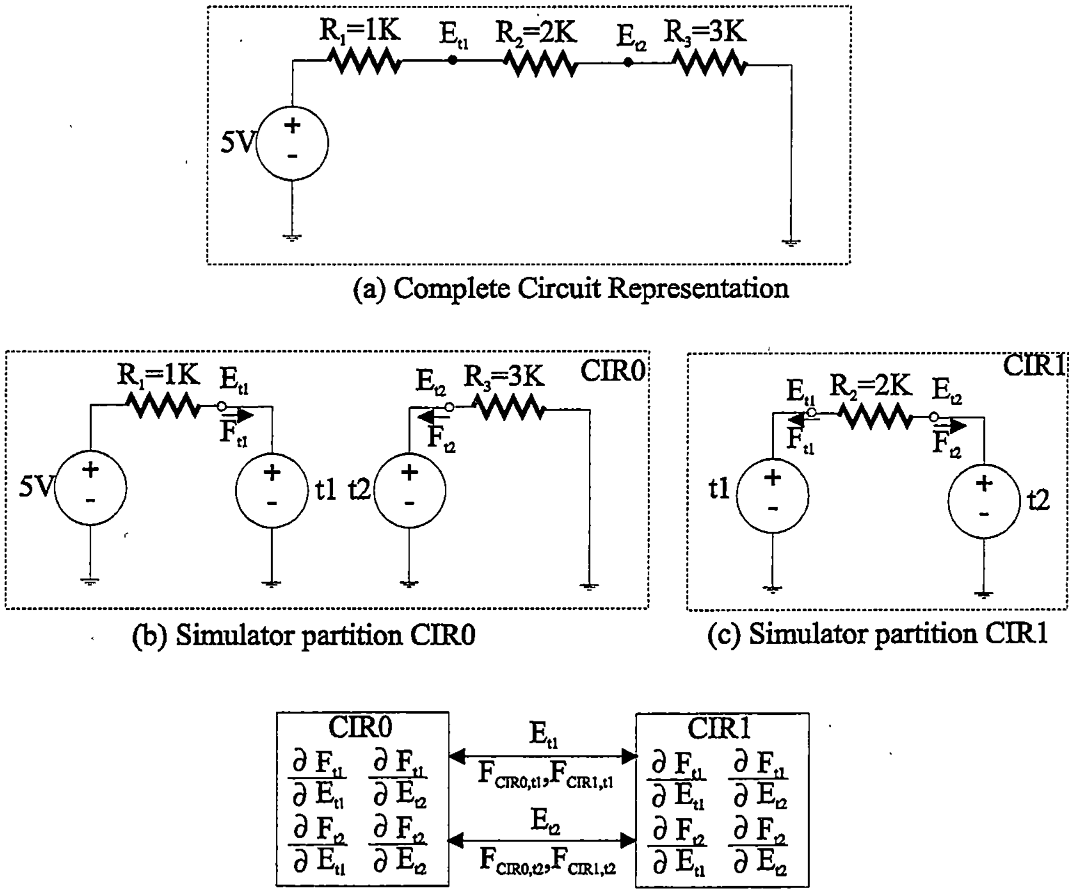
APPENDIX-E

Appendix-E. Backplane Calculation Tutorial

This section provides a detailed outline of the backplane calculation and causality detection processes to clarify the procedures from Chapter 3. The first example is a simple case, which essentially implemented nodal analysis using the backplane procedures. The second example shows how the backplane detects an EFFORT output variable and modifies the sensitivity calculations and backplane system matrix. Both examples show the initialization sequence (DC bias point calculations), which are the most critical steps in the backplane process. These examples showed that this process is a multilevel Newton procedure [12].

Appendix E.1 A Simple Example

A complete representation of this example is shown in Figure E-1 (a). The true solution of this problem was E_{T1}=4.17V and E_{T2}=2.5V with a current of 833 µA. To demonstrate the coupling procedure, this representation was broken into two simulation partitions in Figure E-1 (b) and (c). The interfaces into both simulators were defined as voltage sources. The objective was to calculate the same solution found in the complete representation. Because of the voltage sources, the coupling process only had to calculate conduction parameters ($\frac{\partial F_X}{\partial E_Y}$) as shown in Figure E-1 section (d).



(a) Complete Circuit Representation

(b) Simulator partition CIR0

(c) Simulator partition CIR1

(d) Coupling Representation

Figure E-1. Simple coupling example

To implement the coupling process, 14 different variables were required as shown in Table E-1. Each simulator contributed four variables (information to the backplane) and the backplane created six variables (indicated with the CALC subscript) to be used in the error minimization process. To create functions that were equal to zero for each simulator, the backplane created sensitivity functions. The sensitivity functions were used by the error minimization process in the coupling process and for the sensitivity parameter calculations in each simulator. Since the EFFORT variables were common between both simulators, only one unique EFFORT calculation variable was required in the process.

Table E-1. Simulator and backplane calculation variables

Simulator Return Variables	Backplane Calculation Variables
Partition CIR0 Variables: $E_{CIR0,T1}$, $F_{CIR0,T1}$ $E_{CIR0,T2}$, $F_{CIR0,T2}$	Equivalent CIR0 Variables: $E_{T1}\|_{CALC}$, $F_{CIR0,T1}\|_{CALC}$ $E_{T2}\|_{CALC}$, $F_{CIR0,T2}\|_{CALC}$
Partition CIR1 Variables: $E_{CIR1,T1}$, $F_{CIR1,T1}$ $E_{CIR1,T2}$, $F_{CIR1,T2}$	Equivalent CIR1 Variables: $E_{T1}\|_{CALC}$, $F_{CIR1,T1}\|_{CALC}$ $E_{T2}\|_{CALC}$, $F_{CIR1,T2}\|_{CALC}$

For partition CIR0, the sensitivity functions were:

$$S(F_{CIR0,T1}) = \left.\frac{\partial F_{T1}}{\partial E_{T1}}\right|_{CIR0} \cdot \Delta E_{CIR0,T1} + \left.\frac{\partial F_{T1}}{\partial E_{T2}}\right|_{CIR0} \cdot \Delta E_{CIR0,T2} + \Delta F_{CIR0,T1}$$
$$S(F_{CIR0,T2}) = \left.\frac{\partial F_{T2}}{\partial E_{T1}}\right|_{CIR0} \cdot \Delta E_{CIR0,T1} + \left.\frac{\partial F_{T2}}{\partial E_{T2}}\right|_{CIR0} \cdot \Delta E_{CIR0,T2} + \Delta F_{CIR0,T2}$$

Equation E-1

The corresponding error minimization equations for partition CIR0 were defined by:

$$\begin{aligned}
\Delta F_{CI0,T1} &= F_{CIR0,T1}\big|_{CALC} - F_{CIR0,T1} \\
\Delta F_{CIR0,T2} &= F_{CIR0,T2}\big|_{CALC} - F_{CIR0,T2} \\
\Delta E_{CIR0,T1} &= E_{T1}\big|_{CALC} - E_{CIR0,T1} \\
\Delta E_{CIR0,T2} &= E_{T2}\big|_{CALC} - E_{CIR0,T2}
\end{aligned}$$

Equation E-2

For partition CIR1, the sensitivity functions were:

$$S(F_{CIR1,T1}) = \left.\frac{\partial F_{T1}}{\partial E_{T1}}\right|_{CIR1} \cdot \Delta E_{CIR1,T1} + \left.\frac{\partial F_{T1}}{\partial E_{T2}}\right|_{CIR1} \cdot \Delta E_{CIR1,T2} + \Delta F_{CIR1,T1}$$
$$S(F_{CIR1,T2}) = \left.\frac{\partial F_{T2}}{\partial E_{T1}}\right|_{CIR1} \cdot \Delta E_{CIR1,T1} + \left.\frac{\partial F_{T2}}{\partial E_{T2}}\right|_{CIR1} \cdot \Delta E_{CIR1,T2} + \Delta F_{CIR1,T2}$$

Equation E-3

The corresponding error minimization equations for partition CIR1 were defined by:

$$
\begin{aligned}
\Delta F_{CIR1,T1} &= F_{CIR1,T1}\big|_{CALC} - F_{CIR1,T1} \\
\Delta F_{CIR1,T2} &= F_{CIR1,T2}\big|_{CALC} - F_{CIR1,T2} \\
\Delta E_{CIR1,T1} &= E_{T1}\big|_{CALC} - E_{CIR1,T1} \\
\Delta E_{CIR1,T2} &= E_{T2}\big|_{CALC} - E_{CIR1,T2}
\end{aligned}
\qquad \text{Equation E-4}
$$

The sensitivity functions of both simulators were put into matrix form with the interconnection relationships:

$$
\begin{bmatrix}
0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 \\
\left.\frac{\partial F_{T1}}{\partial E_{T1}}\right|_{CIR0} & \left.\frac{\partial F_{T1}}{\partial E_{T2}}\right|_{CIR0} & 1 & 0 & 0 & 0 \\
\left.\frac{\partial F_{T2}}{\partial E_{T1}}\right|_{CIR0} & \left.\frac{\partial F_{T2}}{\partial E_{T2}}\right|_{CIR0} & 0 & 1 & 0 & 0 \\
\left.\frac{\partial F_{T1}}{\partial E_{T1}}\right|_{CIR1} & \left.\frac{\partial F_{T1}}{\partial E_{T2}}\right|_{CIR1} & 0 & 0 & 1 & 0 \\
\left.\frac{\partial F_{T2}}{\partial E_{T1}}\right|_{CIR1} & \left.\frac{\partial F_{T2}}{\partial E_{T2}}\right|_{CIR1} & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\Delta E_{T1}\big|_{CALC} \\
\Delta E_{T2}\big|_{CALC} \\
\Delta F_{CIR0,T1}\big|_{CALC} \\
\Delta F_{CIR0,T2}\big|_{CALC} \\
\Delta F_{CIR1,T1}\big|_{CALC} \\
\Delta F_{CIR1,T2}\big|_{CALC}
\end{bmatrix}
=
\begin{bmatrix}
0 \\
0 \\
S(F_{CIR0,T1}) \\
S(F_{CIR0,T2}) \\
S(F_{CIR1,T1}) \\
S(F_{CIR1,T2})
\end{bmatrix}
\qquad \text{Equation E-5}
$$

The matrix in Equation E-5 was called the system matrix in this tutorial. The interconnection relationships were defined in the first two rows, and these relationships defined which FLOW variables (or FLOW delta variables) summed to zero. The sensitivity functions were used in the other four rows. After solving for the calculated delta values in Equation E-5, an iteration process was defined for the calculated variables:

$$
\begin{aligned}
F_{CIR0,T1}\big|_{CALC}^{i+1} &= F_{CIR0,T1}\big|_{CALC}^{i} + \Delta F_{CIR0,T1}\big|_{CALC} \\
F_{CIR0,T2}\big|_{CALC}^{i+1} &= F_{CIR0,T2}\big|_{CALC}^{i} + \Delta F_{CIR0,T2}\big|_{CALC} \\
F_{CIR1,T1}\big|_{CALC}^{i+1} &= F_{CIR1,T1}\big|_{CALC}^{i} + \Delta F_{CIR1,T1}\big|_{CALC} \\
F_{CIR1,T2}\big|_{CALC}^{i+1} &= F_{CIR1,T2}\big|_{CALC}^{i} + \Delta F_{CIR1,T2}\big|_{CALC} \\
E_{T1}\big|_{CALC}^{i+1} &= E_{,T1}\big|_{CALC}^{i} + \Delta E_{,T1}\big|_{CALC} \\
E_{T2}\big|_{CALC}^{i+1} &= E_{,T2}\big|_{CALC}^{i} + \Delta E_{,T2}\big|_{CALC}
\end{aligned}
\qquad \text{Equation E-6}
$$

To derive the sensitivity parameters, a perturbation matrix based on the sensitivity function was solved by the backplane component in a simulator to define functions that were zero. The process was defined by:

$$S(F_{T1}) = \frac{\partial F_{T1}}{\partial E_{T1}} \cdot \Delta E_{T1,X} + \frac{\partial F_{T1}}{\partial E_{T2}} \cdot \Delta E_{T2,X} + \Delta F_{T1,X} = 0$$
$$S(F_{T2}) = \frac{\partial F_{T2}}{\partial E_{T1}} \cdot \Delta E_{T1,X} + \frac{\partial F_{T2}}{\partial E_{T2}} \cdot \Delta E_{T2,X} + \Delta F_{T2,X} = 0$$

Equation E-7

The X variable defined the interface where the variable delta was applied. The delta information had to be applied twice (once at each interface) to calculate the four sensitivity parameters. In matrix form, the parameters were solved by using Equation E-8.

$$\begin{bmatrix} \Delta E_{T1,T1} & 0 & \Delta E_{T2,T1} & 0 \\ 0 & \Delta E_{T1,T1} & 0 & \Delta E_{T2,T1} \\ \Delta E_{T1,T2} & 0 & \Delta E_{T2,T2} & 0 \\ 0 & \Delta E_{T1,T2} & 0 & \Delta E_{T2,T2} \end{bmatrix} \begin{bmatrix} \partial F_{T1} / \partial E_{T1} \\ \partial F_{T2} / \partial E_{T1} \\ \partial F_{T1} / \partial E_{T2} \\ \partial F_{T2} / \partial E_{T2} \end{bmatrix} = \begin{bmatrix} \Delta F_{T1,T1} \\ \Delta F_{T2,T1} \\ \Delta F_{T1,T2} \\ \Delta F_{T2,T2} \end{bmatrix}$$

Equation E-8

These equations were the mathematical foundation for the coupling process. This entire procedure attempts to eliminate the error in Equation E-2 and E-4. When these equations were approximately equal to zero, the coupling process had essentially converged.

To demonstrate this coupling process, the simulator information was presented in iteration blocks with sensitivity iterations defined by the format #-S.OBJECT. After each block, the perturbation matrix (Equation E-8) was solved as necessary. The backplane process required seven iteration steps because of the initialization procedures. The initialization procedures are described in Appendix-B and mentioned in this text when appropriate. Once a four-step setup phase was completed, the backplane constructed and solved the system matrix.

On iteration 1, the simulator solution of partition CIR0 was:

Iteration	CIR0 Simulator Response			Source Values
	Object	Effort	Flow	
1	T1	0.000e0	5.000e-3	ESRC = 0.000e0
1	T2	0.000e0	0.000e0	ESRC = 0.000e0
1-S.T2	T1	+0.000e0	+0.000e0	-
1-S.T2	T2	-1.000e-4	+3.333e-8	-
1-S.T1	T1	-1.000e-4	+1.000e-7	-
1-S.T1	T2	+0.000e0	+0.000e0	-

Sensitivity calculations for CIR0:

$$\begin{bmatrix} -1.0e-4 & 0 & 0 & 0 \\ 0 & -1.0e-4 & 0 & 0 \\ 0 & 0 & -1.0e-4 & 0 \\ 0 & 0 & 0 & -1.0e-4 \end{bmatrix} \begin{bmatrix} \frac{\partial T1{:}F}{\partial T1{:}E} \\ \frac{\partial T1{:}F}{\partial T2{:}E} \\ \frac{\partial T2{:}F}{\partial T1{:}E} \\ \frac{\partial T2{:}F}{\partial T2{:}E} \end{bmatrix} = \begin{bmatrix} -1.000e-7 \\ 0 \\ 0 \\ -3.333e-8 \end{bmatrix}$$

$$\frac{\partial F_{T1}}{\partial E_{T1}} = 1.000e-3 \quad \frac{\partial F_{T2}}{\partial E_{T1}} = 0.0 \quad \frac{\partial F_{T1}}{\partial E_{T2}} = 0.0 \quad \frac{\partial F_{T2}}{\partial E_{T2}} = 3.333e-4$$

On iteration 1, the simulator solution for partition CIR1 was:

Iteration	CIR1 Simulator Response			Source Values
	Object	Effort	Flow	
1	T1	0.000e0	0.000e0	ESRC = 0.000e0
1	T2	0.000e0	0.000e0	ESRC = 0.000e0
1-S.T2	T1	+0.000e0	-5.000e-8	-
1-S.T2	T2	-1.000e-4	+5.000e-8	-
1-S.T1	T1	-1.000e-4	+5.000e-8	-
1-S.T1	T2	+0.000e0	-5.000e-8	-

Sensitivity calculations for CIR1:

$$\begin{bmatrix} -1.000e-4 & 0 & 0 & 0 \\ 0 & -1.000e-4 & 0 & 0 \\ 0 & 0 & -1.000e-4 & 0 \\ 0 & 0 & 0 & -1.000e-4 \end{bmatrix} \begin{bmatrix} \frac{\partial T1:F}{\partial T1:E} \\ \frac{\partial T1:F}{\partial T2:E} \\ \frac{\partial T2:F}{\partial T1:E} \\ \frac{\partial T2:F}{\partial T2:E} \end{bmatrix} = \begin{bmatrix} -5.000e-8 \\ +5.0e-8 \\ +5.0e-8 \\ -5.0e-8 \end{bmatrix}$$

$$\frac{\partial F_{T1}}{\partial E_{T1}} = 5.0e-4 \quad \frac{\partial F_{T2}}{\partial E_{T1}} = -5.0e-4 \quad \frac{\partial F_{T1}}{\partial E_{T2}} = -5.0e-4 \quad \frac{\partial F_{T2}}{\partial E_{T2}} = 5.0e-4$$

The simulator responses for iterations 1 to 3 were identical, so these responses are not shown. On the fourth iteration, the backplane constructed and solved the interconnect matrix.

$$F_{CIR1,T1}\Big|_{CALC}^{3} = F_{CIR0,T1}\Big|_{CALC}^{3} = 0.0$$
$$F_{CIR1,T2}\Big|_{CALC}^{3} = F_{CIR0,T2}\Big|_{CALC}^{3} = 0.0$$
$$E_{T1}\Big|_{CALC}^{3} = E_{T2}\Big|_{CALC}^{3} = 0.0$$

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1.0e-3 & 0 & 1 & 0 & 0 & 0 \\ 0 & 3.33e-4 & 0 & 1 & 0 & 0 \\ 5.0e-4 & -5.0e-4 & 0 & 0 & 1 & 0 \\ -5.0e-4 & 5.0e-4 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta E_{T1}\big|_{CALC} \\ \Delta E_{T2}\big|_{CALC} \\ \Delta F_{CIR0,T1}\big|_{CALC} \\ \Delta F_{CIR0,T2}\big|_{CALC} \\ \Delta F_{CIR1,T1}\big|_{CALC} \\ \Delta F_{CIR1,T2}\big|_{CALC} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 5.0e-3 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\Delta F_{CIR0,T1}\Big|_{CALC} = 8.33e-4 \quad \Delta F_{CIR0,T2}\Big|_{CALC} = -8.33e-4 \quad \Delta F_{CIR1,T1}\Big|_{CALC} = -8.33e-4$$
$$\Delta F_{CIR1,T2}\Big|_{CALC} = 8.33e-4 \quad \Delta E_{,T1}\Big|_{CALC} = 4.167 \quad \Delta E_{,T2}\Big|_{CALC} = 2.5$$

$$F_{CIR0,T1}\Big|_{CALC}^{4} = F_{CIR0,T1}\Big|_{CALC}^{3} + \Delta F_{CIR0,T1} = 8.33e-4$$
$$F_{CIR0,T2}\Big|_{CALC}^{4} = F_{CIR0,T2}\Big|_{CALC}^{3} + \Delta F_{CIR0,T2} = -8.33e-4$$
$$F_{CIR1,T1}\Big|_{CALC}^{4} = F_{CIR1,T1}\Big|_{CALC}^{3} + \Delta F_{CIR1,T1} = -8.33e-4$$
$$F_{CIR1,T2}\Big|_{CALC}^{4} = F_{CIR1,T2}\Big|_{CALC}^{3} + \Delta F_{CIR1,T2} = 8.33e-4$$
$$E_{T1}\Big|_{CALC}^{4} = E_{,T1}\Big|_{CALC}^{3} + \Delta E_{,T1} = 4.167$$
$$E_{T2}\Big|_{CALC}^{4} = E_{,T2}\Big|_{CALC}^{3} + \Delta E_{,T2} = 2.5$$

On iteration 4, the backplane applied the solution back to the simulators.

Iteration	CIR0 Simulator Response			Source Values
	Object	Effort	Flow	
4	T1	4.167e0	8.333e0	ESRC = 4.167e0
4	T2	0.000e0	-8.333e0	ESRC = 2.500e0
4-S.T2	T1	+0.000e0	-5.000e-8	-
4-S.T2	T2	-1.000e-4	+5.000e-8	-
4-S.T1	T1	-1.000e-4	+5.000e-8	-
4-S.T1	T2	+0.000e0	-5.000e-8	-

Iteration	CIR1 Simulator Response			Source Values
	Object	Effort	Flow	
4	T1	4.167e0	-8.333e0	ESRC = 4.167e0
4	T2	0.000e0	8.333e0	ESRC = 2.500e0
4-S.T2	T1	+0.000e0	-5.000e-8	-
4-S.T2	T2	-1.000e-4	+5.000e-8	-
4-S.T1	T1	-1.000e-4	+5.000e-8	-
4-S.T1	T2	+0.000e0	-5.000e-8	-

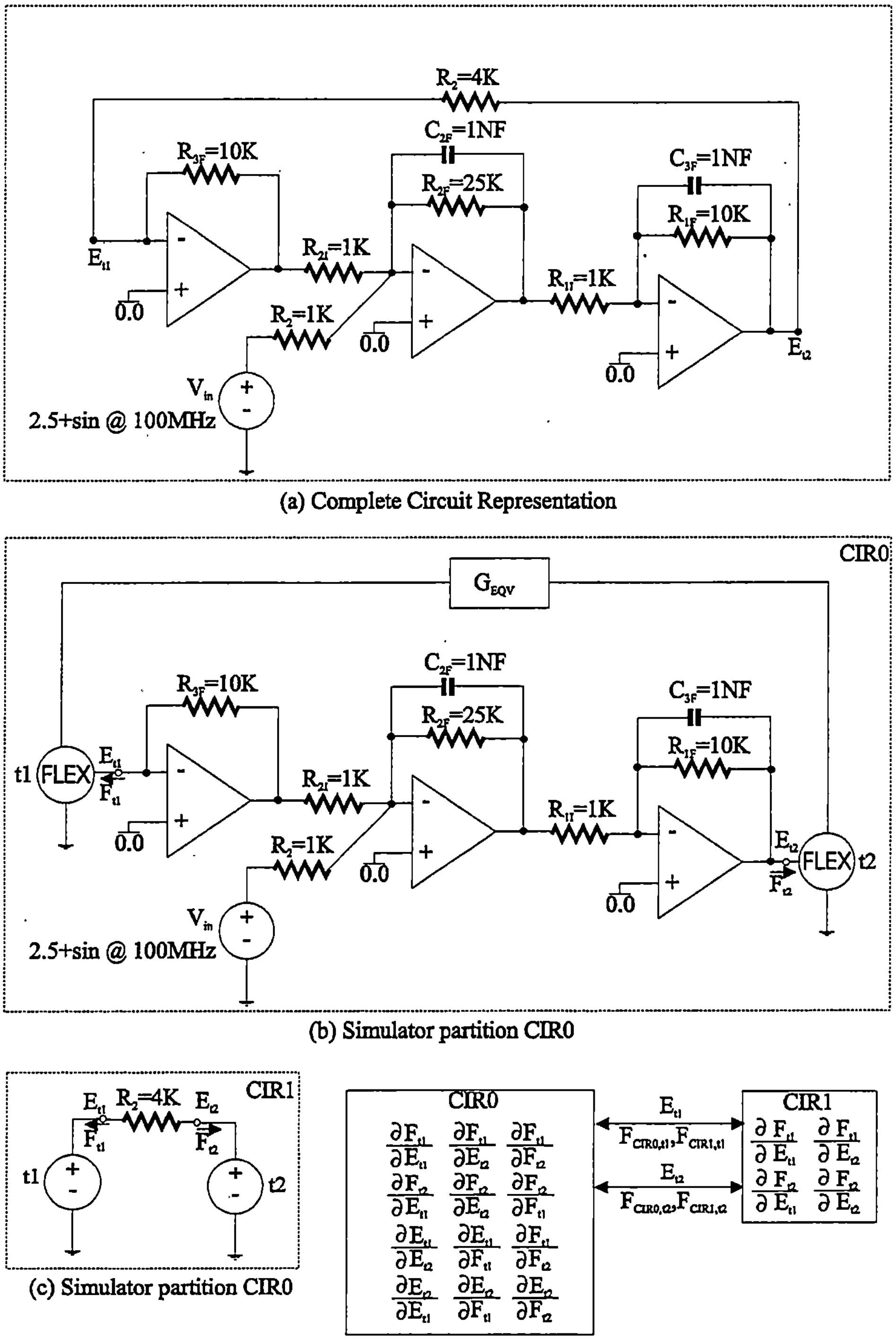
The next backplane sequence had no error between the simulator and calculated values.

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1.0e-3 & 0 & 1 & 0 & 0 & 0 \\ 0 & 3.33e-4 & 0 & 1 & 0 & 0 \\ 5.0e-4 & -5.0e-4 & 0 & 0 & 1 & 0 \\ -5.0e-4 & 5.0e-4 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta E_{T1}\big|_{CALC} \\ \Delta E_{T2}\big|_{CALC} \\ \Delta F_{CIR0,T1}\big|_{CALC} \\ \Delta F_{CIR0,T2}\big|_{CALC} \\ \Delta F_{CIR1,T1}\big|_{CALC} \\ \Delta F_{CIR1,T2}\big|_{CALC} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Once all convergence checks and initialization steps were completed, the backplane started the transient analysis. The transient analysis followed the same procedures of eliminating the error between the calculated variables and the simulator variables.

Appendix E.2 EFFORT Output Example

High gain problems like the example in Figure E-2 (a) were more difficult to solve than the previous example, especially depending on the interface used in a partition. The complete representation was divided into two partitions with two interface terminals like the previous example, so the same coupling variables were created as defined in Table E-1. The CIR1 partition in Figure E-2 (c) was interfaced to the coupling process using voltage sources like the previous example. However, the CIR0 partition in Figure E-2 (b) was interfaced to the backplane using the SYSTEM configuration of a FLEXIBLE interface. Consequently, the CIR0 partition had the capability to calculate additional sensitivity parameters as shown in Figure E-2 (d).


R_2=4K
C_{2F}=1NF
R_{3F}=10K
R_{2F}=25K
C_{3F}=1NF
R_{1F}=10K
E_{t1}
R_{2I}=1K
R_2=1K
R_{1I}=1K
0.0
0.0
0.0
E_{t2}
V_{in}
2.5+sin @ 100MHz
CIR0
G_{EQV}
t1
FLEX
F_{t1}
t2
F_{t2}
CIR1
$\frac{\partial F_{t1}}{\partial E_{t1}}$ $\frac{\partial F_{t1}}{\partial E_{t2}}$ $\frac{\partial F_{t1}}{\partial F_{t2}}$
$\frac{\partial F_{t2}}{\partial E_{t1}}$ $\frac{\partial F_{t2}}{\partial E_{t2}}$ $\frac{\partial F_{t2}}{\partial F_{t1}}$
$\frac{\partial E_{t1}}{\partial E_{t2}}$ $\frac{\partial E_{t1}}{\partial F_{t1}}$ $\frac{\partial F_{t1}}{\partial F_{t2}}$
$\frac{\partial E_{t2}}{\partial E_{t1}}$ $\frac{\partial E_{t2}}{\partial F_{t1}}$ $\frac{\partial E_{t2}}{\partial F_{t2}}$
E_{t1}
$F_{CIR0,t1}$,$F_{CIR1,t1}$
E_{t2}
$F_{CIR0,t2}$,$F_{CIR1,t2}$
$\frac{\partial F_{t1}}{\partial E_{t1}}$ $\frac{\partial F_{t1}}{\partial E_{t2}}$
$\frac{\partial F_{t2}}{\partial E_{t1}}$ $\frac{\partial F_{t2}}{\partial E_{t2}}$

(a) Complete Circuit Representation

(b) Simulator partition CIR0

(c) Simulator partition CIR0

(d) Coupling Representation

Figure E-2. High gain feedback problem

In the high gain situation, the main issue was to get a good initial solution (guess) that met the sufficiently close conditions required by most iteration routines. However, this partitioning process broke the feedback path, so the CIR0 partition had a very poor initialization solution due to the internal stimulus V_{IN}. Without the conduction matrix (G_{EQV}) generated by the SYSTEM configuration in partition CIR0, the backplane was unable to solve the problem. Using the conduction matrix, the backplane essentially created the feedback resistance of the CIR1 partition in partition CIR0. The problem solution was almost trivial since partition CIR1 only contributed a conduction relationship to the process. This example had an EFFORT output at terminal T2, which the backplane had to detect.

There were two reasons for detecting the EFFORT causality at terminal T2 and calculating a different set of sensitivity parameters. By detecting the gain, the backplane calculated parameters that corresponded to the true nature of the models in a simulator. In addition, the backplane used this information in certain interface procedures. The second and most important reason was due to matrix constraints. When applying delta information to a specific interface, the delta of the given interface should correspond to one of the largest delta responses. Without this condition, the matrix became ill conditioned, which caused failures in the pivoting LU decomposition solver routines in the backplane.

At initialization, this example used the same mathematical foundation as the previous example. When the EFFORT output variable was detected at interface T2 in the CIR0 partition, the backplane calculated a different set of sensitivity parameters for the CIR0 partition, and the system matrix changed. The new sensitivity functions solved in the CIR0 partition were defined in Equation E-9.

$$S(F_{T1}) = \frac{\partial F_{T1}}{\partial E_{T1}} \cdot \Delta E_{T1,X} + \frac{\partial F_{T1}}{\partial E_{T2}} \cdot \Delta E_{T2,X} + \Delta F_{T1,X} = 0$$

$$S(E_{T2}) = \frac{\partial E_{T2}}{\partial E_{T1}} \cdot \Delta E_{T1,X} + \frac{\partial E_{T2}}{\partial F_{T2}} \cdot \Delta F_{T2,X} + \Delta E_{T2,X} = 0$$

Equation E-9

In matrix form, the sensitivity parameters were solved using Equation E-10.

$$\begin{bmatrix} \Delta E_{T1,T1} & 0 & \Delta F_{T2,T1} & 0 \\ 0 & \Delta E_{T1,T1} & 0 & \Delta F_{T2,T1} \\ \Delta E_{T1,T2} & 0 & \Delta F_{T2,T2} & 0 \\ 0 & \Delta E_{T1,T2} & 0 & \Delta F_{T2,T2} \end{bmatrix} \begin{bmatrix} \partial F_{T1} / \partial E_{T1} \\ \partial E_{T2} / \partial E_{T1} \\ \partial F_{T1} / \partial F_{T2} \\ \partial E_{T2} / \partial F_{T2} \end{bmatrix} = \begin{bmatrix} \Delta F_{T1,T1} \\ \Delta E_{T2,T1} \\ \Delta F_{T1,T2} \\ \Delta E_{T2,T2} \end{bmatrix}$$

Equation E-10

The variation in the sensitivity parameters changed the sensitivity functions used by the system matrix as shown in Equation E-11.

$$S(F_{CIR0,T1}) = \left.\frac{\partial F_{T1}}{\partial E_{T1}}\right|_{CIR0} \cdot \Delta E_{CIR0,T1} + \left.\frac{\partial F_{T1}}{\partial F_{T2}}\right|_{CIR0} \cdot \Delta F_{CIR0,T2} + \Delta F_{CIR0,T1}$$

$$S(E_{CIR0,T2}) = \left.\frac{\partial E_{T2}}{\partial E_{T1}}\right|_{CIR0} \cdot \Delta E_{CIR0,T1} + \left.\frac{\partial E_{T2}}{\partial F_{T2}}\right|_{CIR0} \cdot \Delta F_{CIR0,T2} + \Delta E_{CIR0,T2}$$

Equation E-11

The new system matrix was defined in Equation E-12. The interconnection relationship was defined in row 1 and row 4. These rows corresponded to the input variables in the sensitivity parameters.

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ \left.\frac{\partial F_{T1}}{\partial E_{T1}}\right|_{CIR0} & 0 & 1 & \left.\frac{\partial F_{T1}}{\partial F_{T2}}\right|_{CIR0} & 0 & 0 \\ \left.\frac{\partial E_{T2}}{\partial E_{T1}}\right|_{CIR0} & 1 & \left.\frac{\partial E_{T2}}{\partial F_{T1}}\right|_{CIR0} & 0 & 0 & 0 \\ \left.\frac{\partial F_{T1}}{\partial E_{T1}}\right|_{CIR1} & \left.\frac{\partial F_{T1}}{\partial E_{T2}}\right|_{CIR1} & 0 & 0 & 1 & 0 \\ \left.\frac{\partial F_{T2}}{\partial E_{T1}}\right|_{CIR1} & \left.\frac{\partial F_{T2}}{\partial E_{T2}}\right|_{CIR1} & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \left.\Delta E_{T1}\right|_{CALC} \\ \left.\Delta E_{T2}\right|_{CALC} \\ \left.\Delta F_{CIR0,T1}\right|_{CALC} \\ \left.\Delta F_{CIR0,T2}\right|_{CALC} \\ \left.\Delta F_{CIR1,T1}\right|_{CALC} \\ \left.\Delta F_{CIR1,T2}\right|_{CALC} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ S(F_{CIR0,T1}) \\ S(E_{CIR0,T2}) \\ S(F_{CIR1,T1}) \\ S(F_{CIR1,T2}) \end{bmatrix}$$

Equation E-12

The error equations (Equation E-2 and E-4), the CIR1 relationships (Equation E-3, E-7, and E-8), and iteration equation (Equation E-6) were identical to the previous example.

The iteration sequence for this example was more complex than the previous example because of the initialization procedures for the SYSTEM configuration in partition CIR0. At initialization, the equivalent matrix G_{EQV} in CIR0 was set to a minimum matrix where

$$G_{EQV} = \begin{bmatrix} 1e-6 & -1e-6 \\ -1e-6 & 1e-6 \end{bmatrix}$$

and

$$G_{EQV} = \begin{bmatrix} G_{1,1} & G_{1,2} \\ G_{2,1} & G_{2,2} \end{bmatrix}$$

$$G_{EQV} \begin{bmatrix} E_{T1} \\ E_{T2} \end{bmatrix} = \begin{bmatrix} F_{T1} \\ F_{T2} \end{bmatrix}$$

On iteration 1, the simulator solution for partition CIR0 was:

Iteration	CIR0 Simulator Response			Source Values
	Object	Effort	Flow	
1	T1	3.757e-8	-1.605e-5	FSRC = 0.000e0
1	T2	1.605e1	1.605e-5	FSRC = 0.000e0
1-S.T2	T1	+1.171e-8	-5.695e-7	-
1-S.T2	T2	-4.094e-4	+8.241e-6	-
1-S.T1	T1	-1.577e-4	+7.670e-6	-
1-S.T1	T2	-8.423e-6	+1.526e-9	-

Sensitivity calculations for CIR0:

$$\begin{bmatrix} -1.58e-7 & -8.42e-6 & 0 & 0 \\ 1.17e-8 & -4.09e-4 & 0 & 0 \\ 0 & 0 & -1.58e-4 & -8.42e-6 \\ 0 & 0 & 1.17e-8 & -4.09e-4 \end{bmatrix} \begin{bmatrix} \frac{\partial T1:F}{\partial T1:E} \\ \frac{\partial T1:F}{\partial T2:E} \\ \frac{\partial T2:F}{\partial T1:E} \\ \frac{\partial T2:F}{\partial T2:E} \end{bmatrix} = \begin{bmatrix} -7.67e-6 \\ 5.69e-7 \\ -1.53e-9 \\ -8.24e-6 \end{bmatrix}$$

$$\frac{\partial F_{T1}}{\partial E_{T1}} = 4.86e1 \quad \frac{\partial F_{T2}}{\partial E_{T1}} = -1.06e0 \quad \frac{\partial F_{T1}}{\partial E_{T2}} = -8.77e-10 \quad \frac{\partial F_{T2}}{\partial E_{T2}} = 2.04e-2$$

$$\frac{\partial F_{T1}}{\partial F_{T2}} = \frac{\partial F_{T2}}{\partial F_{T2}} = \frac{\partial E_{T1}}{\partial F_{T2}} = \frac{\partial E_{T2}}{\partial F_{T2}} = \frac{\partial E_{T1}}{\partial E_{T2}} = \frac{\partial E_{T2}}{\partial E_{T1}} = 0.0$$

On iteration 1, the simulator solution for partition CIR1 was:

Iteration	CIR1 Simulator Response			Source Values
	Object	Effort	Flow	
1	T1	0.000e0	0.000e0	ESRC = 0.000e0
1	T2	0.000e0	0.000e0	ESRC = 0.000e0
1-S.T2	T1	0.000e0	-2.500e-8	-
1-S.T2	T2	+1.000e-4	+2.500e-8	-
1-S.T1	T1	+1.000e-4	-2.500e-8	-
1-S.T1	T2	0.000e0	+2.500e-8	-

Sensitivity calculations for CIR1:

$$\begin{bmatrix} 1.0e-4 & 0 & 0 & 0 \\ 0 & 1.0e-4 & 0 & 0 \\ 0 & 0 & 1.0e-4 & 0 \\ 0 & 0 & 0 & 1.0e-4 \end{bmatrix} \begin{bmatrix} \frac{\partial T1:F}{\partial T1:E} \\ \frac{\partial T1:F}{\partial T2:E} \\ \frac{\partial T2:F}{\partial T1:E} \\ \frac{\partial T2:F}{\partial T2:E} \end{bmatrix} = \begin{bmatrix} 2.5e-8 \\ -2.5e-8 \\ -2.5e-8 \\ 2.5e-8 \end{bmatrix}$$

$$\frac{\partial F_{T1}}{\partial E_{T1}} = 2.5e-4 \quad \frac{\partial F_{T2}}{\partial E_{T1}} = -2.5e-4 \quad \frac{\partial F_{T1}}{\partial E_{T2}} = -2.5e-4 \quad \frac{\partial F_{T2}}{\partial E_{T2}} = 2.5e-4$$

The sensitivity parameters for CIR1 partition were constant through this analysis, so these calculations are not shown again.

On the second iteration, the backplane had to generate an equivalent sensitivity matrix (G_{EQV}) because CIR0 partition has a SYSTEM-level configuration. This calculation used the system matrix with the CIR0 components eliminated and with other matrix modifications for the EFFORT variables as shown in Equation E-13.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ \left.\frac{\partial F_{T1}}{\partial E_{T1}}\right|_{CIR1} & \left.\frac{\partial F_{T1}}{\partial E_{T2}}\right|_{CIR1} & 0 & 0 & 1 & 0 \\ \left.\frac{\partial F_{T2}}{\partial E_{T1}}\right|_{CIR1} & \left.\frac{\partial F_{T2}}{\partial E_{T2}}\right|_{CIR1} & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \left.\Delta E_{T1}\right|_{CALC} \\ \left.\Delta E_{T2}\right|_{CALC} \\ \left.\Delta F_{CIR0,T1}\right|_{CALC} \\ \left.\Delta F_{CIR0,T2}\right|_{CALC} \\ \left.\Delta F_{CIR1,T1}\right|_{CALC} \\ \left.\Delta F_{CIR1,T2}\right|_{CALC} \end{bmatrix} = [x] \qquad \text{Equation E-13}$$

The x vector was changed depending upon which set of equivalent parameters was calculated, and two calculations were required to generate all parameters. The first calculation generated the equivalent information for the E_{T1} variable.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 2.5e-4 & -2.5e-4 & 0 & 0 & 1 & 0 \\ -2.5e-4 & 2.5e-4 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta E_{T1} \\ \Delta E_{T2} \\ \Delta F_{CIR0,T1} \\ \Delta F_{CIR0,T2} \\ \Delta F_{CIR1,T1} \\ \Delta F_{CIR1,T2} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\left.\frac{\partial F_{T1}}{\partial E_{T1}}\right|_{EQV} = \frac{\left.\Delta F_{CIR0,T1}\right|_{CALC}}{\left.\Delta E_{T1}\right|_{CALC}} = 2.5e-4 \quad \left.\frac{\partial F_{T2}}{\partial E_{T1}}\right|_{EQV} = \frac{\left.\Delta F_{CIR0,T2}\right|_{CALC}}{\left.\Delta E_{T1}\right|_{CALC}} = -2.5e-4$$

The second calculation generated the equivalent information for the E_{T2} variable.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 2.5e-4 & -2.5e-4 & 0 & 0 & 1 & 0 \\ -2.5e-4 & 2.5e-4 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta E_{T1} \\ \Delta E_{T2} \\ \Delta F_{CIR0,T1} \\ \Delta F_{CIR0,T2} \\ \Delta F_{CIR1,T1} \\ \Delta F_{CIR1,T2} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\left.\frac{\partial F_{T1}}{\partial E_{T2}}\right|_{EQV} = \frac{\left.\Delta F_{CIR0,T1}\right|_{CALC}}{\left.\Delta E_{T2}\right|_{CALC}} = -2.5e-4 \qquad \left.\frac{\partial F_{T2}}{\partial E_{T2}}\right|_{EQV} = \frac{\left.\Delta F_{CIR0,T2}\right|_{CALC}}{\left.\Delta E_{T2}\right|_{CALC}} = 2.5e-4$$

$$G_{EQV} = \begin{bmatrix} 2.5e-4 & -2.5e-4 \\ -2.5e-4 & 2.5e-4 \end{bmatrix}$$

Since this equivalent information does not change throughout this analysis, these equivalent calculations are not repeated. Typically, the backplane did not recalculate equivalent sensitivity parameters unless a simulator's sensitivity information changed.

On the second iteration, the simulator solution for partition CIR0 was:

Iteration	CIR0 Simulator Response			Source Values
	Object	Effort	Flow	
2	T1	2.629e-6	-1.249e-4	FSRC = 0.000e0
2	T2	4.996e-1	1.249e-4	FSRC = 0.000e0
2-S.T2	T1	+7.49e-11	-3.559e-9	-
2-S.T2	T2	-1.779e-2	+1.154e-5	-
2-S.T1	T1	-1.944e-4	+9.235e-9	-
2-S.T1	T2	+4.617e-2	+1.152e-5	-

During this iteration, the backplane detected gain when the T1 delta was applied, and a new set of sensitivity parameters was calculated using Equation E-10.

$$\begin{bmatrix} -1.94e-10 & 1.154e-5 & 0 & 0 \\ 7.49e-11 & 1.152e-5 & 0 & 0 \\ 0 & 0 & 7.49e-11 & 1.154e-5 \\ 0 & 0 & -1.94e-10 & 1.152e-5 \end{bmatrix} \begin{bmatrix} \frac{\partial T1:F}{\partial T1:E} \\ \frac{\partial T1:F}{\partial T2:F} \\ \frac{\partial T2:E}{\partial T1:E} \\ \frac{\partial T2:E}{\partial T2:F} \end{bmatrix} = \begin{bmatrix} -9.234e-9 \\ 3.559e-9 \\ 1.7792e-2 \\ -4.617e-2 \end{bmatrix}$$

$$\frac{\partial F_{T1}}{\partial E_{T1}} = 4.86e1 \quad \frac{\partial E_{T2}}{\partial E_{T1}} = 2.375e8 \quad \frac{\partial F_{T1}}{\partial F_{T2}} \approx 0.0 \quad \frac{\partial E_{T2}}{\partial F_{T2}} = 5.50e-4$$

$$\frac{\partial F_{T1}}{\partial E_{T2}} = \frac{\partial F_{T2}}{\partial F_{T2}} = \frac{\partial E_{T1}}{\partial F_{T2}} = \frac{\partial E_{T2}}{\partial F_{T2}} = \frac{\partial F_{T1}}{\partial E_{T2}} = \frac{\partial E_{T2}}{\partial E_{T1}} = 0.0$$

As a comparison, the true gain between E_{T1} and E_{T2} was 2.5e8 (the amplifier gain was 1e6, 10x gain, and 25x gain). The backplane calculated the gain to be 2.375e8 because of the small delta information. This delta information was below the calculation tolerance of the simulators, so the backplane was potentially solving the sensitivity parameters based on rounding errors! Nevertheless, the large gain parameter caused some rounding errors in the other parameters. Since the gain dominated all relationships, the errors in the other parameters were negligible.

On the second iteration, the simulator solution for partition CIR1 was:

Iteration	CIR1 Simulator Response			Source Values
	Object	Effort	Flow	
2	T1	0.000e0	0.000e0	ESRC = 0.000e0
2	T2	0.000e0	0.000e0	ESRC = 0.000e0
2-S.T2	T1	0.000e0	-2.500e-8	-
2-S.T2	T2	+1.000e-4	+2.500e-8	-
2-S.T1	T1	+1.000e-4	-2.500e-8	-
2-S.T1	T2	0.000e0	+2.500e-8	-

The third iteration continued the setup procedure, where the T2 interface of CIR1 tracked the EFFORT output in CIR0. The simulator solution for partition CIR0 was:

Iteration	CIR0 Simulator Response			Source Values
	Object	Effort	Flow	
3	T1	2.629e-6	-1.249e-4	FSRC = 0.000e0
3	T2	4.996e-1	1.249e-4	FSRC = 0.000e0
3-S.T2	T1	+7.49e-11	-3.559e-9	-
3-S.T2	T2	-1.779e-2	+1.154e-5	-
3-S.T1	T1	-1.944e-4	+9.235e-9	-
3-S.T1	T2	+4.617e-2	+1.152e-5	-

The simulator solution for partition CIR1 was:

Iteration	CIR1 Simulator Response			Source Values
	Object	Effort	Flow	
3	T1	2.629e-6	1.249e-4	ESRC = 2.63e-6
3	T2	4.996e-1	-1.249e-4	ESRC = 5.00e-1
3-S.T2	T1	0.000e0	-6.270e-6	-
3-S.T2	T2	-2.508e-4	+6.270e-6	-
3-S.T1	T1	+1.001e-4	-2.500e-8	-
3-S.T1	T2	0.000e0	+2.500e-8	-

On the fourth iteration, the backplane started using the calculated variables from the system matrix. The calculated variable delta values were solved using Equation E-12, or

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 4.86e1 & 0 & 1 & 0.0 & 0 & 0 \\ 2.38e8 & 1 & 0 & 5.5e-4 & 0 & 0 \\ 2.5e-4 & -2.5e-4 & 0 & 0 & 1 & 0 \\ -2.5e-4 & 2.5e-4 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta E_{T1}\big|_{CALC} \\ \Delta E_{T2}\big|_{CALC} \\ \Delta F_{CIR0,T1}\big|_{CALC} \\ \Delta F_{CIR0,T2}\big|_{CALC} \\ \Delta F_{CIR1,T1}\big|_{CALC} \\ \Delta F_{CIR1,T2}\big|_{CALC} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1.249e-4 \\ 1.107e-7 \\ 1.249e-4 \\ -1.249e-4 \end{bmatrix}$$

The new calculated values were defined using Equation E-6:

$$F_{CIR1,T1}\Big|_{CALC}^{3} = F_{CIR0,T1}\Big|_{CALC}^{3} = 0.0 \quad F_{CIR1,T2}\Big|_{CALC}^{3} = F_{CIR0,T2}\Big|_{CALC}^{3} = 0.0$$

$$E_{T1}\Big|_{CALC}^{3} = 2.629e-6 \quad E_{T2}\Big|_{CALC}^{3} = 4.996e-6$$

$$F_{CIR0,T1}\Big|_{CALC}^{4} = -1.249e-4 \quad F_{CIR0,T2}\Big|_{CALC}^{4} = 1.249e-4$$
$$F_{CIR1,T1}\Big|_{CALC}^{4} = 1.249e-4 \quad F_{CIR1,T2}\Big|_{CALC}^{4} = -1.249e-4$$
$$E_{T1}\Big|_{CALC}^{4} = 2.629e-6 \quad E_{T2}\Big|_{CALC}^{4} = 4.996e-6$$

On iteration 4, the final initialization step for the SYSTEM configuration applied only the flow information to the interface, which did cause divergence from the true solution.

Iteration	CIR0 Simulator Response			Source Values
	Object	Effort	Flow	
4	T1	2.627e-6	-1.248e-4	FSRC = -1.25e-4
4	T2	9.988e-1	1.249e-4	FSRC = +1.25e-4
4-S.T2	T1	+1.49e-10	7.1e-9	-
4-S.T2	T2	-3.55e-2	1.154e-5	-
4-S.T1	T1	1.944e-4	-9.235e-9	-
4-S.T1	T2	-4.617e-2	-1.152e-5	-

The simulator solution for partition CIR1 is:

Iteration	CIR1 Simulator Response			Source Values
	Object	Effort	Flow	
4	T1	2.629e-6	1.249e-4	ESRC = 2.63e-6
4	T2	4.996e-1	-1.249e-4	ESRC = 5.00e-1
4-S.T2	T1	0.000e0	-6.270e-6	-
4-S.T2	T2	-2.508e-4	+6.270e-6	-
4-S.T1	T1	+1.001e-4	-2.500e-8	-
4-S.T1	T2	0.000e0	+2.500e-8	-

As the last phase of the initialization procedure, the backplane cleared the FLOW variables before the standard calculations procedures were applied.

$$F_{CIR1,T1}\Big|_{CALC}^{4} = F_{CIR0,T1}\Big|_{CALC}^{4} = 0.0 \quad F_{CIR1,T2}\Big|_{CALC}^{4} = F_{CIR0,T2}\Big|_{CALC}^{4} = 0.0$$
$$E_{T1}\Big|_{CALC}^{4} = 2.629e-6 \quad E_{T2}\Big|_{CALC}^{4} = 4.996e-1$$

In this iteration, the system matrix was:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 4.86e1 & 0 & 1 & 0.0 & 0 & 0 \\ 2.38e8 & 1 & 0 & 5.5e-4 & 0 & 0 \\ 2.5e-4 & -2.5e-4 & 0 & 0 & 1 & 0 \\ -2.5e-4 & 2.5e-4 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta E_{T1} \\ \Delta E_{T2} \\ \Delta F_{CIR0,T1} \\ \Delta F_{CIR0,T2} \\ \Delta F_{CIR1,T1} \\ \Delta F_{CIR1,T2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1.249e-4 \\ 1.107e-7 \\ 1.873e-4 \\ -1.873e-4 \end{bmatrix}$$

Using Equation E-6 and the system matrix solution, the new calculated variable values were:

$$F_{CIR0,T1}\Big|_{CALC}^{5} = -3.123e-4 \quad F_{CIR0,T2}\Big|_{CALC}^{5} = 3.123e-4$$
$$F_{CIR1,T1}\Big|_{CALC}^{5} = -3.123e-4 \quad F_{CIR1,T2}\Big|_{CALC}^{5} = 3.123e-4$$
$$E_{T1}\Big|_{CALC}^{5} = 2.63e-6 \qquad E_{T2}\Big|_{CALC}^{5} = 5.62e-1$$

At this point, the backplane directly applied the calculated values to the simulators. However, the interface procedures had to compensate for the system matrix G_{EQV} (as defined in Appendix B).

$$F_{SRC} = F_{CIR01}\Big|_{CALC}^{4} + G_{EQV} \cdot E\Big|_{CALC}^{4}$$
$$= \begin{bmatrix} -3.12e-4 \\ 3.12e-4 \end{bmatrix} + \begin{bmatrix} 2.5e-4 & -2.5e-4 \\ -2.5e-4 & 2.5e-4 \end{bmatrix} \begin{bmatrix} 2.63e-6 \\ 5.62e-1 \end{bmatrix} = \begin{bmatrix} -1.41e-4 \\ 1.41e-4 \end{bmatrix}$$

Iteration	CIR0 Simulator Response			Source Values
	Object	Effort	Flow	
5	T1	2.627e-6	-1.248e-4	FSRC = -1.41e-4
5	T2	1.061e0	1.248e-4	FSRC = +1.41e-4
5-S.T2	T1	+1.59e-10	-7.54e-9	-
5-S.T2	T2	-3.77e-2	1.154e-5	-
5-S.T1	T1	-1.94e-4	-9.231e-9	-
5-S.T1	T2	-4.615e-2	-1.152e-5	-

The calculated variables were applied to partition CIR1.

Iteration	CIR1 Simulator Response			Source Values
	Object	Effort	Flow	
5	T1	2.629e-6	1.405e-4	ESRC = 2.63e-6
5	T2	5.620e-1	-1.405e-4	ESRC = 5.62e-1
5-S.T2	T1	0.000e0	-7.050e-6	-
5-S.T2	T2	-2.82e-4	+7.050e-6	-
5-S.T1	T1	-1.001e-4	+2.503e-8	-
5-S.T1	T2	0.000e0	-2.503e-8	-

On the sixth iteration, the backplane calculated the true solution. The system level matrix was:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 4.86el & 0 & 1 & 0.0 & 0 & 0 \\ 2.38e8 & 1 & 0 & 5.5e-4 & 0 & 0 \\ 2.5e-4 & -2.5e-4 & 0 & 0 & 1 & 0 \\ -2.5e-4 & 2.5e-4 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta E_{T1} \\ \Delta E_{T2} \\ \Delta F_{CIR0,T1} \\ \Delta F_{CIR0,T2} \\ \Delta F_{CIR1,T1} \\ \Delta F_{CIR1,T2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -9.366e-5 \\ 6.202e-8 \\ 1.093-4 \\ -1.093e-4 \end{bmatrix}$$

The new calculated variable values were:

$$F_{CIR0,T1}\Big|_{CALC}^{6} = -1.249e-4 \quad F_{CIR0,T2}\Big|_{CALC}^{6} = 1.249e-4$$

$$F_{CIR1,T1}\Big|_{CALC}^{6} = 1.249e-4 \quad F_{CIR1,T2}\Big|_{CALC}^{6} = -1.249e-4$$

$$E_{T1}\Big|_{CALC}^{6} = 2.63e-6 \quad E_{T2}\Big|_{CALC}^{6} = 4.996e-1$$

Then, the simulators confirmed the solution.

Iteration	CIR0 Simulator Response			Source Values
	Object	Effort	Flow	
6	T1	2.629e-6	-1.249e-4	FSRC = -4.6e-14
6	T2	4.996e-1	1.249e-4	FSRC = +4.6e-14
6-S.T2	T1	+7.49e-11	-3.559e-9	-
6-S.T2	T2	-1.779e-2	1.154e-5	-
6-S.T1	T1	1.944e-10	-9.235e-9	-
6-S.T1	T2	-4.617e-2	-1.152e-5	-

Iteration	CIR1 Simulator Response			Source Values
	Object	Effort	Flow	
6	T1	2.629e-6	1.249e-4	ESRC = 2.63e-6
6	T2	4.996e-1	-1.249e-4	ESRC = 5.00e-1
6-S.T2	T1	0.000e0	-6.27e-6	-
6-S.T2	T2	-2.51e-4	+6.27e-6	-
6-S.T1	T1	-1.001e-4	+2.503e-8	-
6-S.T1	T2	0.000e0	-2.503e-8	-

All variables meet the tolerance criteria, so the backplane converged. The transient analysis would apply the same procedures, where any variable errors between the simulator and the calculated values in the backplane were eliminated (error minimization).

Appendix E.3 Final Comments

In these linear examples, the backplane initialization sequence actually caused divergence in the solution (Example 2, iteration 4). Of course, the backplane initialization procedures had to be careful not to cause too much divergence. This initialization sequence did improve convergence in certain nonlinear problems, but these routines were not very robust and the routines should be improved in the future! Future initialization procedures should consider using the EFFORT output information to implement a different set of routines to account for any internal stimuli in a gain path. However, the interface into the simulator was ultimately the most important facet of the coupling procedure as shown in Example 2. By incorporating valid condition sensitivity parameters in the CIR0 partition, the coupling solution was significantly simplified and the coupling process easily converged.

The second example demonstrated a very simple fact- the most reliable method of coupling simulators with high gain components was to complete the feedback path. Without defining the feedback path, the local simulator cannot always calculate valid sensitivity information and the initialization solution can have significant errors (Example 2 iteration 1). If the "sufficiently close" conditions required by most iteration routines were not meet, then the backplane cannot solve the problem based on the interfaces into the simulators.

Besides using an interface into the simulator, the other coupling approach would be to transfer the complete solver matrix of each simulator to the backplane. The backplane would construct and solve a single matrix representing ALL simulator information. This approach would require significant information extraction from the simulator and would increase the backplane complexity. Of course, the approach was beyond the scope and intent (simple and easy coupling via an interface) of this development.

VITA

Lloyd Gabriel Clonts was born in Cleveland, Tennessee on April 11, 1968. He attended Bradley County school systems and graduated from Bradley Central High School in May 1986. He entered the University of Tennessee at Knoxville (UTK) in September 1986 and completed the Bachelor of Science degree (Electrical Enginerring) in May 1990. While working as a graduate research assistant at Oak Ridge National Laborities (ORNL), he completed his Masters of Science in May 1993. He entered the Doctoral program at UTK in January 1996 after working for ORNL and several small companies in the Knoxville area. He is currently employed as a CAD tool/system administrator and Application Specific Integrated Circuit (ASIC) designer in the Instrumentation and Control (I&C) division at ORNL.