

**A Methodology for Analog Circuit Design
and Knowledge Transfer.**

by

Jean-Marc G. Patenaude, BAsC, P. Eng.

This thesis is submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of

Master of Engineering

Department of Electronics

Carleton University

Ottawa, Ontario, Canada

April 18, 1996

© Copyright, 1996

Jean-Marc G. Patenaude



National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file *Votre référence*

Our file *Notre référence*

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-612-13848-8

Canada

Name Jean-Marc Patenaude

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Electronics and Electrical

SUBJECT TERM

0544

U·M·I

SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS
 Architecture 0729
 Art History 0377
 Cinema 0900
 Dance 0378
 Fine Arts 0357
 Information Science 0723
 Journalism 0391
 Library Science 0399
 Mass Communications 0708
 Music 0413
 Speech Communication 0459
 Theater 0465

EDUCATION
 General 0515
 Administration 0514
 Adult and Continuing 0516
 Agricultural 0517
 Art 0273
 Bilingual and Multicultural 0282
 Business 0688
 Community College 0275
 Curriculum and Instruction 0727
 Early Childhood 0518
 Elementary 0524
 Finance 0277
 Guidance and Counseling 0519
 Health 0680
 Higher 0745
 History of 0520
 Home Economics 0278
 Industrial 0521
 Language and Literature 0279
 Mathematics 0280
 Music 0522
 Philosophy of 0998
 Physical 0523

Psychology 0525
 Reading 0535
 Religious 0527
 Sciences 0714
 Secondary 0533
 Social Sciences 0534
 Sociology of 0340
 Special 0529
 Teacher Training 0530
 Technology 0710
 Tests and Measurements 0288
 Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS
 Language
 General 0679
 Ancient 0289
 Linguistics 0290
 Modern 0291
 Literature
 General 0401
 Classical 0294
 Comparative 0295
 Medieval 0297
 Modern 0298
 African 0316
 American 0591
 Asian 0305
 Canadian (English) 0352
 Canadian (French) 0355
 English 0593
 Germanic 0311
 Latin American 0312
 Middle Eastern 0315
 Romance 0313
 Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY
 Philosophy 0422
 Religion
 General 0318
 Biblical Studies 0321
 Clergy 0319
 History of 0320
 Philosophy of 0322
 Theology 0469

SOCIAL SCIENCES
 American Studies 0323
 Anthropology
 Archaeology 0324
 Cultural 0326
 Physical 0327
 Business Administration
 General 0310
 Accounting 0272
 Banking 0770
 Management 0454
 Marketing 0338
 Canadian Studies 0385
 Economics
 General 0501
 Agricultural 0503
 Commerce-Business 0505
 Finance 0508
 History 0509
 Labor 0510
 Theory 0511
 Folklore 0358
 Geography 0366
 Gerontology 0351
 History
 General 0578

Ancient 0579
 Medieval 0581
 Modern 0582
 Black 0328
 African 0331
 Asia, Australia and Oceania 0332
 Canadian 0334
 European 0335
 Latin American 0336
 Middle Eastern 0333
 United States 0337
 History of Science 0585
 Law 0398
 Political Science
 General 0615
 International Law and Relations 0616
 Public Administration 0617
 Recreation 0814
 Social Work 0452
 Sociology
 General 0626
 Criminology and Penology 0627
 Demography 0938
 Ethnic and Racial Studies 0631
 Individual and Family Studies 0628
 Industrial and Labor Relations 0629
 Public and Social Welfare 0630
 Social Structure and Development 0700
 Theory and Methods 0344
 Transportation 0709
 Urban and Regional Planning 0999
 Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES
 Agriculture
 General 0473
 Agronomy 0285
 Animal Culture and Nutrition 0475
 Animal Pathology 0476
 Food Science and Technology 0359
 Forestry and Wildlife 0478
 Plant Culture 0479
 Plant Pathology 0480
 Plant Physiology 0817
 Range Management 0777
 Wood Technology 0746
 Biology
 General 0306
 Anatomy 0287
 Biostatistics 0308
 Botany 0309
 Cell 0379
 Ecology 0329
 Entomology 0353
 Genetics 0369
 Limnology 0793
 Microbiology 0410
 Molecular 0307
 Neuroscience 0317
 Oceanography 0416
 Physiology 0433
 Radiation 0821
 Veterinary Science 0778
 Zoology 0472
 Biophysical
 General 0786
 Medical 0760
EARTH SCIENCES
 Biogeochemistry 0425
 Geochemistry 0996

Geodesy 0370
 Geology 0372
 Geophysics 0373
 Hydrology 0388
 Mineralogy 0411
 Paleobotany 0345
 Paleocology 0426
 Paleontology 0418
 Paleozoology 0985
 Palynology 0427
 Physical Geography 0368
 Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES
 Environmental Sciences 0768
 Health Sciences
 General 0566
 Audiology 0300
 Chemotherapy 0992
 Dentistry 0567
 Education 0350
 Hospital Management 0769
 Human Development 0758
 Immunology 0982
 Medicine and Surgery 0564
 Mental Health 0347
 Nursing 0569
 Nutrition 0570
 Obstetrics and Gynecology 0380
 Occupational Health and Therapy 0354
 Ophthalmology 0381
 Pathology 0571
 Pharmacology 0419
 Pharmacy 0572
 Physical Therapy 0382
 Public Health 0573
 Radiology 0574
 Recreation 0575

Speech Pathology 0460
 Toxicology 0383
 Home Economics 0386

PHYSICAL SCIENCES
 Pure Sciences
 Chemistry
 General 0485
 Agricultural 0749
 Analytical 0486
 Biochemistry 0487
 Inorganic 0488
 Nuclear 0738
 Organic 0490
 Pharmaceutical 0491
 Physical 0494
 Polymer 0495
 Radiation 0754
 Mathematics 0405
 Physics
 General 0605
 Acoustics 0986
 Astronomy and Astrophysics 0606
 Atmospheric Science 0608
 Atomic 0748
 Electronics and Electricity 0607
 Elementary Particles and High Energy 0798
 Fluid and Plasma 0759
 Molecular 0609
 Nuclear 0610
 Optics 0752
 Radiation 0756
 Solid State 0611
 Statistics 0463
 Applied Sciences
 Applied Mechanics 0346
 Computer Science 0984

Engineering
 General 0537
 Aerospace 0538
 Agricultural 0539
 Automotive 0540
 Biomedical 0541
 Chemical 0542
 Civil 0543
 Electronics and Electrical 0544
 Heat and Thermodynamics 0348
 Hydraulic 0545
 Industrial 0546
 Marine 0547
 Materials Science 0794
 Mechanical 0548
 Metallurgy 0743
 Mining 0551
 Nuclear 0552
 Packaging 0549
 Petroleum 0765
 Sanitary and Municipal System Science 0554
 System Science 0790
 Geotechnology 0428
 Operations Research 0796
 Plastics Technology 0795
 Textile Technology 0994

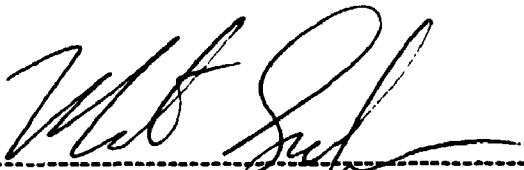
PSYCHOLOGY
 General 0621
 Behavioral 0384
 Clinical 0622
 Developmental 0620
 Experimental 0623
 Industrial 0624
 Personality 0625
 Physiological 0989
 Psychobiology 0349
 Psychometrics 0632
 Social 0451



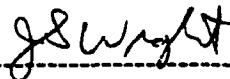
The undersigned recommend to the Faculty of Graduate Studies and Research the acceptance of the thesis:

“A Methodology for Analog Circuit Design and Knowledge Transfer.”

submitted by Jean-Marc G. Patenaude, P. Eng. in partial fulfillment of the requirements for the degree of Master of Engineering.



Professor Martin Snelgrove
Thesis Supervisor



Professor J.S. Wight
Chairman,
Department of Electronics

Abstract

This thesis describes an attempt to develop an improved methodology for designing analog circuits using the traditional SPICE simulator coupled with Maple[®], a mathematical programming language with an efficient user interface. The traditional SPICE-based approach to design circuits is examined and its drawbacks are exposed. A better solution is proposed where the circuit design knowledge is developed using symbolic analysis routines programmed in Maple[®], while it is captured using the executable Maple[®] worksheet interface, thus providing a vehicle for efficient knowledge transfer and reuse by fellow engineers. A C language program provides a link between the Maple[®] engine and Eldo[®], a SPICE-like circuit simulator. An operational amplifier is designed using the Maple-Eldo prototype tool in the case study section. It is concluded that the tool is useful in increasing the designer's productivity and understanding of the circuit's behaviour, while providing a means to capture and reuse frequency domain circuit design knowledge through the Maple[®] worksheets¹.

1. For simplicity, the registered trademark symbol ® has been omitted in all subsequent references to Eldo and Maple in this thesis.

Acknowledgements

I would like to express my gratitude to my thesis supervisor, Dr. M. Snelgrove, for his guidance and support during the course of this M. Eng. program.

I would also like to thank my former employer, Bell-Northern Research and Northern Telecom Ltd. - Semiconductor Components Group, in Ottawa, for providing access to their technology, computing resources, and CAD software tools. In particular, I would like to thank my manager, Mr. G. Hupé, for his help and cooperation during the early stages of this thesis. Without this cooperation from my employer and co-workers, none of this research work would have been possible.

In addition, I would like to thank the following individuals:

- Mr. G. Quesnel, for being a mentor and for providing the 'seed' to many ideas in this thesis.
- The late Dr. N. Battersby, for providing inspiration and for being a friend.
- Mr. J. Carette, for his help and guidance in implementing 'good programming practices' in the Maple programming language.
- Ms. D. Hagglund, for providing technical support on Maple related questions.

Finally, I would also like to thank my current employer, LSI Logic Corporation of Canada, for providing access to their computing resources, and for allowing the time off work to complete this thesis.

TABLE OF CONTENTS

1.0	Introduction.....	1
1.1	Importance of productivity, design quality and predictability	2
1.2	Analog design automation	3
1.2.1	Motivation	3
1.2.2	Analog standard cells	4
1.3	The proposed tool	6
1.3.1	Analog circuit design and reuse using the proposed tool	6
1.4	Overview of the dissertation	9
2.0	Overview of Existing Circuit Design Approaches.....	11
2.1	The traditional SPICE-based approach.....	11
2.1.1	The general circuit design flow	12
2.1.2	Multiple design issues lead to graph complexity	14
2.1.3	Extensive usage of SPICE	16
2.1.4	Design knowledge reuse and SPICE.....	17
2.2	Analog circuit design automation	18
2.2.1	Analog circuit design knowledge	19
2.3	Overview of the literature:	22
2.4	Overview of the literature relating to symbolic analysis	27
3.0	The Proposed Method.....	29
3.1	Goals of the analog design automation tool.....	29
3.2	The conceptual analog design automation tool.....	31
3.2.1	Generally accepted design practices and methods	31
3.2.2	Important design activities	32
3.2.3	Features of the analog design automation tool	35
3.2.4	The power of symbolic and hybrid circuit analysis.....	36
3.2.5	Efficient design space exploration in the transformed domain	39

3.3	The Maple-Eldo prototype.....	47
3.3.1	Maple V and its features for hybrid symbolic analysis	47
3.3.2	The design flow using the Maple-Eldo prototype	48
3.3.3	Hybrid symbolic analysis in Maple	56
3.3.4	Small signal analysis in Maple	57
3.3.5	Effects of large signal parameters	61
3.3.6	PVIT variations analysis.....	65
3.3.7	Small signal modelling issues	68
3.3.8	Knowledge capture using the Maple-Eldo prototype	68
3.4	Using the Maple-Eldo prototype.....	71
3.4.1	The UNIX scripts and the Maplenet C program	71
3.4.2	The Maple smallsignal package	76
3.4.3	The mnasolve routine	76
3.4.4	The analysis routines	79
3.4.5	The support routines	86
3.5	Automated verification and design centering using simulations	87
3.5.1	Design centering and circuit characterization	87
4.0	Case Study - Amplifier Design	91
4.1	Basic two stage operational transconductance amplifier	91
4.1.1	Functional Description of the amplifier	92
4.1.2	Initial design effort	94
4.2	Design of the amplifier using the Maple-Eldo tool.....	100
4.2.1	Maple worksheet for initial analysis.....	100
4.2.2	Root placement optimization and design centering.....	114
4.2.3	Conclusions	115
5.0	Evaluations and Conclusions	117
5.1	Summary and discussions	117
5.2	Recommendations for future work	119
6.0	REFERENCES	123

APPENDIX A	Examples of Important Files	126
A.1	Transistor-level netlist: 'netlist'	126
A.2	User-modified 'netlist.ss'	126
APPENDIX B	Maple worksheet.....	130
APPENDIX C	Selected code from the smallsignal package	150
APPENDIX D	Autopvit system	156

LIST OF FIGURES

Figure 1	Traversing the design flow graph.....	14
Figure 2	Hierarchy of Design Knowledge.....	21
Figure 3	The iterative design flow using a simulator as a verification tool.....	40
Figure 4	The generalized small signal model of the MOS transistor.....	42
Figure 5	The iterative design flow with flexible design space exploration.....	45
Figure 6	First phase of the design flow using the Maple-Eldo prototype.....	49
Figure 7	Format of the 'lookup_table' file.....	52
Figure 8	The MOS small signal equivalent model used in the Maple-Eldo prototype.....	67
Figure 9	Circuit design/optimization through design space exploration.....	69
Figure 10	Format of the 'netlist.ss' file for a MOS transistor instantiation.....	73
Figure 11	Invoking the 'mnasolve' routine within the Maple worksheet.....	77
Figure 12	Example of the output produced by the 'reduce_order' routine.....	80
Figure 13	Example of the output produced by the 'print_sensitivities' routine.....	81
Figure 14	Output produced by the 'deltagen' routine.....	82
Figure 15	Example of the 'rootlocus' plot using the 'inseq' option.....	85
Figure 16	Example of the output generated by 'expression_sensitivities'.....	86
Figure 17	Example of the autopvit.out report file.....	90
Figure 18	Example of a simulation corner case as presented in the 'pvitcases' file.....	90
Figure 19	Schematic of the two-stage operational amplifier.....	93
Figure 20	Pole splitting effect when $c[10]$ is increased.....	99
Figure 21	Miller zero moving to a lower frequency as $r[13]$ is increased.....	99
Figure 22	Phase and magnitude Bode plots for the initial analysis of the amplifier.....	102
Figure 23	Root placement of the amplifier with initial values as shown in Figure 19.....	103
Figure 24	Root locations.....	104
Figure 25	Root and DC gain Sensitivities.....	104
Figure 26	Root loci for values of $r13 = \{500, 665, 3120\}$ Ohms.....	107
Figure 27	Parametric Bode plots. Parameter $r[13] = \{500..5000\}$	108
Figure 28	Root and DC gain locations and parametric sensitivities for $r13 = 1200$	109
Figure 29	Sensitivities to the length and width when $r13 = 1200$	111
Figure 30	Sensitivities to the size (correlated length and width) when $r13 = 1200$	112

1.0 Introduction

Although integrated analog circuit design has existed since the invention of the integrated circuit in the early 1970's, today's engineers still design such circuits with the same 'ad-hoc' approach they used back then. It is clear of course, that many specialized tools have come into existence over the past 20 years to help in the design of specific types of analog circuits. However, when it comes to plain analog circuit design, such as amplifiers, regulators, comparators, and the like, designers tend to rely mostly on their preferred commercial version of SPICE, the analog circuit simulator originally developed at the University of California at Berkeley in the 1960's. An important shortcoming of this approach is that designers find it difficult to explore the design space efficiently because SPICE does not provide any interpretation of the simulated results, nor does it provide any sense of history between a collection of simulation results. The design knowledge is typically developed manually by the designer while being loosely captured in his engineering notebook. This can impair the engineer's productivity and can result in increased costs and slower time-to-market for the corporation. This thesis is an attempt to improve this situation using a more systematic approach to design analog circuits in the frequency domain than the currently used 'ad-hoc' method using SPICE alone. The methodology integrates a SPICE-like simulator with a mathematical analysis tool and a worksheet user interface, thus making self-documentation possible during design and engineering trade off analysis. The implementation is a proof-of-concept tool constructed by writing code to combine the worksheet and analysis functions of one piece of commercial software (Maple) with the accurate MOS transistor models (Northern Telecom's MISNAN) and the numerical circuit simulation capabilities of another (Eldo). It is demonstrated for frequency domain analysis

only, for a typical industrial design problem. A full commercial implementation would integrate the function in a single tool with improved numerical and symbolic analysis algorithms.

1.1 Importance of productivity, design quality and predictability

Increasing competition in today's semiconductor marketplace is forcing corporations to find better ways to improve designer's productivity, circuit design quality, and predictability in their product development cycles. The relevance of each of these aspects cannot be overstated [1]. A more productive designer can turn out a product using less resources and design lead time, which translate into higher corporate profits through the early acquisition of a larger market share. A better quality product can lead to higher performance than its competitor's offerings. In terms of correctness, a better quality product can translate into substantial savings in costs required to fix design errors, and in loss of time and reputation caused by the mistakes. Further, more predictable design lead time and product performance can lead to more accurate product, project and schedule planning, and can provide substantial leverage for timely product introduction into the marketplace and for enhancement of the corporate image. Finally, predictability can also help reduce the technical risks inherent in a technical design project, since early and accurate product performance assessments can provide the information necessary to properly evaluate the product's requirements, and its technical feasibility.

1.2 Analog design automation

1.2.1 Motivation

With the ASIC revolution of the mid-1980's, the previously held view in the industry where most relevant analog functions would be replaced by some digital equivalent has changed [2]. It is true indeed, that much of the traditional analog functionality has since been implemented in the digital domain, but at the same time, a higher on-chip complexity and integration level, combined with the need for higher performance, often implied that chips must be mixed-signal in their nature. With the current drive for even higher levels of integration, and with more consumer applications requiring complex analog human interfaces, it is reasonable to expect that this trend will at the very least sustain itself, if not make mixed-signal ASICs a much sought-after commodity in the near future.

This widely held view of the late 1970's and early 1980's pushed the digital circuit design community to efficiently address its need for design automation. Design methodologies which included row-based layout from standard cell libraries [3], semi-custom place and route [4], and logic synthesis with technology mapping [5] have been developed, and provided a relatively clear and reliable path for the designer to map functional ideas into a working silicon solution. At the same time, however, the success of the digital circuit design world made it more obvious that the analog circuit design community had a serious lack of comparable semi-custom and full-custom design tools.

Hence, the growing presence of analog functions on mixed-signal integrated circuits, combined with the ever increasing market pressure to decrease the design time for new mixed-signal ASICs, are currently forcing industry and academy to address and resolve a range

of challenging problems in the analog design automation world. The goal is to try to improve the designer's productivity, while ensuring a high standard for design quality and circuit performance. The improved predictability can play a key role in the corporation as it can help designers, their management, and other key corporate functions such as product line management and marketing, in their effort to better plan their resource utilization and to assess more accurately a timely introduction of the product into the marketplace.

1.2.2 Analog standard cells

Since digital standard cells have proven to be robust and universally useful in the digital circuit design world, one would be led to believe that similar results could be obtained with analog standard cell libraries. Although analog standard cells can be proven to be useful, reused cells often require further customized sections, due to certain performance specifications particular to a given mixed-signal ASIC project. In general, a mixed-signal ASIC can be expected to find 70%-90% of its analog cells in a good library [2]. However, the additional design effort required to develop the much-needed custom analog cells can have an enormous impact on a project with short time-to-market constraints, especially when the custom analog circuits are pushing the limits of the process technology in terms of performance. Moreover, developing a good analog standard cell library can be very time consuming, given the relatively large pool of different types of analog circuits required in mixed-signal ASIC design projects. The amount of effort underlying the full development of an analog standard cell library is further compounded by the accelerated evolution of process technologies, and by the business need of corporations to provide sec-

ond and third sourcing of their silicon products into silicon fabrication facilities with slightly different process parameters.

The challenges posed by analog standard cell libraries are therefore much greater than their digital counterparts. The production volumes, or usage, of analog standard cells are generally much less than the usage of a digital standard cell library, given that most of today's ASICs are completely digital. Further, the effort required to develop a useful analog cell library is greater than for a digital cell library, given that each analog cell tends to have more complexity than a basic digital gate. It is also likely that a good analog cell library will have a larger number of cells than a comparable digital cell library, thus compounding the effort required for its development. Finally, the need for customized sections of analog cells on a per project basis reduces even further the payback expected from the full development of a complete analog standard cell library.

The above arguments provide a basis for contemplating a different approach than a full standard cell library for increasing analog circuit design productivity while minimizing costs. Although there is always a need for some kind of analog cell library, for such basic cells as generic operational amplifiers, comparators, and the like, there is still a need for productivity improvement in the way analog circuit designers develop custom analog cells. The goal of analog circuit design automation is to address this issue by providing a tool box and by suggesting a method which will make the analog circuit designer more productive, and less prone to design errors, or to uncovered but yet important design issues.

1.3 The proposed tool

At present analog circuit design involves a great deal of duplication of effort, with designers almost repeating one another's work (or earlier efforts of their own) on particular types of circuits. This problem has proven difficult to solve because of the diversity of issues pertaining to analog circuit design.

An analog designer typically alternates between hand symbolic analysis, which makes for a thorough understanding of simplified circuit models, and numeric simulations, which is used to verify that the simplifications required during hand analysis are warranted. She also keeps a written record of design decisions, so as to be able to review the overall progress as the knowledge is developed. The tool proposed integrates these three key elements - workbook, analysis and simulations - in a single tool.

This strategy is desirable to the designer because it provides for a proven and familiar design methodology. It is preferable to the existing method, even for a single design, because automating and integrating symbolic analysis allows the use of higher order models and faster analysis cycles, and rapid exploitation of such 'textbook' tools as root locus plots, for example. It is even better when designs must be revisited at a later date because of the inclusion of the workbook aspect in the tool, which provides a vehicle for knowledge capture and transfer.

1.3.1 Analog circuit design and reuse using the proposed tool

Many useful tools exist in the marketplace to help the designer achieve a specific goal very efficiently. Signal processing analysis tools such as Matlab[®] or Comdisco[®], for example,

allow the designer to analyze and understand the theory behind filters and data converters. At the level of circuit design, however, the SPICE circuit simulator is probably the most commonly used circuit verification tool used by the analog circuit design community. Since simulation fills such a fundamental need in circuit design verification, most analog design tool vendors provide their own version of the SPICE simulator, integrated within their own framework for waveform display, design capture, and physical layout capture and verification.

One common problem with simulators is that they display the resulting response of a circuit without providing much information about how the circuit operates, or why it behaves in a particular manner under certain conditions. As a consequence, simulators alone provide little insight on how to improve the performance of a given circuit. Designers thus rely on observation, hand analyses, experience, and the like in order to better understand the operation of the circuit. It is only after a good grasp of the fundamentals of the circuit that a designer can try to modify and optimize the circuit for a particular set of specifications. The effort and time required to obtain this '*design knowledge*' is often very substantial, and thus costly to the corporation in terms of funding and missed opportunities due to a longer time-to-market. Hence, productivity and quality become a mutual trade off in the business, because the view is often one where engineers are forced to be less thorough in their analyses in order to stay on track with an aggressive schedule. A side result of this is often an impaired predictability, where designs often fail in the latter stage of the design cycle due to unforeseen problems.

One way to address the problem of circuit knowledge development and reuse is by means of integrating accurate SPICE simulations with the concept of symbolic analysis and the idea of an executable worksheet user interface. The term symbolic analysis implies that the circuit under investigation is described in terms of algebraic equations. The results of the SPICE simulations are needed in order to analyze the circuit under realistic operating conditions. Thus, all the information describing the electrical behavior of the circuit should be contained within this set of accurate algebraic equations, and the table of accurate symbolic parameter values provided by the SPICE simulations. From a circuit design knowledge point of view, the issue is to extract the relevant design information from these equations, and to present it to the designer in a way that allows her to fully understand the important design considerations and trade-offs. Moreover, these equations can be analyzed and pasted into some form of computerized document, or 'worksheet'. This executable worksheet interface can be used to keep track of the design information as it is developed, so that it can be easily transferred to another engineer at a later time, thus addressing the need for knowledge transfer and reuse. One limitation to this approach is that the resulting equations must be simple enough to be manageable and understandable by the designer. Another limitation is that only closed-form analytic equations can be made easily manageable and, in general, such closed-form analytic equations do not exist for all type of analog circuit analyses. In particular, time-domain circuit analyses generally involve implicit equations which can only be solved through iterative methods. On the other hand, when a circuit is linearized at a given operating point, a set of closed-form equations always results. Symbolic analysis then applies particularly well to this special case, and can therefore be used to perform small signal DC and frequency analysis of electronic circuits. A

DC operating point simulation can be performed with SPICE to obtain an accurate table of symbolic parameter values for numeric substitutions, as may be required by graphics or simplification of the resulting symbolic expression.

1.4 Overview of the dissertation

The value of the prototype tool is demonstrated by using it for a typical industrial design problem: The design of a two-stage MOS operational amplifier, optimized for fast settling. This is a problem addressed in general terms by many textbooks and tutorial papers [24][27][28][29]. Textbook models are simplified, however, so they provide little more than a starting point, especially for high performance designs, which occurs frequently in the industry. In such circuits, the important design knowledge involves multiple trade-offs between second-order effects which are unfortunately rarely discussed in the literature. The case study demonstrates how the proposed tool can be used to accurately and efficiently explore an amplifier's frequency domain performance, and its applicable design space. In particular, it is shown how a designer can easily and quickly understand the trade-offs between important second order effects affecting the amplifier's stability. A Maple worksheet capturing this design effort has been developed and is discussed in the case study section of this thesis. Furthermore, a set of UNIX scripts have been written to characterize and tabulate the frequency domain performance of the amplifier over relevant simulation corners. This script was written in a format that is generic enough to characterize the circuit over any relevant corner as dictated by the application. This programmability is an important aspect, as amplifiers require different sets of simulation corners than voltage regulators, for example.

The limitations of existing popular tools such as SPICE are examined in details in chapter 2, with the goal of developing an improved method. In chapter 3, symbolic analysis using the Maple V software package is introduced since it provides an efficient tool to develop the routines necessary to perform the required mathematical analysis addressing the special case of small signal analysis and design. Symbolic analysis is used in this context as a circuit design tool, helping the designer to gain insight into the circuit studied, while the Maple V interface provides a platform to capture, reuse and transfer the circuit design knowledge developed through the executable worksheet interface. The recommended methodology is then implemented in a prototype which integrates the Maple V package with Eldo, a SPICE-like circuit simulator. The case study is presented in chapter 4, whereas the value of the tool is evaluated in chapter 5. Conclusions and recommendations for future work are also exposed in this chapter.

2.0 Overview of Existing Circuit Design Approaches

The art of analog and mixed-signal ASIC design is generally not considered to be a trivial task. Since each circuit design project has its own set of design issues, there is, generally speaking, no single optimal and complete design solution yielding the absolute best circuit performance and robustness. As a consequence, designers tend to approach each particular design problem in their very own ways, which makes it very difficult for anyone to draw general conclusions aiming at providing a definition of the exact best design methodology.

In a practical sense, methodologies depend upon a designer's experience, the circuit topology considered, the set of design specifications, and the environmental operating conditions. Hence, a detailed discussion about any given state of the art design approach is deemed futile, as no general conclusions could be drawn from such details.

Instead, it shall be considered appropriate to consider the more general design flow, skipping the project-specific details, in order to help define relatively accurately the thoughts and design practices experienced by the designer during the design process. Although an imperfect generalization, this approach should provide enough information for proper guidance towards the development of a useful analog circuit design and reuse methodology.

2.1 The traditional SPICE-based approach

One of the first step typically performed by an analog circuit designer is to define the analog functions of the mixed-signal integrated circuit as seen from the system's perspective. This phase alone can require substantial efforts from the designer's part, as the underlying product requirements are better understood. Although this is clearly an area where an im-

provement in efficiency is important and possible, this thesis will not try to address that phase of the design challenge.

2.1.1 The general circuit design flow

The SPICE simulator is normally used extensively at the circuit block level. In general, designers tend to iterate several times as they weave through the numerous design possibilities. They may choose to make specific design decisions based on certain assumptions, which they may later try to validate. Care is taken in making assumptions which are realistic and result in a reasonable 'hand solution'. The designer may also choose to limit the number of degrees of freedom in his design because some validations need to be performed by manual and possibly tedious analytic methods. The design process can therefore be seen as traversing a complex and fluid design decision graph, where numerous iterations can occur around each node. Figure 1 on page 14 provides an abstract illustration of such a design decision graph, and its traversal by the circuit designer.

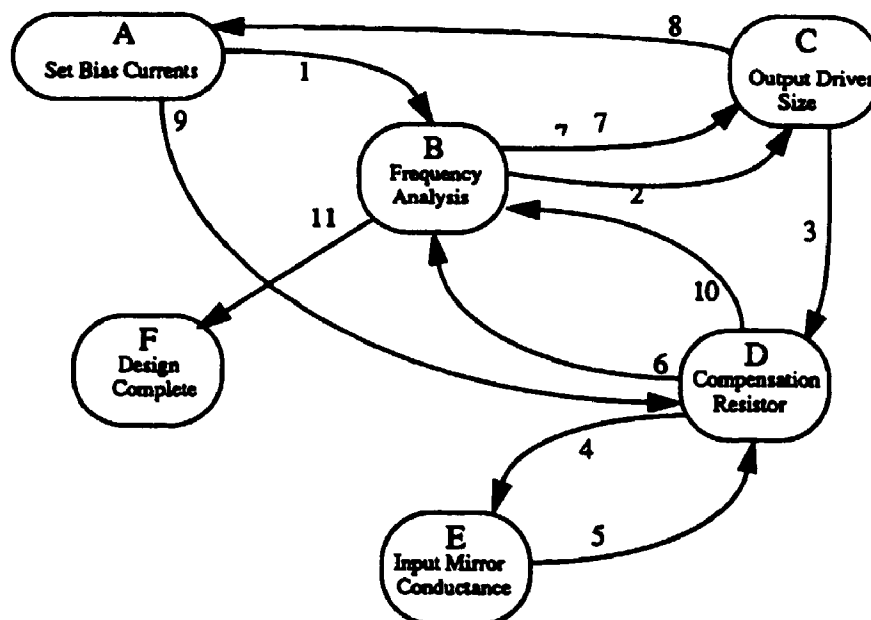
Consider each node in the graph to represent a design activity, whether a design decision, an analysis, or an assumption. Each directed line in the graph represents the time sequence upon which each node is invoked. Thus, the designer may start by making an assumption at node A. For the purpose of this discussion consider the design of an amplifier. The assumption at node A corresponds to fixing the input and output stage bias currents, possibly based upon the minimum slew rate the designer needs while assuming a 1 pF compensation capacitor between the first and second stage. As the designer progresses through her design, she will move to node B (step 1), which corresponds to performing a frequency analysis of her amplifier, and then to node C (step 2), which corresponds to resizing the

second stage output driver transistor. She will then move on to consider other issues, such as the value of the compensation resistor and the output conductance of the input stage current mirrors, which corresponds respectively to nodes D and E (step 3 and 4). She may then decide to go back to node D (step 5) before she chooses to perform another frequency analysis at node B (step 6). This may lead her to reconsider the size of the output drivers at node C (step 7), or even to come back to her original bias current assumptions at node A (step 8). She might have discovered that the 1pF compensation capacitor is too small for what she is trying to achieve, and thus, by increasing the compensation capacitor, she might be forced to also increase the bias current in order to maintain a proper slew rate. Eventually, at node F (step 11), she will reach a point where she feels that the design is reasonably optimal for her application. The point of this graph is that it provides a simplified representation of the activities of the designer as a function of time. Since the designer always re-evaluates the complete design at any step in the design graph (at any node), it is therefore clear that predicting how the designer inter-relates each of these activities is both futile and useless, because it depends on too many factors. Depending upon the set of specifications, the technology, the circuit topology, and many other factors, the preferred design procedure pursued by a particular designer will vary substantially between each case.

It is important to recognize that the designer needs and wants the ability to traverse the design space with maximum flexibility. This is a fundamental requirement of the analog circuit design process. As each designer has his own way of approaching a design problem, and as each circuit structure has its own set of relevant design issues, it is reasonable to assume that the design flow graph for any particular design problem will very rarely be iden-

tical, if ever. It follows that flexibility constitutes a fundamental feature required by any successful and useful analog circuit design and reuse method. Stated differently, the user needs the ability to specify his own design decision graph as he explores and understand the circuit's design space. A hard-coded expert would therefore defeat this purpose. This is a very important aspect of design which is traditionally served well via the engineer's workbook.

Figure 1 Traversing the design flow graph



2.1.2 Multiple design issues lead to graph complexity

It is now evident that the design flow graph of a typical circuit is normally very complex and difficult to predict as the designer needs to re-evaluate the entire circuit after each design consideration. To make matters worse, many design issues are addressed several

times within the graph, as part of multiple iterative loops. It is important to realize why this is the case. The short answer to this question is that there are a large set of unknowns to consider simultaneously during the design of the circuit. For example, there are, in a typical circuit, relationships trading off all of the following design issues, as well as many other considerations:

- power consumption
- bias currents
- operating voltages
- loop stability
- process variations of all active and passive devices
- operating conditions such as environmental noise coupling
- noise coupling/rejection from nearby circuits
- mismatch effects over process variations
- environmental conditions such as temperature
- block size, and other cost-related issues
- manufacturing and parametric yield issues
- non-ideal (non-linear) effects of linear devices (resistors, capacitors)
- parasitic capacitances of large resistors
- non-ideal and parasitic effects of active devices
- production testing issues
- digital noise coupling through the substrate
- etc, etc, etc...

In general, most of these effects, including their absolute and relative variations, must be considered simultaneously during the design process in order to build the required robustness into the circuit block. Since it is impossible to address all issues at once, designers tend to start with a set of assumptions, usually taken from experience, and make their best

effort to understand the circuit. The local iterative loops are thus required as the designer often needs to re-validate or modify a previous design assumption. Through such iterative looping at various levels, the designer eventually reaches a consensus point where she believes that the circuit has the required robustness, and is well optimized for the targeted application. In summary, there is therefore no point in trying to find a simple enough model to automate the design process. The circuit design activities clearly need to be designer driven. A useful method therefore needs to encapsulate this flexibility.

2.1.3 Extensive usage of SPICE

Since there are so many design issues to consider simultaneously, designers tend to make certain assumptions, followed by the verification of these assumptions through simulations using SPICE. By constantly alternating between circuit modifications and simulations, the designer tests his assumptions, and eventually moves closer to the region where the design will be considered as complete and optimal. At this juncture, one must realize the intent of the SPICE simulator. Since it only simulates the behaviour of a circuit under the conditions specified by the designer, it provides a means for verification, and as such, it is not a design tool, in the sense that a design tool ought to tell the designer how and why a circuit behaves in a particular manner. But because such true circuit design tools do not exist in the marketplace, designers end up using SPICE more extensively than they ought to. In particular, designers often use SPICE to collect a series of plots and graphs, where the circuit has been exercised under slightly different conditions. By studying these observations, the designer tries to understand what is happening within the circuit. By coupling these observations with theories and empirically observed phenomena, the designer tries

to draw a conclusion relating the observations to the theory. The result, although generally true, is often not obtained without extensive efforts since many simulations are required, and the links between theory and SPICE observations are not always obvious. In fact, it is often very difficult to correlate the observations to the theory, because the links are very complex and may involve many parameters. The major problem in this case is not so much the time required to accumulate the plots through multiple simulations, but rather the fact that SPICE does not provide any direct insight in how or why the circuit works. Again, one must realize that this was not the intent of SPICE, as it was originally intended to be used as a design verification tool by providing simulation results.

2.1.4 Design knowledge reuse and SPICE

Since SPICE was never intended to be a true circuit design tool, in the sense of telling the designer why or how the circuit works, it follows that it is not the best vehicle to capture and later reuse the resulting design knowledge. Most of the design knowledge is in the designer's mind, and SPICE simply generates plots at each simulation run, for analysis by the designer. Moreover, SPICE does not provide any sense of history, whereas design knowledge in the designer's mind evolves with time. From a design knowledge development, capture and reuse point of view, SPICE is very much incomplete. A framework which focuses on developing and capturing the design knowledge as it is learned by the designer would be more appropriate. This framework, which could be in the form of an executable document or a worksheet, could be read, understood and executed by another designer, thus providing a means upon which the design knowledge is stored, reused and transferred to another designer. Clearly, SPICE does not provide any such facility. Howev-

er, since design verification forms an integral part of the design cycle, it remains imperative to hook up SPICE into such a framework in order to have a tool that is complete in the sense that it can help the designer during both the design and circuit verification phases.

2.2 Analog circuit design automation

The analog circuit design and reuse challenge is really a subset of the larger analog circuit design automation problem. Since all aspects of analog design automation are inter-related, it is important to understand what are the basic elements upon which analog circuit design automation is based. In general, it can be divided into three lines of design effort: Synthesis, analysis and verification. Synthesis comprises the effort required to create circuit topologies and to size their devices, as well as the development of a set of system specification and its practical implementation. Synthesis efforts include both layout synthesis and circuit synthesis. Analysis, on the other hand, is normally done using analytic equations or empirical rules in order to predict the behavior of the circuit or the layout. Finally, verification is typically done via simulations, thus providing information to the designer regarding the functionality and robustness of the circuit. These three important tasks are normally tightly intermingled during the design flow, with iterations at all levels. As the designer learns more about the circuit topology using the feedback provided by analytical and verification tools, he can then feed this more accurate information back into the synthesis task. A good design tool should therefore provide the openness and the flexibility to let the designer interact at all levels within the design flow, and therefore influence the direction of the analysis, synthesis, and verification efforts.

2.2.1 Analog circuit design knowledge

Before trying to address the issue of analog design automation more completely, the nature of analog circuit design knowledge needs to be defined. In essence, this can roughly be divided into three different categories:

1. Analytic design knowledge
2. Empirical design knowledge
3. Executable design knowledge

The analytic design knowledge usually takes the form of design equations modelling various aspects of a circuit's behavior. For instance, the small signal model of an active device can be regarded as analytic design knowledge. Similarly, large signal design equations, distortion or noise equations, as well as the approaches or procedures followed by the designer to analyze simulation results can all be considered as analytic design knowledge. In general, analytic design knowledge is anything that can be thought of as helping the designer directly analyze a certain physical behavior of the analog circuit.

Empirical design knowledge effectively complements the analytic design knowledge pool of information. It includes any observations or rules-of-thumb that a designer may be using to help in the analysis of a circuit. For example, a designer may start a design by fixing the channel length of certain MOS current sources to a given value, because from her experience, such a channel length has been proven to be adequate in her application. Similarly, a design procedure could be entirely empirical, where a designer starts by tackling a particular aspect of the circuit, because that allows her to freeze many parameters and start analyzing the circuit analytically or empirically at some operating point. In general, em-

empirical design knowledge can be seen as any form of design knowledge which is mostly based on experience or observations, and for which no complete physical analytic understanding has been sought, achieved, or deemed necessary.

These two forms of design knowledge are then encapsulated into an executable design knowledge format. Executable design knowledge is, effectively, a general implementation of analytic and/or empirical design knowledge. For instance, a behavioral model of a circuit can be seen as a computerized implementation of a set of analytic and/or empirical equations. In that manner, a circuit simulation can be thought of as executable knowledge, since it is a computerized implementation of some form of analytic and empirical model of a circuit. In this context, executable design knowledge can be of great help to the designer, as it provides feedback regarding the functionality of the circuit.

There are other forms of executable design knowledge. For instance, a design procedure performed by a circuit designer to synthesize, analyze and verify the functionality of a given circuit can also be seen as executable design knowledge. In such a case, the designer executes the design procedure, by making decisions, performing analytic and empirical analyses, and using computerized executable design knowledge to provide guidance into the various phases of circuit development and optimization.

Figure 2 Hierarchy of Design Knowledge

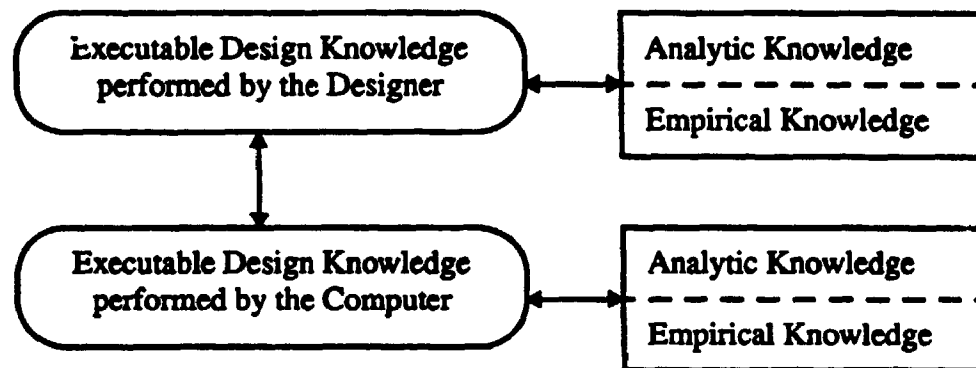


Figure 2 shows the relationship between the executable design knowledge and the analytic and empirical design knowledge. From this figure, we see that a subset of the total analytic and empirical design knowledge is resolved by the computer, whereas the remaining portion is performed by the circuit designer. Circuit simulators, such as SPICE or other similar tools, are computer programs which execute design knowledge already encapsulated in a form by which the computer can relate and execute. Through the reasonably accurate transistor models embedded in the simulator, the computer can analyze the circuit, and convey useful information to the circuit designer. The designer can then take such information, analyze it further, and possibly re-synthesize parts of the circuit. At this point, the designer is effectively accessing her own database of design knowledge, which can be in the form of design notes, theories/theorems, observations, relationships between circuit building blocks, etc. In short, the designer is executing that portion of the executable design knowledge which has not been, or cannot be executed by the computer. Note that po-

tentially a large portion of the design knowledge executed by the designer can be tedious, repetitive, and time-consuming. This observation can help emphasize the motivation of an analog design automation tool. Its main goal is to allow a larger portion of the executable design knowledge to be performed by the computer, thus relieving the designer from wasting time and effort on repetitive, tedious and low value-added tasks.

2.3 Overview of the literature

Since the concept of analog design automation and reuse is not a new idea, it seems ironic that there has been little effort spent to try to address the issue in both academia and industry. In general, those who have tried to address the issue have done so by taking a software-oriented, expert systems approach [6][7][8][9][16][17]. Although useful for certain special cases, these approaches are not flexible enough to be introduced as an industrial-quality framework for analog circuit design synthesis and reuse purposes. They all lack the ability to teach the most important design issues to the designer. Consequently, responsible industrial designers will still tend to seek a more complete understanding of the underlying circuit, in order to have the ability to foresee whether the resulting circuit can perform reasonably well within the targeted application, under normal and abnormal operating conditions.

In general, the expert systems presented in the literature rely upon a combination of numeric optimization and analytic or heuristic design equations. The analytic and heuristic design equation approach can generally be regarded as an improved spreadsheet [9], which, if fed with the right information, can yield reasonably good results. Its main draw-

back is that it requires a substantial pool of effort to develop and capture such design knowledge *a priori*.

This approach assumes that an expert circuit designer is generally available to develop the set of analytic (or empirical) design equations necessary to design and optimize a circuit topology for any potential application. This can be a difficult and time-consuming task, simply because different applications of the same circuit structure may not all be analyzed with the same set of design specifications. For instance, the basic two-stage operational transconductance amplifier (OTA) structure may be optimized for a given bandwidth in one application, but may be optimized for input common-mode and power-supply noise rejection in a noise-sensitive application. The relevant design issues, the environmental effects, and the approach taken to create the circuit layout will be totally different in each case. Thus, for an expert designer to foresee all these potential applications, as well as to develop a good, reliable, and complete set of design equations for all potential applications can quickly grow to become an enormous task, for which there may not be any immediate payback for the corporation. It is important to realize that the financial payback of such design efforts rely strongly upon the assumption that enough designers will reuse such design knowledge efficiently in order to offset the resource costs required for the initial design knowledge development. Another key issue is that most corporations have a very limited pool of such expert designers, who are typically overworked and involved in strategic corporate projects. In practice, it therefore becomes generally unreasonable to pull them away from such projects for long periods of time, for the only purpose of developing the extensive circuit design knowledge bank required by each building block under consideration.

A priori design knowledge development also has the drawback that the intimate design details are hidden from the user of the design automation tool. It could be argued that the user could simply ask specific questions to the expert designer who had developed the initial set of design equations. In practice, however, this is difficult to do because the expert designer may be unavailable or overworked. In such a case, the user of the tool can only hope that the tool will contain the design information relevant to his application. If the tool can be entirely trusted upon, because the initial design efforts performed by the expert designer were complete and thorough, then perhaps the user can trust the results obtained with the design tool. But in general, this would not be the case and consequently, the confidence in the circuit used in a different application needs to be re-built from the user's point of view. Hence, there is a clear need for the user to establish a good understanding of the relevance of each design issue of the circuit block, as viewed in terms of the overall system's performance.

In contrast with *a priori* design knowledge development, the optimizer-based approach takes an unsized schematic, and tries to find a solution which would meet all the stated specifications, usually in terms of mathematical constraints. There is an obvious lack of flexibility in such a case, since many industrial systems specifications are difficult to state in mathematical terms. For instance, the effects of temperature gradients, digital noise coupling and crosstalk, parametric yields, or process and device mismatches are difficult to quantify fully in practice. When manual optimization is done, however, the designer generally gives himself a wide margin so that these issues are not so critical to the design. This is normally considered to be a good practice since there is a general lack of practical process characteristics information relating the effects of these important parameters.

Business cost reasons and time are most often given to explain this incomplete compilation of useful process characteristics information.

Even a complex and stable numerical optimization approach can not be considered as perfectly practical in an industrial sense, especially if there are no efforts spent on developing a good understanding of the circuit. Although it remains possible that an optimizer could reach a good practical solution, a responsible circuit designer in industry would still be required to understand the circuit in order to build confidence in the robustness of the solution reached. As a consequence, optimizers may be tuned to obtain a good and well optimized solution within hours, but it may still take days or even weeks for the designer to completely understand the consequences of the solution found.

This is not to say that optimizers should never be used. Once the designer understands a circuit thoroughly, an optimizer can still help him to obtain a solution which may be more optimal than what has been found manually. However, care must be taken using this approach, as optimizers tend to find an optimal solution with a given set of constraints, which does not necessarily include all real and relevant constraints important to the practical implementation. The designer must be careful to state all relevant constraints as a set of mathematical trade offs or else the optimizer will find a solution that is optimal from a mathematical constraints point of view, but isn't necessarily the most robust from an industrial usage standpoint. Again, this shows that a good and comprehensive understanding of the circuit, as well as the conditions in which it will operate must be considered as very fundamental within the analog circuit design loop.

In all cases, the design systems proposed in the literature tend to hide the design knowledge from the designer. From an industrial point of view, this is most likely the wrong approach. It is true that in many cases, designers may not need to be exposed to all design details. In practice, however, designers still seek to understand the relationship between all relevant design parameters. This will provide them with the confidence they require in order to positively state that their design meets all quality and robustness requirements as imposed by the targeted application. They need to be able to stand in front of their peers and their management, and confidently show that they fully understand the circuit in its targeted application, and hence are in a capacity to guarantee its real-world performance. The need is therefore not a theoretical one, but a very pragmatic one, based upon a designer's confidence in her own design. The fact is that the industrial designer is always ultimately held responsible for any problem her circuit may suffer once on the production line. Consequently she will prefer to take more time up front to gain a reasonable understanding of the important design aspects, in order to be able to foresee any potential problem which may arise when the circuit operates within a complete system at the customer's site. Unfortunately, an automatic synthesis tool can take away this confidence from the designer's perspective.

The solution proposed in this thesis addresses the analog circuit design automation challenge from a different point of view. The philosophy is to provide a method which will help the designer understand all of the most important design trade-offs, in the most practical and in the fastest way possible. The emphasis is on reducing the amount of tedious work performed by the designer, as well as on providing some means to filter out the irrelevant details during the complex process of design space exploration. The research work

was confined to DC and AC small signal analysis, since it is an area particularly well-suited to symbolic analysis.

2.4 Overview of the literature relating to symbolic analysis

Although the theory behind efficient symbolic analysis algorithms form a secondary subject to this thesis, it has been thought appropriate to provide a short overview of the literature dealing with the subject. The concept of symbolic analysis has been around since the 1960's, with much of the research efforts to develop efficient algorithms done in the 1970's and 1980's. Reference [25] provides a good overview of some of the latest algorithms which can be used for solving symbolic systems of equations and for computing the sensitivities of the poles and zeros. The method used in this thesis uses Cramer's rule to solve the symbolic system of equations, similar to what is suggested in the reference. However, poles and zeros sensitivities are computed through partial differentials and perturbations of the physical system, which is a direct but more CPU intensive method than the method described in the reference. The method described in the reference is more difficult to program, but can provide all required sensitivities with comparably less computing. The method is based on solving a modified system, and then solving for the sensitivities based upon the adjoint system. The perturbation method was chosen by the author in this case because it was easy to program, and was nevertheless fast enough to solve precisely the operational amplifier described in the case study. Implementing the adjoint method in Maple is therefore left as a future improvement to the tool.

Finally, the method used to reduce the symbolic expression in Maple is straightforward but efficient enough for the purpose of proving the methodology. However, it is noted that

more efficient algorithms do exist in the literature, but naturally involve much more programming than the method used with the current Maple routines. For instance, reference [14] describes an algorithm which approximates the symbolic expression while it is being generated. In this reference, the author claims that transfer functions of large integrated circuits can be solved efficiently using this method. However, the current work was limited in this scope in order to stay focused on the goal of developing and proving a better methodology for analog circuit design. Implementing the more advanced expression reducing techniques is thus recommended as future work.

3.0 The Proposed Method

The issues relevant to the development of a useful analog circuit design tool need to be examined in order to formulate a set of required features. Thus, the main goals of the tool are stated in section 3.1, followed by a discussion of the generally accepted methods to analyze electronic circuits in section 3.2, with a special focus on linear small signal analysis. The power of symbolic analysis is exposed in this section as a way to efficiently explore the design space in the linearized small signal domain. MOS transistors modelling issues are also outlined as this ultimately limit the accuracy of the small signal analysis. The Maple-Eldo prototype, and its design flow are described in section 3.3, while section 3.4 provides a reference guide on the usage and general algorithms for the routines and scripts within the tool. Final optimization and design centering is exposed in section 3.5.

3.1 Goals of the analog design automation tool

Most solutions presented in the literature hide the circuit design knowledge from the designer. It was argued earlier that this is not the best approach since in most practical situations, the responsible designer would much rather understand the circuit completely before committing it to volume production. A better analog design automation tool must therefore provide the infrastructure for designers to develop and capture their circuit design knowledge efficiently and quickly since the goal is to understand the circuit as quickly and completely as possible. Moreover, the problem of circuit design knowledge reuse can also be addressed since the knowledge captured can be used by a different designer at a later time to quickly master all the important trade offs and design issues previously considered by the initial designer. In practice, each design problem tends to have its own set of

important design issues, and thus it becomes almost impossible to come up with a single generalized approach. An oversampling analog-to-digital converter, for instance, consists of a totally different set of design parameters than a low noise amplifier. Even when considering a specialized family of analog circuits, such as a type of amplifier structures, their design parameters will still differ depending upon their targeted application, process technology, and operating conditions. As mentioned earlier, a given type of amplifier structure could be optimized for different applications, and will therefore have a different set of constraints and design issues for each case, even though the basic structure of the circuit remains the same. In short, trying to provide a framework into which general circuit design knowledge has been captured might not be the best approach. As each circuit design issue tends to be specialized in light of its targeted application, it is such specialized circuit design knowledge that is most relevant to the designer, and not so much the more general circuit design knowledge.

A better solution should therefore focus on an improved method for generating such specialized circuit design knowledge. Since the designer will be called upon to use the tool often, it is also important to focus on the generally accepted design practices and methods currently developed and used within the analog circuit design community. Another important goal is to redirect a larger proportion of the executable design knowledge performed by the designer to a more computerized format. A useful design automation tool should meet these fundamental requirements, while providing a very flexible and user friendly working environment.

3.2 The conceptual analog design automation tool

3.2.1 Generally accepted design practices and methods

In the circuit analysis world, there are two general methods helping the designer to analyze and understand an analog circuit: Linear and non-linear circuit analysis. Linear circuit analysis makes the assumption that a circuit can be linearized about a DC operating point. This assumption is generally valid if the designer wants to observe the effects of the circuit on a small signal (typically less than 10mV). Once linearized, a circuit can be analyzed in the frequency and time domains. Although simple in theory, linear circuit analysis is a very powerful method that has been used extensively within the academic and industrial design community.

Non-linear circuit analysis, although more accurate in theory, tends to be generally less useful in practice. This is mostly due to the enormous complexity presented by a thorough non-linear analytic solution. For this reason, among many others, designers tend to rely on their own common sense and experience when designing a non-linear section of a circuit.

For both linear and non-linear circuits, designers rely upon their favorite circuit simulator to verify the functionality of their designs. Although there is no doubt that a high quality simulator is very useful if not essential, it must be noted that good common sense always play a very fundamental role in the analog circuit design process. A circuit should always be understood qualitatively before it is simulated. This gives the designer the ability to make a judgment regarding the meaning and usefulness of the simulator output.

A useful analog design automation tool should therefore capitalize on these well developed methods for circuit analysis. Since simulators provide a good numeric answer to both linear and non-linear circuits, they serve the obvious and critical need of circuit verification. However, they generally provide little insight on how and why the circuit works, since they simply tell the designer how the circuit performs given a specific stimulus. Since the analog design automation tool seeks to help the designer to understand the circuit, stand-alone simulators provide little help toward achieving this goal. In order to develop an automated approach to develop this design knowledge, a study of the typical design flow normally performed by hand by the designer is first required.

3.2.2 Important design activities

In general, a design team is interested in building a working system that is globally optimized, as opposed to a basket of unrelated and locally-optimized circuit building blocks. Thus, one important aspect is to spend the time necessary to completely understand the systems issues, and hence come up with a reasonable partitioning of the various circuit building blocks, each supported with their own set of design specifications. Once this partitioning is complete, the design efforts shift to an analog cell design strategy. Equipped with a set of functional specifications, the analog circuit designer is required to find a suitable circuit topology which is believed to have the capacity to meet the building block specifications. In most cases, the designer will be aware that certain circuit structures are best for the application considered, and will therefore work out the basic, first-order set of design equations required to obtain a circuit which can be simulated. Once the structure is captured, the designer can start to optimize the circuit, and eventually obtain a sized sche-

matic which meets all design specifications, and which is robust enough to perform as required within the anticipated range of operating conditions.

In many instances, however, the designer may be required to explore unknown structures, for a variety of reasons. Incremental modifications to a basic topology are often considered, in order to address some annoying issues existing in the original structure. At some other time, a penchant for research might be granted because the existing topologies may all have a fundamental flaw. Innovative low voltage circuits, for example, might fall under this category. But in general, the designer needs to work out some basic approximate equations by hand, in order to get a feel for the advantages and the limitations of the topologies considered. This process will help to obtain a grossly sized circuit schematic, which can then be captured and simulated. In short, this provides the initial starting point for any circuit design.

Once a circuit structure has been proposed, the designer needs to conduct a complete analysis of the circuit, using the tools and methods generally available within the design community. This is achieved in order to identify and understand the secondary effects hidden in the circuit structure, aiming at formulating a solution which relates all such effects to the design specifications of the circuit block. Such design specifications must include all robustness issues relevant to the given application and operating conditions within which the circuit must operate. The designer seeks to obtain a clear understanding of all the non-ideal effects, and needs to quantify their impact on the overall circuit performance.

The process of identifying, understanding, and resolving these effects is generally one of the most time consuming design activities in practice, especially for designers with limited

experience. A useful design automation tool should therefore provide a framework to the designer allowing for intelligent and efficient design space exploration, in order to quickly obtain a good grasp of the numerous design issues and their inter-relationships. It is imperative for the designer to understand well how the circuit works, in order to perform the right type of simulations, and to properly determine the bounds at which such simulations can be relied upon. It is clear that in a production environment, a badly understood circuit can lead to devastating effects over the entire project - hence the importance of identifying and understanding non-ideal and secondary effects hidden within the circuit structure.

Several features can be considered as crucial in helping the designer identify and understand such effects. One important aspect is to provide the ability to the designer to navigate within the design space, while observing the effects of various relevant parameters. Since there is an enormous quantity of modelling information underlying a given circuit structure, it is important for the designer to have the ability to easily filter out all the less relevant details, while displaying the useful information in a way that leads to a clear, quick, and concise understanding of the major design issues. This filtering exercise is an important productivity aspect as it must be assumed that the designer is on a very tight schedule, and hence a simple and straightforward answer given to the designer is warranted and welcomed.

Since the design knowledge developed in such a framework is likely to be reused by a different designer at a later time, the framework must also make allowances to help the designer capture the design information in a systematic and useful way. In practice, this must translate into an easy, natural, and error-free method to capture the design information. It

is therefore preferable to have a framework into which the designer's thoughts and thinking processes are captured as they evolve, as opposed to a system where only the final results are captured. This way, the designer is not caught with only cryptic design notes when time runs short; As she spends time understanding various issues, the information is captured and analyzed, and thus aborting the process half way during the design space exploration phase will not lead to a total lack of captured design knowledge. Hence, sudden external scheduling pressures will not automatically lead to a complete lack of useful design knowledge for reuse purposes. Although not necessarily complete nor entirely thorough, the information captured so far would still remain useful for another designer at a later time who wants to take the existing design as a starting point.

3.2.3 Features of the analog design automation tool

It is now possible to summarize the goals and the features of the design automation tool. The goals are twofold: First, it must be possible to quickly and accurately understand the circuit under consideration, as this provides an effective solution to improve the designer's productivity, while minimizing the risks of design errors and overlooked design considerations. Second, the design knowledge developed must be captured in a form that makes it readily reusable and transferable. An effective way to achieve this is to provide a method to keep a 'trace' of all the information processed and developed while the designer is exploring the design space. Since not all information captured is necessarily relevant from a reuse point of view, the designer also needs the ability to edit and modify this information.

Bearing in mind the above stated goals, the key features of the tool can be summarized as follows. In essence, the tool must provide:

- the ability to navigate within the mathematical and physical design space,
- the ability to filter out all irrelevant details,
- a framework to capture the design knowledge as it is developed, during the design space exploration phase,
- a systematic, error-free and physically accurate method,
- an easy to use method while emphasizing maximum flexibility,
- a 'trace' of the designer's thoughts and the analytic results obtained as the design space is explored and better understood,
- a way to edit the information captured, since not all of it is necessarily relevant.
- a proper link to SPICE, as it is the main verification tool to prove the circuit performance.

3.2.4 The power of symbolic and hybrid circuit analysis

The single most important feature of the design knowledge development tool is the ability to explore the design space in a way that can provide insights to the designer regarding which directions to take. A symbolic mathematical expression which describes the circuit behavior can, if properly analyzed, provide that sort of information to the circuit designer. For instance, the parameters of the expression can be swept, and the results can be plotted. Such a plot will help the designer visualize the effects of this particular parameter on the overall circuit. Similarly, the partial derivatives with respect to each symbolic parameter can be computed from the symbolic expression, which can provide to the designer the relative sensitivities of the circuit performance with respect to each of such symbolic parameter. The symbols appearing in the expression must be freely chosen by the designer to be meaningful to the issue at hand. For instance, they could be layout matching parameters, small signal capacitance or transconductance, layout geometry, temperature, or a large signal quantity such as the bias current. The concept of symbolic sensitivities is a powerful

one, as it allows the designer to see how any parameter affects the performance of the circuit in the current configuration, and it also tells the designer how the performance can be changed by modifying each of such parameters on a one-by-one basis.

In contrast, the numerical expression resulting from a simulation often lacks this kind of information to help the designer in his decisions to re-size devices (optimization), or alternatively to help him consider a different local circuit structure allowing for a better performance (circuit synthesis). Although numeric sensitivities can be computed by a simulator, it is generally not the case that the sensitivity of *all* relevant parameters can be computed. Most simulators provide only the sensitivity of a quantity with respect to passive or active device sizes. On the other hand, since the output of a symbolic analysis tool will be a complete symbolic expression, sensitivities with respect to *any or all* relevant parameters is always possible. Clearly, symbolic analysis provides the most flexibility to the designer when considering sensitivities.

One of the problems encountered with the automatic generation of symbolic expressions from a circuit netlist is that the expressions generated quickly grow to become enormous and unmanageable, even when generated by a powerful computer. Moreover, it can easily be argued that not all the information provided by the resulting symbolic expression will be useful to the designer, as many terms are irrelevant second or higher order effects. Hence, one important challenge is to reduce the size of such expressions, preferably before much computations are performed on them, in order to save on memory and CPU time. Recent advances in symbolic analysis for large circuits have been focussing on making the resulting and intermediate expressions more manageable [14][18][22]. Although

related, the topic of efficient symbolic expression generation is considered to be beyond the scope of the present work.

This requirement on symbolic computation leads to the concept of hybrid symbolic analysis. Instead of trying to solve the complete circuit symbolically, the designer can tell the computer that most elements are to be solved numerically, but only those parameters of interest to him are to be solved symbolically. The name "hybrid symbolic analysis" comes from the fact that the analysis is performed with partially symbolic and partially numeric quantities.

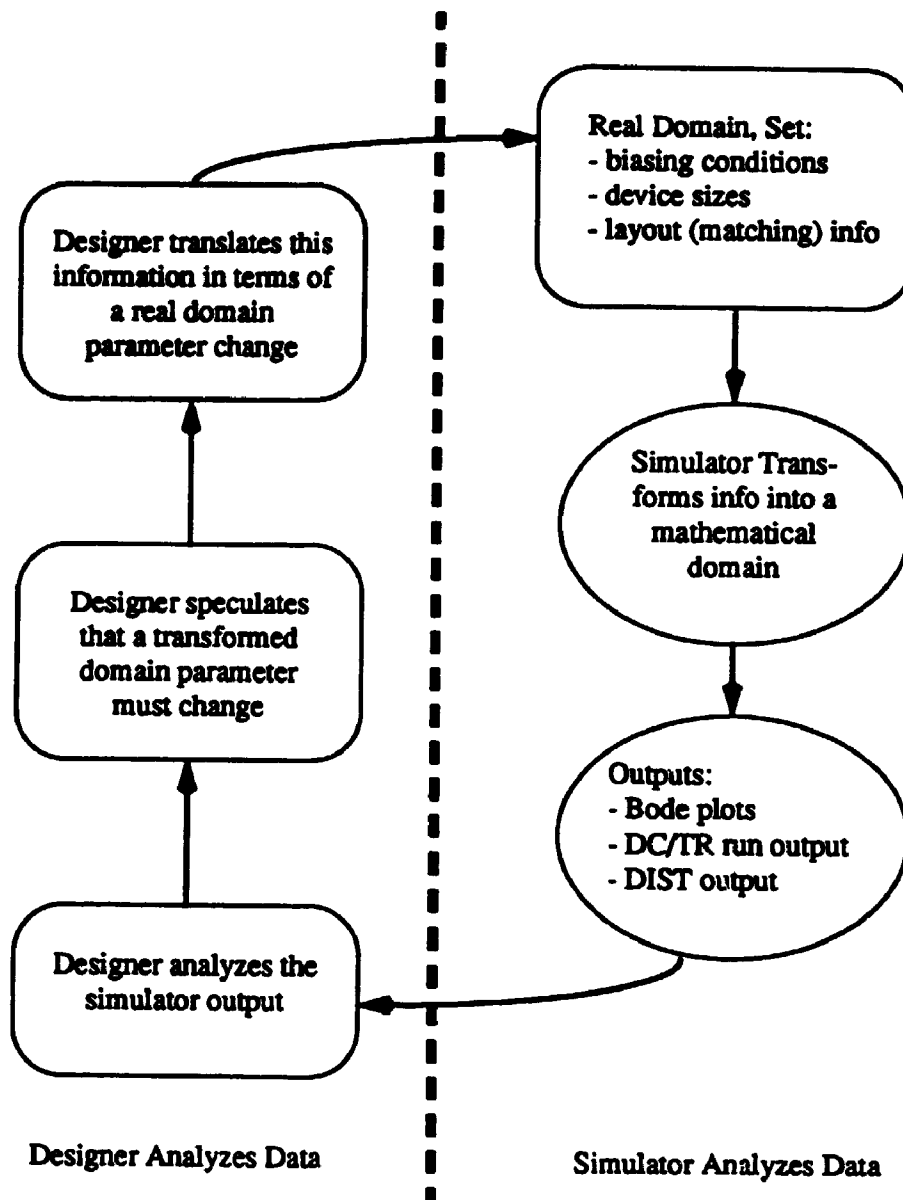
An important element which requires consideration in hybrid symbolic analysis is the process of choosing the elements that are to be solved numerically, and those that should be retained as symbolic quantities. As a simple approach, the designer can seek to retain as symbolic only those quantities which she considers relevant to the immediate problem. This implies that, for instance, circuit parasitics and other second order effects might not be required to be retained symbolically, unless an analysis on parasitics is the immediate goal of the designer. However, the designer typically does not care to change such quantities during the design space exploration and circuit synthesis phase, and hence, it is generally acceptable to solve these numerically. Moreover, several first order parameters can be chosen to be numerically fixed for a given type of circuit analysis, as the designer would not normally care to consider their impact during the current analysis problem. It then follows that such quantities might be chosen to stay numeric as well. In general, the designer might want to consider only a dozen or so of well-chosen design parameters which she considers as relevant for the current analysis. These parameters will thus be chosen to be

symbolic, while every other quantity will be retained as numeric quantities. This has the effect of generating hybrid expressions of a manageable size, which can then be reduced, simplified or plotted using a reasonable amount of CPU and memory resources.

3.2.5 Efficient design space exploration in the transformed domain

Electrical engineers have developed several analysis tools in order to understand and visualize the performance of an electronic circuit. These tools are typically some form of mathematical model which greatly simplifies the underlying physical phenomenon, while emphasizing the relevant performance parameters. For instance, the whole realm of linear, small signal analysis, is based upon this concept of mathematical modelling. Fourier and Laplace analysis, for example, are linear mathematical tools which attempt to model a certain subset of the physical behavior of a real circuit. In essence, the designer seeks to map the real, physical parameters of a circuit onto the linearized small signal mathematical model. He then manipulates and analyzes the circuit information in that domain, and eventually reaches a conclusion. The conclusion could be a piece of information leading towards a more optimized circuit performance, or it could be information explaining why the given circuit cannot be optimized to meet the performance requirements. In either case, the designer is eventually required to transform the meaning of his conclusions reached in the linear domain into a real, physical domain conclusion. He will normally perform this transformation by hand, and interpret it in the form of a modified simulator netlist or set of simulations parameters. The resulting modified circuit and/or circuit test-bench is then verified through accurate circuit simulations. Figure 3 describes this process as an iterative loop in a flowchart format.

Figure 3 The iterative design flow using a simulator as a verification tool

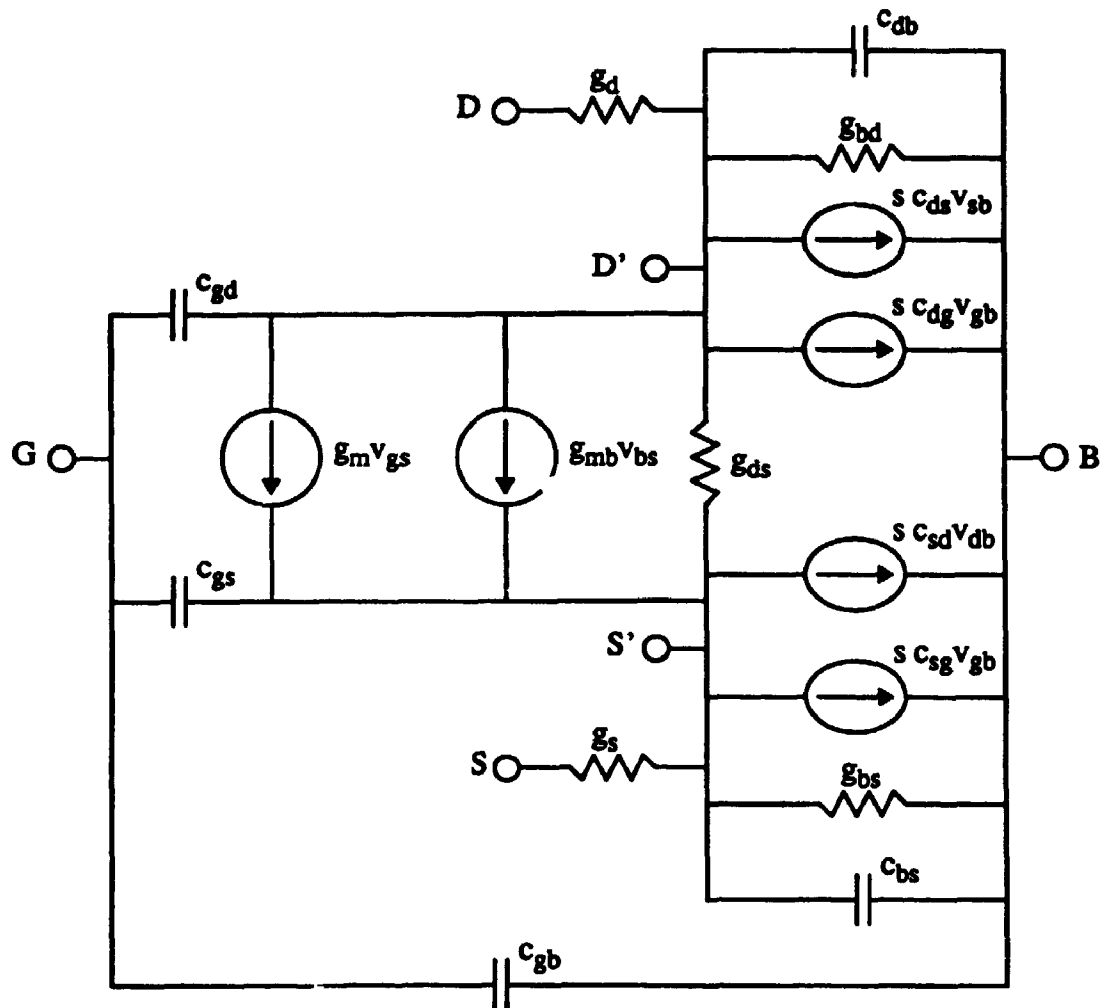


The real, physical domain includes all physical and environmental parameters playing a role in the circuit performance. Although not exhaustive, the following is a list of such important parameters:

- circuit structure,
- device sizes,
- D.C. biasing conditions,
- environmental conditions such as junction operating temperature ranges,
- silicon processing parameter variations,
- physical layout information such as device matching, physical noise coupling resulting from placement and routing, etc.

The transformed domain, such as the linear small signal domain, attempts to map such information onto a mathematical model which permits the designer to perform a simplified, yet reasonably accurate analysis of the circuit under consideration. Figure 4 shows the generalized small signal model of a MOS transistor, as an example of a small signal model for an electronic device. Note that other types of transformed domains also exist, such as large signal non-linear analysis domains. However, the present work shall be considering only the linear small signal analysis domain, due to its general usefulness and its wide acceptance in the electronic engineering and design community. A discussion on the MOS transistor small signal modelling is thus warranted as it provides the groundwork for further discussions on small signal symbolic analysis.

Figure 4 The generalized small signal model of the MOS transistor



Note: All signals and elements are written in lower case as a reminder that these are small signal (linearized) values.

The MOS transistor small signal model shown in Figure 4 is more complete than the generally accepted small signal model described in electronic engineering textbooks [24], because it includes the effects of the mutual capacitances. Unlike normal capacitances, where the capacitive current is related to the derivative of the voltage across the capaci-

tance terminals, the mutual capacitance current is related to the derivative of the voltage across any other two nodes in the surrounding circuitry. The following equation describes the relationship, where $v_{3,4}$ is the voltage across any two terminals in the circuit except the capacitance terminals, and $i_{1,2}$ is the current through the mutual capacitance:

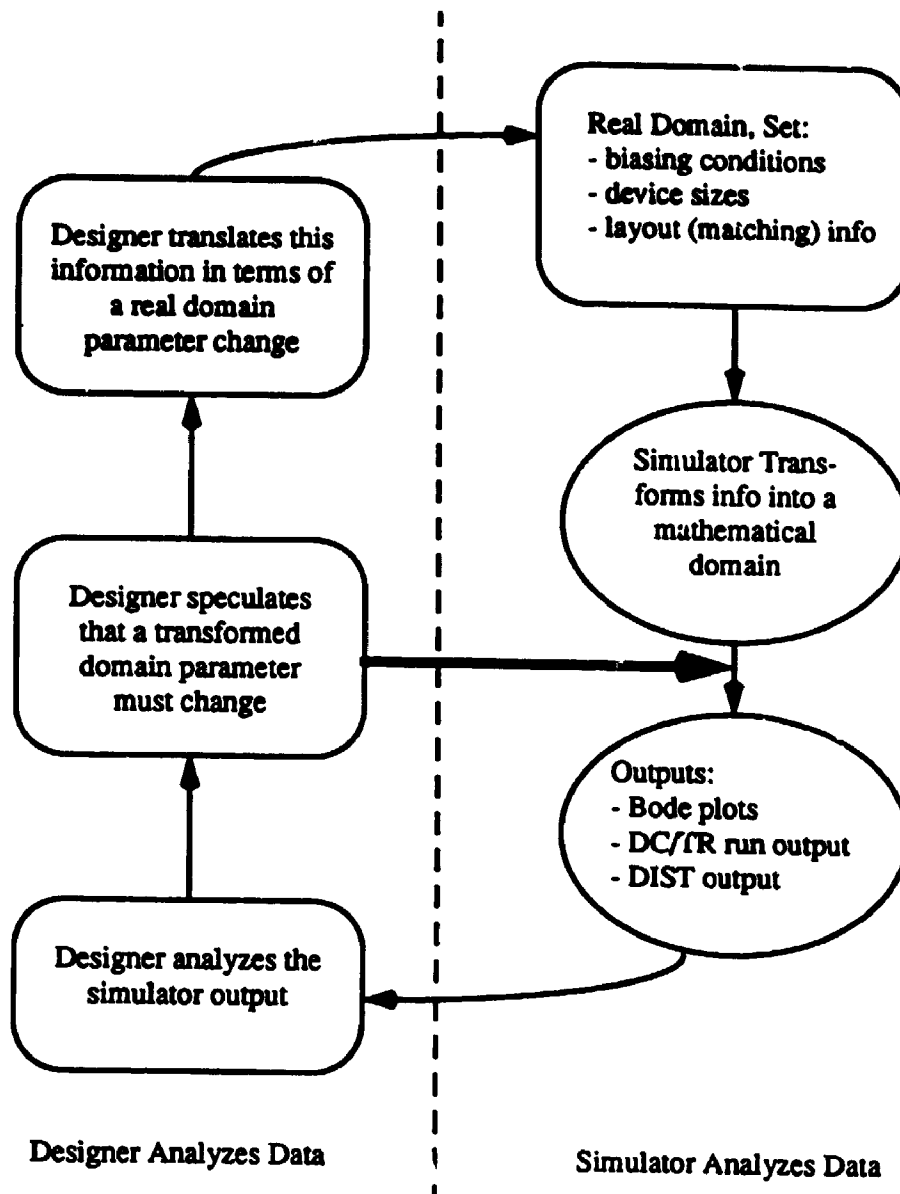
$$i_{1,2} = C \cdot \frac{dv_{3,4}}{dt} \quad (\text{EQ 1})$$

The small signal model shown in Figure 4 includes the four possible mutual capacitances present in a four terminal MOS transistor. The mutual capacitive current flowing between the source and the body is controlled by two inter-nodal voltages namely, v_{gb} and v_{db} . Note that v_{gd} is not necessary because once v_{gb} and v_{db} have been considered, then the v_{gd} component is implicitly included in the system by definition of Kirchoff's voltage law. Similar to the source to body capacitive current, the drain to body mutual capacitive current is controlled by v_{gb} and v_{sb} . Again, the v_{gs} component is not necessary due to Kirchoff's voltage law.

Since the designer performs her analysis in the linear small signal domain, it would be useful for her to have the ability to visualize various performance scenarios within that domain, even though such scenarios might not be physically realizable outside the transformed (small signal) domain. For example, she might want to have the ability to visualize the performance of the circuit under consideration, with all parameters assuming their value corresponding to their physical realization of the circuit, with the exception of the small signal parameter c_{gs} , which would assume a value that is, say, 50% smaller than its physically realizable value. Although the resulting small signal model then becomes a distortion

of the physical reality, the designer can use it to gain a better understanding of the meaning of the c_{gs} parameter, and its effect on the overall circuit performance. This more flexible form of design space exploration, which allows for exploring outside the physically realizable domain, can have the potential to greatly help the designer in understanding the relationship between the small signal model parameters of the circuit, and the resulting circuit performance. The flowchart presented in Figure 3, which illustrates the iterative loop of the design process, can therefore be enhanced with the addition of an important link forming a smaller loop within the small signal analysis domain. This additional link is shown as a boldface arrow in Figure 5. The additional flexibility provided by this feature allows the designer to draw speculative conclusions and analyze his results, by exploring the design space in the small signal domain only, even though such conclusions may not correspond to a physically realizable circuit. Of course, when the designer has obtained a reasonable understanding of the small signal design parameters, and their effects on the circuit performance, then he can draw conclusions which can be translated into a set of corresponding physical design parameters. At this point, the larger traditional loop is pursued, where the circuit netlist and/or circuit test-bench is modified, followed by a simulation run which generates a circuit performance output in the small signal domain. This output is then analyzed by the designer, who can perform further iterations, whether within the small signal iterative loop, or the larger traditional loop encompassing the physical circuit, until an optimized circuit is obtained.

Figure 5 The iterative design flow with flexible design space exploration



Hybrid symbolic analysis can be used efficiently to analyze an electronic circuit in the small signal domain. The physical circuit can be analyzed by the circuit simulator, and the value of the small signal parameters for each device small signal model can be output.

This information can be passed over to the hybrid symbolic analysis tool, which already has the circuit network matrix (that is, the circuit netlist) in its memory. Substitution of those small signal parameters with their numeric values can be performed, since the simulator has generated those values previously. The remaining small signal parameters are retained as symbolic elements, and the circuit matrix is solved. The resulting transfer functions, which are hybrid symbolic/numeric expressions, can be analyzed by the designer. For instance, Bode plots can be generated, which should look identical to those generated by the simulator, assuming all symbolic parameters have been substituted with their values provided by the simulator. However, different values for these small signal parameters can be used, in order to visualize their effects on the overall circuit performance. For instance, parametric plots can be generated, where the parameter is a given small signal symbolic parameter swept around its equivalent physical value. Such plots can provide to the designer a feel for the sensitivity of the overall (or partial) circuit performance with respect to the small signal parameter considered. Parametric Bode plots, root locus plots, and pole/zero sensitivities are described in the next section, where the Eldo-Maple V prototype is described.

3.3 The Maple-Eldo prototype

A prototype of a hybrid symbolic analysis tool has been developed using the Maple V symbolic computational engine and the Eldo circuit simulator. This section provides a descriptive overview of the tool and its principal algorithms, as well as a general overview on how to use the tool efficiently.

3.3.1 Maple V and its features for hybrid symbolic analysis

Maple V is a computer algebra software program used for symbolic manipulation, precise numerical computations, graphics and mathematical programming. It was jointly developed by researchers and students at the University of Waterloo, the Swiss Federal Institute of Technology (ETH) at Zurich, Drexel University, and Waterloo Maple Software, Inc.

Among its most notable features Maple V provides:

- Advanced algorithms for symbolic manipulation, including all standard algebraic operations, calculus and linear algebra.
- A powerful graphical user interface (gui) for displaying two and three dimensional graphics and complex mathematical expressions.
- A large collection of numerical functions, operating on numbers of arbitrary precision and magnitude.
- A powerful programming language for writing user-specific applications.
- A worksheet-based interactive interface, providing maximum flexibility to the designer during interactive design space exploration.

The interactive worksheet front-end provided with the Maple language interpreter constitutes an efficient way for the designer to capture his design knowledge as it is developed. The worksheet features the basic word processing editing facilities, as well as the ability to paste in graphics, and mix in user text with Maple outputs. The worksheet can then be

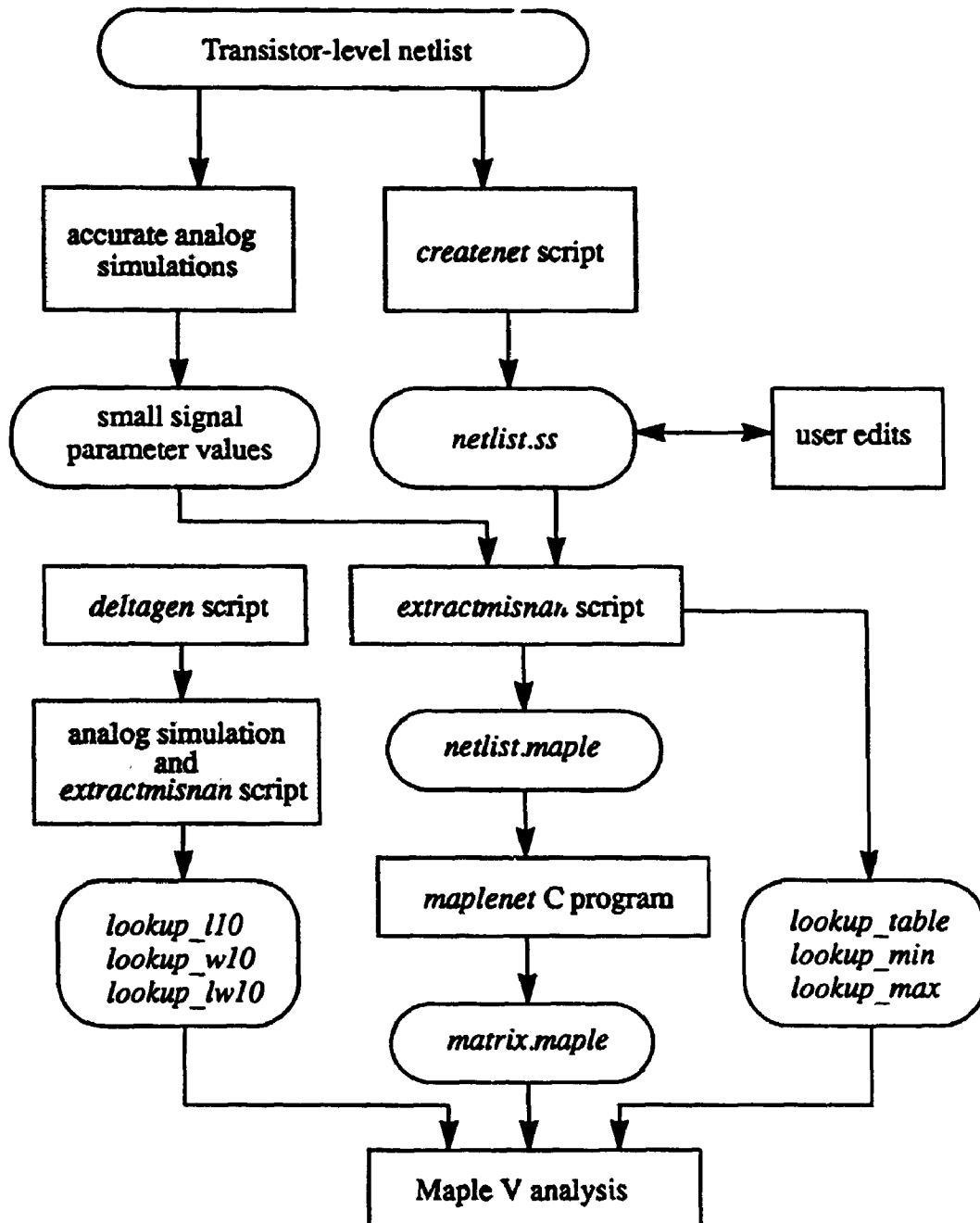
saved on disk or printed, thus providing a document tracing the steps of the designer as he gained a better understanding of the design space of his circuit. Such documentation can then be kept as a database of design knowledge, which can be reused later by a potentially different user. This user can then quickly learn the essential elements and issues relevant to the design space previously explored by the initial designer. Design knowledge reuse and transfer can therefore be achieved in this manner. In essence, the Maple worksheets effectively become a complement to the designer's engineering notebook, where concepts and postulated theories can be executed and verified with accuracy.

3.3.2 The design flow using the Maple-Eldo prototype

The Maple-Eldo prototype allows the designer to perform hybrid symbolic small signal analysis on a given circuit. Although the concept of symbolic analysis can certainly be extended to any form of linear or non-linear analysis, the scope of the tool was limited to linear small signal analysis for its simplicity and its general usefulness amongst electronic circuit designers.

The general design flow using the tool can be divided into two phases: The first phase consists in performing a number of tasks outside Maple, in order to build a proper hybrid symbolic matrix representing the circuit under study. The second phase consists of solving the hybrid symbolic matrix with the Maple engine, and performing a variety of analyses on the results in order to gain a better understanding of the circuit. These two phases will be described separately.

Figure 6 First phase of the design flow using the Maple-Eldo prototype



The task flow during the first phase is illustrated in Figure 6. The data files are illustrated as rounded boxes, while the programs or executable scripts are illustrated as square shaded boxes.

The starting point is a SPICE-like transistor level netlist of the circuit being studied. The ending point of this phase is comprised of seven files, all of which are used by the Maple-Eldo prototype during the second phase of the analysis:

- *lookup_table*
- *lookup_min*
- *lookup_max*
- *lookup_110*
- *lookup_w10*
- *lookup_lw10*
- *matrix.maple*

The aim of the phase one flow is to generate the seven files used as input to Maple. Phase two, on the other hand, will use these files to analyze the circuit symbolically and numerically. The file '*matrix.maple*' contains the small signal circuit network information, in the form of the well-known modified nodal admittance (MNA) matrix, in a format that Maple can read and execute. Since most elements will be solved numerically, the matrix will contain many numbers as representation of their corresponding linear elements. Only those elements which the designer wants to solve symbolically will appear in the matrix as a symbolic value. In general, however, the matrix will have the same appearance as the standard MNA matrix of an electronic circuit. For a more complete discussion on the modified nodal admittance matrix representation of an electronic circuit, please refer to reference [25].

The file '*lookup_table*' provides a table of all linear elements, with their corresponding numeric values under nominal process, voltage, bias current and temperature operating

conditions. This is necessary since once the matrix has been solved in phase two, the designer might wish to substitute the physical values for those parameters which have been solved symbolically, and hence obtain a numeric answer identical to what the simulator would have provided. Moreover, if the designer wants to explore with a scaled version of the physically realizable value of a linear parameter, he can do so by scaling the value appearing in the *'lookup_table'* accordingly. In short, making the *'lookup_table'* available to the designer during the second phase can provide flexibility and efficiency in using the tool. An example of the *'lookup_table'* file is illustrated in Figure 7, where a capacitor *'c{ld}'*, and a MOS transistor *'m8'* are described, using the standard Maple language table format. This format can be directly read and executed by the Maple engine, and hence the table *'lookup_table'* will be assigned the values for the elements described in the file. Note that the choice of small signal elements describing the small signal MOS model depends on the accuracy level of the model used by the SPICE simulator. The elements shown in the figure correspond to a simplified version of Northern Telecom's MISNAN model, but it closely reflects the generalized small signal model shown earlier in Figure 4 on page 42. The exact model used by the Maple-Eldo prototype tool is shown in Figure 8 on page 67 and is described in section 3.3.7 on page 68.

The files *'lookup_min'* and *'lookup_max'* are identical in format to *'lookup_table'* except that the small signal parameter values are the minimum values (in *'lookup_min'*) and the maximum values (in *'lookup_max'*) simulated over all process, voltage, bias current and temperature conditions, as specified in the control file for the *'autopvit'* script, which is not shown in Figure 6. This script is used to simulate the circuit over all PVIT (process, voltage, current and temperature) corners, and is therefore used to characterize the circuit

over such corners. The details on how the script works, and the format of its files, will be discussed later in section 3.3.6 on page 65, and section 3.5.1 on page 87.

Figure 7 Format of the 'lookup_table' file

```

lookup_table := table(
#-----
#   Passive Element: c[ld]
#-----
c[ld]          = 5.000000e-13 ,

#-----
#   Transistor MOS: m8
#-----
gm[8]          = 8.373900e-04 ,
gm[b8]         = 1.855200e-04 ,
g[ds8]         = 3.109100e-06 ,
c[gs8]         = 6.418700e-13 ,
c[gd8]         = 6.107000e-14 ,
c[gb8]         = 8.416500e-15 ,
c[bs8]         = -1.275100e-13 ,
c[bd8]         = 2.041100e-17 ,
mc[sg8]        = 2.301600e-13 ,
mc[sd8]        = -1.279100e-16 ,
mc[dg8]        = -1.761300e-13 ,
mc[ds8]        = 2.523400e-13

):

```

The files 'lookup_l10', 'lookup_w10', and 'lookup_lw10' are also identical in format as 'lookup_table'. They are generated by the 'deltagen' script, and they are used as input by Maple to compute the sensitivities of the symbolic roots with respect to the length and width of the transistors. Thus, 'lookup_l10' is the result of a nominal simulation with all MOS transistor lengths increased by 10%. Similarly, 'lookup_w10' is the result of a nominal simulation with all MOS widths increased by 10%, whereas 'lookup_lw10' is the re-

sult of a simulation with all MOS widths and lengths *simultaneously* increased by 10%. These three files, combined with the symbolic sensitivities of the roots of the symbolic expression with respect to the small signal elements, provide all the necessary information for Maple to compute the sensitivities with respect to the size of the transistors. This is useful to help the designer during the optimization phase of the design. The '*deltagen*' script is described in detail in section 3.4.1 on page 71. Symbolic sensitivities with Maple are described later in section 3.4.4 on page 79.

Referring back to Figure 6, it can be seen that the flow requires usage of the analog circuit simulator (Eldo), as well as multiple scripts and a C program. The C program and the scripts are used to provide the link between Maple and the Eldo circuit simulator. The first step in this phase is to execute the script called '*createner*'. This script is an awk program which parses the Eldo transistor netlist, and replaces each transistor instantiation by a corresponding small signal model instantiation. Thus, each single line transistor instantiation in the netlist becomes a multi-line network of linear elements representing the small signal equivalent circuit for that device. The output is written to a file called '*netlist.ss*', where the '*.ss*' extension stands for small signal netlist. Since the actual physical value for each of the small signal elements is not yet known, the value given by the '*createner*' script is '*ssvalue*' a token word which will later be replaced by the actual small signal value for that particular element, once that value will be known. All other elements (passive elements such as resistors and capacitors) are passed through directly, left untouched.

The output file '*netlist.ss*' is then edited by the designer, who will decide which elements are to be solved symbolically. If no edits are made, all small signal elements will later hold

their physical numeric value, because each is associated the token '*ssvalue*' by default. Thus, the designer should edit '*netlist.ss*', and change the token '*ssvalue*' for the token '*name*' for each of those elements which he wants to later solve symbolically. The token '*name*' means that the actual name of the element, as stipulated in the instantiation inside '*netlist.ss*', will be used as a symbolic value. If a different name is to be used, or if the designer would like to impose a formula, or a hard number, he only needs to change the token for the expression he wishes. This particular feature can be especially useful for mismatch analyses for example, as the name plus a given delta can be stated in place of the token '*name*' or '*ssvalue*'.

The designer may also wish to actively modify the file '*netlist.ss*', in order to simplify certain instantiation of the small signal MOS model. For example, a 10 transistor netlist might contain 3 or 4 transistors which should be solved with a high degree of accuracy, including all second-order parasitics. However, the remaining transistors may not require a high level of modelling accuracy even to maintain a reasonable amount of accuracy in solving the problem at hand. Hence, the designer may choose to simplify the model of these transistor instantiations, in order to reduce the CPU time required by Maple for solving the matrix.

Once the designer is happy with the '*netlist.ss*' file, he then needs to generate the physical values of all small signal parameters, before moving on to further processing. This step is performed using a DC analysis on the original transistor netlist with the analog circuit simulator. The simulator output file is parsed, and each small signal parameter is extracted from the simulation output. This extraction exercise is performed by an awk script called

'extractmisnan'. The name *'misnan'* comes from Northern Telecom's proprietary MISNAN MOS transistor model, which is the transistor model used for all circuit simulations in this thesis. Hence, the *'extractmisnan'* script effectively extracts the MISNAN small signal parameter values for each transistor instantiation in the circuit. The small signal model used is described in section 3.3.7 on page 68.

After the small signal parameters have been extracted from the simulator output file, the *'extractmisnan'* script reads the *'netlist.ss'* file, searches for the tokens *'ssvalue'*, and replaces them with the actual value of the small signal parameter as extracted from the simulation output. The resulting file is thus identical to *'netlist.ss'*, with the exception of having physical values substituted for each instance of *'ssvalue'*. This output file is called *'netlist.maple'*. Moreover, the value for all small signal parameters extracted are written to the file *'lookup_table'*, in a format which is directly readable by Maple, as described earlier. Note that the *'extractmisnan'* script extracts all small signal parameters from all transistors, and builds the lookup table which Maple can refer to later as a lookup. Thus, the small signal netlist *'netlist.ss'* can be modified at will by the designer, and elements can be added or deleted, as long as there are no new parameters introduced which are not somehow expressed in terms of the parameters included in the lookup table. If a new parameter were to be introduced in *'netlist.ss'*, then Maple would not be able to find a numeric value for this newly introduced parameter when it attempts later to calculate the sensitivities or to draw Bode or root locus plots.

The file *'netlist.maple'* is used as input to the C program *'maplenet'*. This program transforms a hybrid (numeric/symbolic) small signal netlist into a modified nodal admittance

matrix (MNA) form, which can be readily solved by Maple. The matrix output file is called '*matrix.maple*', and it is the file which Maple can read and solve in the second phase of the design flow.

3.3.3 Hybrid symbolic analysis in Maple

The second phase of the design flow using the Maple-Eldo prototype consists of solving the hybrid symbolic matrix using the Maple V engine, and analyzing the results in order to gain insight into how the circuit works. A set of Maple routines has been developed to assist the designer in her analyses with Maple. The routines have been encapsulated in the form of a standard Maple package, named the '*smallsignal*' package. A Maple package is a collection of user-defined routines written in the Maple programming language.

Once Maple is up and running, the user first loads the '*smallsignal*' package into Maple's memory. The '*lookup_table*' file previously generated in phase one is automatically read by Maple once the user loads '*matrix.maple*' into Maple's memory. The other lookup files, that is, '*lookup_min*', '*lookup_max*', '*lookup_l10*', '*lookup_w10*' and '*lookup_lw10*' need to be loaded separately as needed.

Once these files are in memory, the designer can proceed to solve the MNA matrix representation of the circuit. This is achieved by invoking the '*mnasolve*' routine of the '*smallsignal*' package. This routine solves the circuit matrix without approximations using Cramer's rule. The evaluation of the matrix determinant uses a sparse technique, as this has been found to be both memory and CPU efficient since most realizable and useful analog circuits have a corresponding matrix which is reasonably sparse in general.

Solving the matrix yields an exact hybrid symbolic solution of the circuit. The solution is hybrid because it consists of a mixture of numeric and symbolic values. This implies that the solution is physically valid only at the DC operating point at which the circuit was previously analyzed with the circuit simulator. Nevertheless, the fact is that we now have an expression with symbolic elements which can be substituted with numeric values other than what was obtained with the circuit simulator, which allows design space exploration outside the realm of what is only physically realizable. If used and analyzed properly, this hybrid expression can provide additional insights about the way the circuit performs, and how it may be sensitive to environmental or processing variations. This can be a great help during the circuit optimization phase, for example.

3.3.4 Small signal analysis in Maple

One of the first exercise the designer should perform is a sanity check of the resulting hybrid expression against the results obtained with the circuit simulator. This will tell the designer whether she made any gross approximations when she was building the small signal netlist, '*netlist.ss*'. For instance, the designer could have removed an important capacitance from the small signal netlist, in the hope of simplifying the resulting expression without affecting the validity of the results. A sanity check would point out the differences between the Maple and the simulator solutions.

The Maple solution can be compared against the results of the simulator by graphical means. Generating a magnitude and phase Bode plot over the frequency of interest, for example, is a simple yet efficient way to achieve this, because it can be readily compared to the Bode plots generated by the circuit simulator. By invoking the '*bodeplot*' routine from

the Maple *'smallsignal'* package, the designer tells Maple to perform two things: First, to use the procedure *'lookup_table'* to substitute the equivalent physical numeric values for all symbolic parameters. Second, it tells Maple to generate the magnitude and phase Bode plots using the resulting numeric expression. The graphical results can be compared with the simulator's output in the region of interest.

Once the designer has confirmed the validity of the exact hybrid symbolic expression, she can then proceed to a more in-depth analysis. The following step-by-step procedure is a summary of all the analyses possible using the *'smallsignal'* Maple package:

1. The designer first solves the MNA matrix using the *'mnsolve'* routine.
2. A Bode plot is generated using the *'bodeplot'* routine. The result is compared with the simulator's output, as a sanity check of the validity of the exact hybrid symbolic expression.
3. The exact expression is reduced to a lower order, and the smaller minterms are substituted with their numeric equivalent. This is achieved by invoking the *'reduce_order'* routine.
4. The designer can compute the sensitivities of the DC gain and each root with respect to the most important contributors amongst the symbolic elements. This is done using the *'print_sensitivities'* routine.
5. The designer can study the resulting sensitivities for each relevant root. The idea is to understand how each symbolic parameter incrementally affects the root placements. The designer thus identifies which symbolic parameter value should be modified in order to improve the root placement.
6. A sensitivity of the roots with respect to the actual size of the transistors is also possible using the *'deltagen'* routine, and by loading the lookup tables *'lookup_l10'*, *'lookup_w10'*, and *'lookup_lw10'* into Maple's memory.

7. The design rules developed in the sensitivity analyses in step 5 and 6 are then verified graphically using two different graphical routines. The first one is the three dimensional parametric Bode plot (the '*bodeplot3d*' routine), where the parameter of the plot is a sweep of the symbolic parameter considered. The second is a root locus plot (the '*rootlocus*' routine), where the root locus parameter is the symbolic parameter under consideration. Note that the sensitivity analysis is performed on the reduced expression, for CPU speed reasons. The plots, however, should be performed on the exact symbolic expression in order to maintain maximum accuracy. This is important as the plots are used mainly as a sanity check, confirming that the root placement is affected as expected from the result of the sensitivity analysis.

Generation of Design Equations

The '*reduce_order*' routine can be used for different purposes, including the generation of relatively simple design equations. The usefulness of '*reduce_order*' lies in the imagination of the designer and the application. The routine's algorithm is simple and efficient. In order to illustrate the algorithm, it is important to realize that in general, a small signal transfer function is of the following form:

$$H(s) = \frac{(a_{00} + a_{01} + \dots) + (a_{10} + a_{11} + \dots)s + (a_{20} + a_{21} + \dots)s^2 + \dots}{(b_{00} + b_{01} + \dots) + (b_{10} + b_{11} + \dots)s + (b_{20} + b_{21} + \dots)s^2 + \dots} \quad (\text{EQ 2})$$

Both the numerator and denominator are composed of terms in 's', whereas each 's' term is composed of a sum of minterms, that is, the 'a' and 'b' terms. The algorithm works as follows: For each 's' term, the routine examines each minterm separately and substitutes its numerical equivalent if the effect on the overall coefficient is less than a given tolerance, '*mintol*', the minterm tolerance. It has been observed experimentally that the 's' coefficients of the expression of even relatively small circuits can include several thousands

of minterms, most of them having a very small effect on the overall value of the coefficient. This algorithm has been developed while keeping this observation in mind. By replacing each minterm by its equivalent numeric value, the end expression still retains all of the accuracy of the original expression when the physical values for all symbolic expression is substituted. However, the complexity of the expression has been reduced by several orders of magnitude, and it therefore becomes much more computationally efficient.

Moreover, the *'reduce_order'* routine can also eliminate certain 's' terms if they are judged to be irrelevant to the overall transfer function over the frequency range of interest. Thus, an intermediate transfer function is generated, where the 's' term under consideration is eliminated. By comparing the magnitude and the phase of the resulting transfer function with the original transfer function at a number of pre-defined frequency points, the algorithm decides whether or not this coefficient is required. The result of the comparison is performed against a parameter *'ctol'*, the coefficient tolerance. The *'reduce_order'* routine also prints an overall accuracy report for the resulting transfer function (with both minterm reduction and coefficient 's' term reduction, if any) as compared with the exact transfer function, over the frequency range of interest. The user therefore knows how accurate the resulting expression is. An example of the output produced by the *'reduce_order'* routine is given in section 3.4, where a user guide for all routines in the *'smallsignal'* package is included.

The above algorithm can be used for other applications than the reduction of circuit transfer functions. In modelling distributed transmission lines, for example, this approach could prove to be particularly useful. The designer can solve the complete distributed

transmission line symbolic model, and then reduce the minterms using the 'reduce_order' routine. The resulting expression would have numerical substitution for all less relevant minterms, while keeping the most important minterms symbolic. In this way, the designer can develop a much more thorough understanding of the transmission line model because he then knows which parameter affects the model the most, and how it does so. The resulting simplified relationship can then be used as general design equations, for example.

Note that better but more complex algorithms exist in the literature. The above algorithm was chosen because of its simplicity and because it was sufficient for our purposes. As mentioned earlier in section 2.4, reference [25] provides a good starting point to the interested reader for exploring better algorithms. Improving the algorithm is, however, a subject for future work.

3.3.5 Effects of large signal parameters

Although the hybrid symbolic expression obtained in Maple contains small signal symbolic parameters, it does not necessarily mean that the only analyses possible are in the small signal domain. In Maple, symbols are equivalent to expressions. Hence, it is possible for the designer to substitute a small signal parameter with an expression of some kind, possibly relating the small signal parameter to a large signal parameter, or to a physical characteristic of the device such as a process mismatch expression. As a simple example, consider the transconductance (gm) of a MOSFET, and its substitution by its well-known large signal expression as described as follows:

$$gm = \mu_n C_{ox} \left(\frac{W}{L} \right) (v_{GS}^0 - V_T) \quad (\text{EQ 3})$$

$$gm = \sqrt{(2\mu_n C_{ox}) \frac{W}{L} i_D^0} \quad (\text{EQ 4})$$

where the superscript '0' implies that the voltage or current under consideration is taken at the onset of strong saturation.

In the above example, the designer is able to see the effect on the roots by performing a parametric bode plot or a root locus plot, but using a large signal or a physical parameter as the sweep variable, instead of '*gm*'.

Similarly, the designer can obtain the sensitivity of the roots to the device physical parameters '*W*' and '*L*', as well as to the large signal biasing conditions (*i_D*) or the threshold voltage '*V_T*'. This can prove useful in helping a designer to better understand the trade-off inherent in a particular circuit or sub-circuit. Note that the '*deltagen*' routine can also be used to achieve this through the use of the simulator, instead of manipulating analytic expressions directly. In the above example, we relate the transconductance to the device size '*W*' and '*L*' through an equation. With the '*deltagen*' routine, the simulator finds this relationship numerically by simulating the circuit with a modified '*W*' and '*L*'. The results are stored in the file '*lookup_l10*', '*lookup_w10*' and '*lookup_lw10*', and the '*deltagen*' routine uses this information to obtain the sensitivity of the roots with respect to the device sizes.

For instance, consider the case where '*deltagen*' uses the data in '*lookup_l10*'. The idea is to relate the root sensitivities to the transistor lengths, in this case. If instead, '*lookup_w10*' or '*lookup_lw10*' were used, then the resulting sensitivities would be related to the transistor widths or to the simultaneous MOS lengths and widths respectively.

The '*deltagen*' routine calculates the sensitivities using the concept of partial derivatives. This is applicable since in general, a physical parameter such as the length affects the value of all small signal parameters. Consequently, partial derivatives must be used to obtain the total sensitivity. Thus, the algorithm used by '*deltagen*' consists of the following steps:

1. The sensitivity of each small signal element with respect to the MOS size (length for '*lookup_110*') is computed. Consider the value of '*gm*' for the sake of an example. Its absolute sensitivity with respect to the length is done as in (EQ 5) since '*lookup_110*' has MOS lengths 10% larger than '*lookup_table*':

$$\text{sensitivity}|_{gm} = \frac{(gm|_{\text{lookup_110}} - gm|_{\text{lookup_table}})}{10\%} \quad (\text{EQ 5})$$

2. The sensitivity of each root with respect to the sensitivity of each element as described in (EQ 5) is found by multiplying one with the other. This gives the root sensitivity to the MOS length assuming that the MOS length would only affect the element considered ('*gm*' in this example). This corresponds to the partial derivative (partial sensitivity) with respect to the parameter under consideration ('*gm*' in this example).
3. Since the MOS length likely affects many symbolic elements simultaneously, each of their sensitivities need to be added together in order to obtain an aggregate sensitivity for the given root, with respect to the MOS length. This is the total sensitivity or, in mathematical terms, the differential of the root.

In step 2 above, the partial sensitivities are found. In step 3, each of these partial sensitivities are added together to form an aggregate sensitivity for the given root, with respect to the length of the given MOS transistor. This provides an efficient and easy method for obtaining the sensitivities of the roots with respect to a physical quantity. Note that other sensitivities could be obtained in this manner, such as the sensitivity to temperature, bias

current conditions, voltage, or even to process. For simplicity, however, the present work has only considered the sensitivity to MOS transistor widths and lengths.

Along the same lines, non-ideal process characteristics such as process mismatches or variations can be analyzed to see their effects in the small signal domain. This could again be analyzed empirically using '*deltagen*', or analytically by substituting formulas for the transconductance, as an example. In such a way, the root sensitivity of a submicron circuit with respect to mismatches or variations in transistor length '*L*' can be analyzed. This can prove to be a useful exercise in minimum length designs of submicron sizes. As semiconductor processes move forward into deep submicron feature sizes, it is likely that designers will see a much higher level of mismatch between minimum gate length transistors.

Note that computing the derivatives in the manner described above (the perturbation method) is known in the literature to be computationally less efficient and potentially less accurate than other methods such as the adjoint technique. However, coding the adjoint technique in the present tool implies that the entire MOS transistor model (MISNAN) be coded in Maple, in order to have access to the internal small signal parameters, or to have the ability to compute the sensitivity with respect to geometry (length and width), or other quantities such as bias current or temperature. The method used by the prototype tool is simple and very flexible, which is one of its most important aspects. We have only calculated the sensitivities with respect to transistor lengths and widths in this thesis, but it would be easy to extend this to any other relevant design parameter such as bias current, process, layout geometry and matching, or temperature, depending upon what the designer requires. The idea here is to have a basic toolbox that is flexible and easily adaptable to be

useful to designer for many different types of circuit structure or circuit optimization goals.

3.3.6 PVIT variations analysis

Another important feature of the Maple-Eldo prototype is the ability to analyze the design over process, voltage, bias current and temperature (PVIT) variations. It is clear that the roots of any design will move substantially from their nominal location over PVIT variations. Hence, no detailed analysis of a circuit design is complete without at least verifying whether the roots stay within a reasonable window over such variations.

The Maple-Eldo prototype includes an automated simulation script called '*autopvit*'. This script was written using the cshell script language and the awk programming language. Its purpose is to automate the invocation of a number of simulations, each of them with a given set of parameters such as operating voltage, temperature, reference bias current, and process characteristics for NMOS, PMOS, resistors, and capacitors. These seven parameters are treated as independent variables, and simulations at each best and worst corners are performed automatically by the script. Thus, a total of $2^7 = 128$ simulations are invoked. Simulation #0 is also invoked as the nominal simulation, from which the values in '*lookup_table*' are extracted.

The results of each simulation are analyzed by the '*autopvit*' script, in order to extract useful small signal parameters. The '*extractmisnan*' script is invoked by '*autopvit*' in order to generate the '*lookup_table*' file (for simulation #0), and to generate the '*lookup_min*' and the '*lookup_max*' files for all other simulations. The file '*lookup_min*' contains the mini-

imum value observed for each small signal parameter over the 128 simulations. Similarly, the '*lookup_max*' contains the maximum value observed for each small signal parameter over the 128 simulations. One of the outputs of the '*autopvit*' script is therefore the two lookup tables, one containing the lowest bound while the other contains the highest value of each small signal parameter. The advantage of this is that it gives the designer a feel for how much each small signal parameter can vary over all possible simulation corners. The '*autopvit*' script also generates a gain, bandwidth and stability (phase margin and gain margin) report which can help the designer to identify the problem corners. A more detailed description of the '*autopvit*' script is given in section 3.3.6 on page 65.

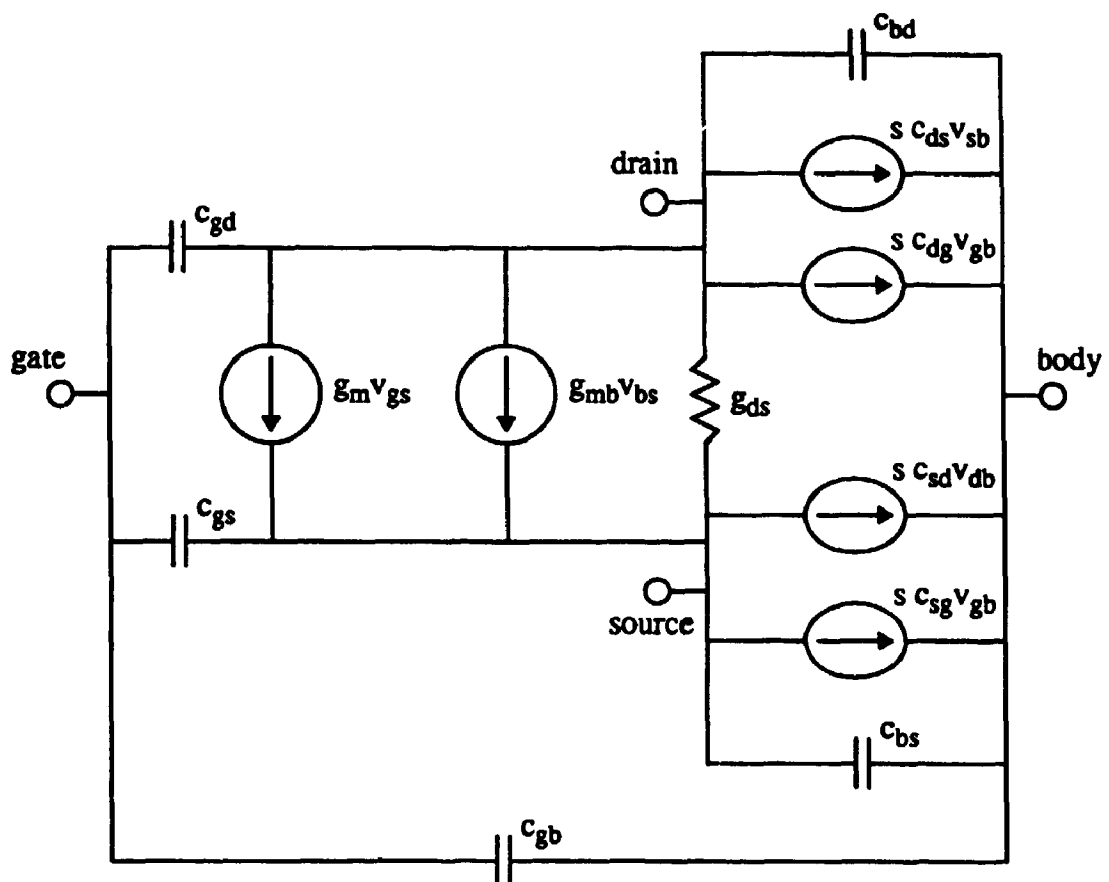
The '*autopvit*' script is an example of an important property of an executable worksheet. The designer can write a script to automate the methodical checks needed for a reliable design. In another application, the designer might want to add other corners (such as the particular sheet resistances) or look for extremes within the N-dimensional space of process and environmental variations.

Note that for this particular script, the resulting small signal parameter variations represent the maximum variation of each parameter *as an aggregate*. This essentially means that all tracking information between parameters has been lost in the process, since '*autopvit*' will put the small values together in '*lookup_min*' and the large values in '*lookup_max*'. These results from '*autopvit*' must therefore be interpreted with care by the designer as they do not represent a real condition by themselves. However, they can be useful in providing the designer with a feel for how much each of the small signal parameter values can vary over the process corners. This information, combined with the knowledge about the parametric

root sensitivities, can help the designer understand the effects of the simulation corners on the location of the roots.

In Maple, the designer can view the best, worst and nominal values for each relevant small signal symbolic parameter using the '*display_indets*' routine from the '*smallsignal*' package. This routine is described in section 3.4.5 on page 86.

Figure 8 The MOS small signal equivalent model used in the Maple-Eldo prototype



Note: All signals and elements are written in lower case as a reminder that these are small signal (linearized) values.

3.3.7 Small signal modelling issues

Any analytical tool is only as accurate as is dictated by its modelling of the physical reality. The analysis performed in Maple thus depends on how well the small signal equivalent model used represents reality. Figure 8 illustrates the linearized model used for small signal analysis with the Maple-Eldo prototype. Although it does not include all of the parameters used by Northern Telecom's MISNAN MOSFET model, it nevertheless produces generally accurate results when benchmarked against the simulator. This can be attributed to the presence of the mutual capacitances with respect to all nodes in the model. It is important to ensure that the model used within Maple is accurate over the frequency range of interest if useful conclusions are to be drawn from the analyses.

3.3.8 Knowledge capture using the Maple-Eldo prototype

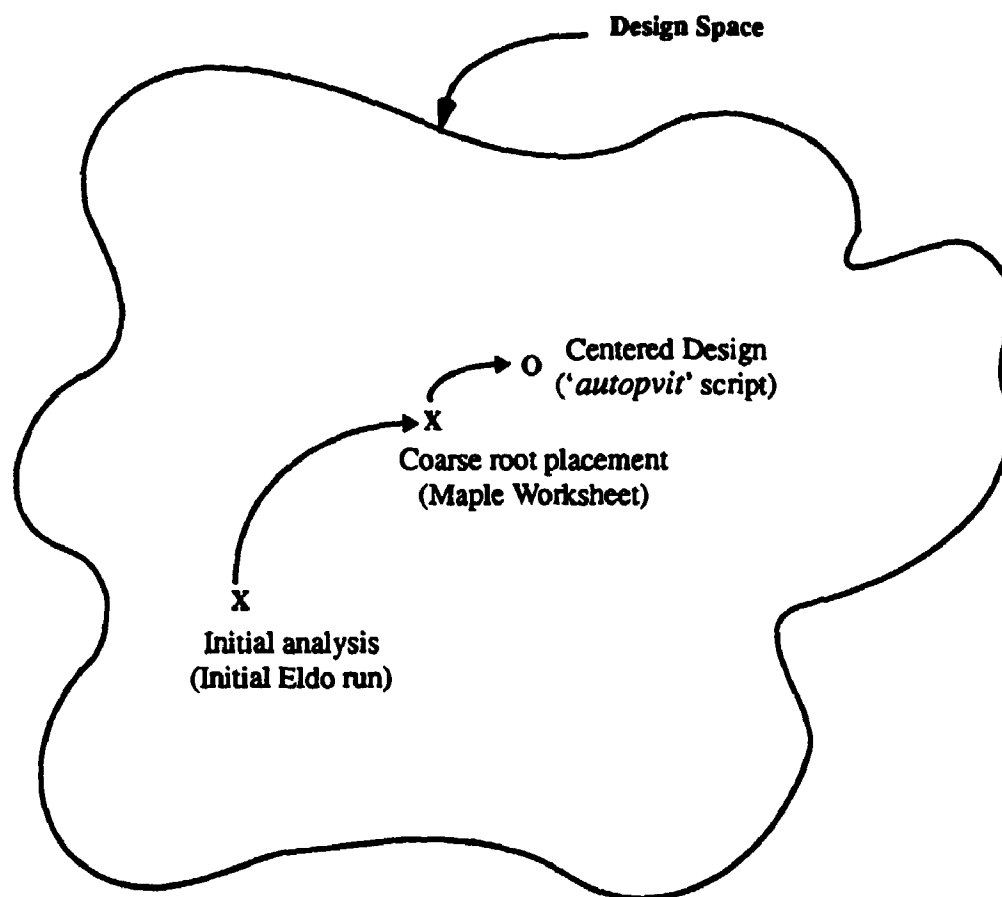
A knowledge capture and reuse system must provide a means to capture the learning path followed by the designer in order to keep a trace of the knowledge developed. So far, all the discussions on the Maple-Eldo prototype have focused on helping the designer to gain a better insight into how the circuit operates, and to understand why it behaves in a particular fashion. A second goal is to provide a logged record of the learning path pursued by the designer in his search towards a better understanding of the circuit. This record is captured in the form of Maple worksheets, where each worksheet corresponds to a set of simulation runs of a given netlist.

A Maple worksheet consists of a sequence of Maple commands, Maple results, user comments, and Maple graphs. Thus, a worksheet can provide a log of the evolution of the designer's analyses, postulates, and conclusions regarding the functionality of the circuit. In

this manner, the design knowledge is captured as it is developed and as it evolves in the designer's mind.

Using the '*smallsignal*' package, the designer can find out where the design currently resides within the design space, and he can determine the direction in which he needs to move the design parameters in order to achieve a more optimal solution. This concept is illustrated in Figure 9. The figure describes the evolution of an amplifier circuit within its design space.

Figure 9 Circuit design/optimization through design space exploration



The starting point in Figure 9 corresponds to the initial SPICE analysis. In this case, the amplifier might initially be marginally stable and definitely does not meet the design specifications. Nevertheless, the designer can take the output of this simulation, and analyze them using the Maple-Eldo tool. The conclusions reached will allow the designer to gain a very good understanding of the trade offs required to optimally place the roots of this circuit. The designer can use these rules to re-size the devices properly, and can thus run a number of simulations and correlate these results with the knowledge developed using Maple. He can thus resize his circuit and verify its functionality using the simulator.

Once reasonable results are obtained from simulations, the designer can then proceed to the design centering phase of the circuit. This is achieved using the '*autopvit*' script, which exercises the design over all required simulation corners. For the purpose of the amplifier example, seven corners have been chosen, each of which with their best and worst value: supply voltage, temperature, reference bias current, NMOS, PMOS, resistor, capacitor. Thus, the script will invoke $2^7 = 128$ simulations in total. Moreover, the script produces a report which gives stability and performance information for each simulation corner. Using the rules established in the Maple worksheet, the designer can then fine tune the placement of the roots until an optimized design has been obtained. The design is, of course, verified over all corners using the '*autopvit*' script. The case study in section 4.0 on page 91 describes in detail the design of an amplifier using the Maple-Eldo tool. The Maple worksheet used to design this amplifier is shown in APPENDIX B on page 130.

3.4 Using the Maple-Eldo prototype

This section describes the many programs, scripts and Maple routines which together form the components of the complete tool. The system comprises two main sections. The first section is a collection of scripts and a C program which produce the seven files for Maple, as illustrated earlier in Figure 6 on page 49. The second section is a collection of Maple routines (the '*smallsignal*' package) which the user can invoke in an attempt to analyze the circuit, using the seven files produced by the C program and the scripts as its input.

Figure 6 on page 49 illustrated the flow of the tasks necessary to generate the files required by the Maple tool. The first script invoked is the '*createnet*' script, which generates the file '*netlist.ss*' from the circuit netlist. The user then edits this file, and invokes the '*extractmisnan*' script which extracts the small signal parameter values from the simulator output file, and writes them to the '*lookup_table*' file. This script also generates the '*netlist.maple*' file. This file is similar in structure to the '*netlist.ss*' except that all the components which are not required to be solved symbolically have a numeric value associated with them. Thus, the '*netlist.maple*' file is hybrid in the sense that it contains both numeric valued components and symbolic components. Finally, the '*maplenet*' program translates the '*netlist.maple*' file into an MNA (Modified Nodal Admittance) matrix format which Maple can read and solve. The following section provides more details about the '*maplenet*' C program, and the '*createnet*', '*extractmisnan*' and the '*deltagen*' scripts.

3.4.1 The UNIX scripts and the Maplenet C program

createnet. This script is used to create a small signal netlist corresponding to the physical netlist of the circuit. The input parameter is the file name of the physical netlist to

be translated. The output is an equivalent netlist that comprises linearized small signal components only. Thus, a MOS transistor instantiation in the physical netlist will be replaced by its equivalent small signal sub-circuit. All linear elements such as resistors and capacitors are left unchanged. The value of all elements, however, are instantiated with the keyword '*ssvalue*'. This keyword is a token which is later replaced by the corresponding small signal value for the linear element. The '*extractmisnan*' script performs this function by parsing the output of an eldo simulation output file, extracting the small signal component values for each device, and replacing the '*ssvalue*' token by the actual value found from simulations.

As an example of the function performed by '*createner*' consider the instantiation of a MOS device in the simulator netlist:

M8 drain gate source body MPCH L=3U W=150U

After processing this particular line through the '*createner*' script, the MOS instantiation will be replaced by its equivalent small signal sub-circuit. Thus, the output file '*netlist.ss*' will look as illustrated in Figure 10.

In this small signal netlist, the '*gm*' elements are transconductances, the '*g*' elements are conductances, the '*c*' elements are capacitances, and the '*mc*' elements are mutual capacitances.

Figure 10 Format of the 'netlist.ss' file for a MOS transistor instantiation

```

*****
*  mos: m8  model: mpch
*****
gm8      gate  source  drain  source  ssvalue
gmb8     body  source  drain  source  ssvalue

gds8     drain source                                ssvalue

cgs8     gate  source                                ssvalue
cgd8     gate  drain                                ssvalue
cgb8     gate  body                                 ssvalue
cbs8     body  source                              ssvalue
cbd8     body  drain                              ssvalue

mcs8g    gate  body  source body  ssvalue
mcs8d    drain body  source body  ssvalue
mcd8g    gate  body  drain  body  ssvalue
mcd8s    source body  drain  body  ssvalue

```

Note that the 'createnet' script has a feature that deletes those small signal elements which are redundant. For example, if the source and the body of a MOS are connected together, the following elements would not be seen in 'netlist.ss':

{gmb, cbs, mcs8g, mcs8d, mcd8s}.

These elements are redundant because their active nodes are connected between the source and the body, and thus, to the same node.

Once the 'netlist.ss' file is created, the user performs some editing to make the file more useful for Maple analysis. Modifications may include the deletion of components which are believed to add little effect on the small signal analysis. Although not necessary for small circuits, it may become necessary to simplify the small signal equivalent circuit of some device instantiations in order to maintain a circuit matrix of a reasonable dimension. A high dimension matrix is inherently very difficult to solve by Maple, as the amount of memory and CPU time required increases extremely fast. Through experimentation, it has been found that a matrix

dimension higher than 15 is impractical to solve, as the workstation CPU time required is of the order of an hour, using over 64 Megabytes of RAM. In contrast, a matrix of dimension 8 (that is, a circuit with 8 nodes) will require less than 10 seconds of CPU time and about 2 Megabytes of RAM. This benchmark has been performed on a Sun Sparc 20 UNIX workstation.

The user can also replace the *'ssvalue'* keyword by the keyword *'name'*. As mentioned above, the keyword *'ssvalue'* means that the corresponding element will be evaluated numerically later in the flow. If the keyword *'name'* is instantiated instead, then this will tell the *'maplenet'* C program that this element is to be evaluated symbolically. The *'maplenet'* program will use the instance name (for example, *'gmE'*) as the symbolic name. Alternatively, the user can simply specify any alphanumeric name in place of the *'ssvalue'* token, and the *'maplenet'* program will understand this as being the symbolic value for the element. This feature can be useful when two similar elements are understood to track each other. In such a case, they can be given the same symbolic name by the user. This would be the case for the transconductance of a matched differential input pair, for instance.

Note that in order to generate a transfer function, it is important to define an excitation source. Normally, the physical netlist will not include such a source, as it is not part of the circuit, strictly speaking. The simulator input deck is structured such that it comprises all simulator commands and the test bench, and it includes the circuit netlist during execution. Consequently, it is important to provide such an excitation source in the file *'netlist.ss'* so that Maple will be able to derive the transfer function. This source can be explicitly added by the user after *'createnet'* is run, or it can be instantiated in a file called *'header.ss'*. The file *'header.ss'* is simply a header for *'netlist.ss'*.

extractmisnan. This script is used to extract the small signal parameters of all MOS transistors by reading and parsing the output listing file produced by the Eldo simulator. Since Northern Telecom's MISNAN MOS transistor models are used, the script essentially extracts the MISNAN small signal parameters from the output file, and hence the name of the script. Note that the small signal sub-circuit is a standard linearized mode¹ which includes transconductances, capacitances, mutual

capacitances and conductances. Figure 8 on page 67 illustrates the small signal model used by the *'extractmisnan'* script, and it is a simplified version of the full MISNAN model.

Once the small signal parameters have been extracted from the simulator output file, the script parses the *'netlist.ss'* file in search of the keyword *'ssvalue'*. It replaces the *'ssvalue'* keyword by the corresponding numeric value for the small signal element. The results are written to the *'netlist.maple'* file. Finally, the script will also write the *'lookup_table'* file described previously in section 3.3.2 on page 48.

deltagen. This script is used to generate the lookup tables used to compute the physical sensitivities. These tables (*'lookup_l10'*, *'lookup_w10'*, and *'lookup_lw10'*) are then used in Maple by the *'deltagen'* routine of the *'smallsignal'* package.

The *'deltagen'* script takes three arguments as its input. The first argument is either *'l'*, *'w'* or *'lw'*, and it tells *'deltagen'* to parse the netlist file and modify the lengths, widths, or simultaneously the lengths and widths respectively. The second argument is the factor by which the lengths and/or widths are to be modified. This factor is normally 1.1, implying that the quantities are to be increased by 10%. The last input argument is the name of the lookup table output file, which should correspond to one of *'lookup_l10'*, *'lookup_w10'* or *'lookup_lw10'*. The input file netlist is called *'netlist'*. The script has been designed in this way to allow freedom at a later stage when other types of simulations are supported, such as for temperature sensitivities, process sensitivities, voltage sensitivities, bias conditions sensitivities, and the like.

Since the lookup tables generated have the same format as *'lookup_table'*, the *'extractmisnan'* script is used to extract the small signal parameter values out of the simulation output.

maplenet. This C program assumes that the file *'netlist.maple'* exists in the directory in which it is invoked, and it produces an MNA matrix as its output. The output is printed on the screen but can be redirected to the *'matrix.maple'* file, so that Maple can later read it in its memory. If no argument is specified, the program will pro-

duce a user readable matrix. If the option '*maple*' is stated, then it will produce a series of Maple commands which correspond to the MNA circuit matrix.

3.4.2 The Maple *smallsignal* package

The Maple '*smallsignal*' package is a collection of author-written Maple routines specifically developed to analyze the MNA matrix generated by the '*maplenet*' program. The first routine which needs to be invoked is '*mnasolve*', which solves the circuit matrix, and generates a transfer function as its result. From this solution, several types of plots can be generated. Normally, a Bode plot is created in order to graphically compare the Maple solution with the simulator's as a sanity check. A sensitivity analysis can be performed which computes the relative sensitivities for all roots with respect to all symbolic parameter or with respect to the size of the active devices. This is useful in helping to understand how the circuit functions, and to understand how the roots can be placed in the complex plane. Once the sensitivity analysis is carefully examined, the designer can reach conclusions and verify their validity using the root locus plotting routine. This routine will plot the roots in the complex plane as a function of a parameter. This allows the designer to graphically see how the roots will move when a given parameter is changed. In a similar function, three dimensional Bode plots can also be generated where the third dimension is the parameter being swept.

3.4.3 The *mnasolve* routine

This routine is used to solve the MNA matrix previously read into Maple's memory. There are two arguments to the routine: a list and the name of the unknown. The list contains three items: the MNA matrix, the source vector, and a list of all unknowns found in the

matrix. This list is created by the 'maplenet' program, and it is assigned the variable name 'mna1ist' in the 'matrix.maple' file, so the user needs only to specify 'mna1ist' as the first argument to 'mna1olve'. The second argument is the name of the unknown Maple is seeking to solve. The 'mna1olve' routine finds the transfer function between the given unknown and the excitation source specified in the source vector which was previously instantiated in 'netlist.ss'. Figure 11 illustrates what the Maple worksheet should look like when invoking 'mna1olve'. The lines starting with the '>' character correspond to user-entered command lines.

Figure 11 Invoking the 'mna1olve' routine within the Maple worksheet

```
> with(smallsignal);
> read 'matrix.maple':
Warning: new definition for      norm
Warning: new definition for      trace

      unknowns := [ p  n  iinput  vc1  vc2  ir13  vout  vbias  vx  vm ]
                  Use mna1ist as input argument to mna1olve.
> vo := mna1olve( mna1ist, vout );
                  Solving for parameter vout in row #7
```

This figure illustrates the three steps required to invoke the 'mna1olve' routine. The first step is to tell Maple to use the 'smallsignal' package. The Maple 'with' command performs this function.

The second step is to read the 'matrix.maple' file into Maple's memory, and to execute it. The 'read' command achieves this. The commands executed in the file print the vector of unknowns, which tells the designer which variables are available in the matrix and thus can be solved by Maple. Finally, the third step is to invoke 'mna1olve' by specifying the

'*mna*' variable, followed by the name of the parameter the routine is to solve for which is, in this case, '*vout*'.

The routine solves the matrix using Cramer's rule[26] using sparse techniques to evaluate the required determinants, as explained earlier. This is acceptable for relatively small circuits, with less than about 15 nodes. Much research has been done recently to find ways to solve larger symbolic MNA matrices[14]. One method emerging from the research involves the approximation of the symbolic expression while the expression is being generated. The motivation is to perform the analysis much faster since no time needs to be wasted in generating the symbolic terms that would normally be discarded later in the simplification process. Moreover, the memory needs will be much smaller thus allowing larger circuits to be analyzed. This approach is promising and should be regarded as a possible improvement to the '*smallsignal*' package.

The concept of improving efficiency in solving the symbolic matrix is an important one which deserves much research attention. However, one must also remain practical when developing a useful circuit design tool. The goal of this thesis is to develop a useful tool for industrial analog circuit designers. Although it would be nice to have a more efficient algorithm to solve larger circuits, there are many relatively small circuits or sub-circuits which can be solved very effectively using Cramer's rule with sparse techniques. The research efforts in this thesis were therefore focused on developing the infrastructure required to make the tool useful from a design knowledge development, capture and reuse point of view. However, improvements of the algorithm can be considered to be an excellent topic for future improvements to the '*smallsignal*' package.

3.4.4 The analysis routines

This section provides a user reference description of all analysis routines available within the '*smallsignal*' package.

reduce_order. This routine is used to reduce the order of the expression generated by the '*mnasolve*' routine. The algorithm used has been previously described in section 3.3.4 on page 57. The routine requires four input arguments: the transfer function, the coefficient tolerance (*ctol*), the minterm tolerance (*mtol*), and a list of frequency points. The meaning of '*ctol*' and '*mtol*' have been previously defined in section 3.3.4. The list of frequency points is simply a list of discrete frequencies at which the reduced expression is to be evaluated in order to determine its relative deviation from the exact expression. However, the list can also be made of four elements which are arguments to automatically generate a list of discrete frequencies increasing logarithmically. The first element of this list is the keyword '*log*', the second element (*fmin*) is the starting frequency, the third element (*fmax*) is the end frequency, and the fourth element is the number of points (*N*) to be generated. This tells '*reduce_order*' to create a logarithmically increasing set of *N* frequency points starting at '*fmin*', and ending at '*fmax*'. It provides a short-hand method to specify a set of frequency points.

The output of '*reduce_order*' is a list of two elements: The first element is the reduced expression, and the second element is the maximum observed relative error of the expression when compared against the exact expression.

Figure 12 provides an example of what '*reduce_order*' can print upon invocation. Note where the two arrows are located. The list of absolute sensitivities given corresponds to the total weight of the numeric minterms when compared with the total coefficient value. Thus, the user can use these values to tighten the value of '*mtol*' as required until it can be guaranteed that the expression has not converted an overly large number of minterms into their numeric equivalent. It is therefore useful in assessing the accuracy of the reduced expression in sensitivity analyses. Also, note that the sequence of the 's' order terms shown in the list has a direct mapping to

the corresponding numeric list. The fact that the sequence of the 's' order terms is not in ascending order is a by-product of Maple's internal data structure manipulations.

Figure 12 Example of the output produced by the 'reduce_order' routine

```

> vo_reduced := reduce_order( vo, 0.01, 0.002, [log, 1e3, 1e10, 15] ):

fmin:                .1e4
fmax:                .1e11
Number of Frequency Points: 15

List of Frequency Points:
=====

[1000., 3162., 10000., 31620., 100000., 316200., .1000 107, .3162 107, .1000 108, .3162 108, .1000 109, .3162 109,
.1000 1010, .3162 1010, .1000 1011]

      The s terms kept in the numerator are.: [s, s2, s3, s4, s5, s7, s6]
      The corresponding absolute maximum minterm sensitivity error is.:
      [.0017870167, .0025140334, .009547750, .0018182768, .014109288, .000042490, .0032518300]
      The s terms kept in the denominator are.: [s, s2, s3, s4, s5, s7, s6]
      The corresponding absolute maximum minterm sensitivity error is.:
      [.0083929871, .0165381588, .0419428945, .0800754816, .0724421027, .012013141, .0345726528]

At frequency 1.000000e+03: Relative Error is 1.15e-12
At frequency 3.162000e+03: Relative Error is 2.23e-12
At frequency 1.000000e+04: Relative Error is 9.01e-12
At frequency 3.162000e+04: Relative Error is 1.28e-11
At frequency 1.000000e+05: Relative Error is 3.55e-11
At frequency 3.162000e+05: Relative Error is 2.11e-10
At frequency 1.000000e+06: Relative Error is 7.89e-13
At frequency 3.162000e+06: Relative Error is 2.00e-10
At frequency 1.000000e+07: Relative Error is 4.84e-12
At frequency 3.162000e+07: Relative Error is 1.11e-11
At frequency 1.000000e+08: Relative Error is 1.00e-10
At frequency 3.162000e+08: Relative Error is 1.07e-10
At frequency 1.000000e+09: Relative Error is 4.81e-12
At frequency 3.162000e+09: Relative Error is 1.46e-10
At frequency 1.000000e+10: Relative Error is 4.93e-12

```

Maximum relative error observed, 2106105221 10⁻⁹

Figure 13 Example of the output produced by the 'print_sensitivities' routine.

```

> sensitivities := print_sensitivities( vo_reduced[1], 0.1 ):

Sensitivity Cutoff = 10.0 %
DC gain = 64.28 dB
Root Frequencies:
=====
[ $z_1 = -202 \cdot 10^8$ ,  $z_2 = -232 \cdot 10^9$ ,  $z_3 = .509 \cdot 10^9$ ,  $z_4 = -.104 \cdot 10^{10} + .320 \cdot 10^7 i$ ,  $z_5 = -.104 \cdot 10^{10} - .320 \cdot 10^7 i$ ,
 $z_6 = .134 \cdot 10^{10}$ ]
[ $p_1 = -87000.$ ,  $p_2 = -.197 \cdot 10^8$ ,  $p_3 = -226 \cdot 10^9$ ,  $p_4 = -.285 \cdot 10^9$ ,  $p_5 = -.599 \cdot 10^9$ ,  $p_6 = -.106 \cdot 10^{10}$ ]

Sensitivities:
=====
table([
  c_gsl = [-.52 = z_4, -.52 = z_5, -.43 = p_3, .23 = p_4, -.41 = p_5, -.22 = p_6]
  c_gdl = [-.43 = z_3, .33 = p_3, -.41 = p_6]
  gm_4 = [1.0 = DCgain, .82 = z_2, 1.0 = z_3, .12 = p_2, .50 = p_3, .69 = p_4, -.54 = p_5, .15 = p_6]
  s_dsl = [-.12 = DCgain, .13 = p_1, .17 = p_3, -.17 = p_4]
  c_gsl9 = [.17 = p_3, -.79 = p_5]
  c_gdl9 = [-.44 = z_6, -.18 = p_1, -.55 = p_3, .83 = p_4, -.10 = p_5, -.31 = p_6]
  s_dsl9 = [-.85 = DCgain, .85 = p_1, .15 = p_3, -.15 = p_4]
  c_gsl1 = [-.22 = p_3, .27 = p_4, -.12 = p_6]
  gm_1 = [.47 = z_4, .47 = z_5, 1.2 = p_3, -.76 = p_4, .45 = p_5]
  c_10 = [-.45 = z_4, -.45 = z_5, -.18 = z_6, -.76 = p_1, .28 = p_3, .88 = p_5, -.14 = p_6]
  r_13 = [-.78 = z_4, -.78 = z_5, .38 = z_6, -.19 = p_3, .17 = p_4, 1.2 = p_5, -.22 = p_6]
  gm_9 = [.99 = DCgain, -.23 = z_4, -.23 = z_5, 1.4 = z_6, -.93 = p_1, .13 = p_3, .32 = p_4, 1.0 = p_5, -.61 = p_6]
  c_ld = [.10 = p_4, -.65 = p_5, -.18 = p_6]
  s_dsl9 = [-.93 = DCgain, .90 = p_1, .11 = p_3, -.11 = p_4]
])

```

print_sensitivities. This routine prints the symbolic sensitivities of the roots and the DC gain of the argument expression with respect to all symbolic elements. Figure 13

gives an example of the output generated by the *'print_sensitivities'* routine. There are two input arguments to this routine. The first argument is the symbolic expression to be analyzed, and the second argument is a cutoff point for symbolic sensitivities. Although the symbolic sensitivities for all roots are computed with respect to all symbolic elements, the designer is interested in those which affect the root by a relatively large amount. Thus, the cutoff value is a percentage which is used to discard all symbolic sensitivities below this value. For instance, if a 1% change in a particular symbolic parameter causes a 0.01% change in a pole, it is likely that the designer will consider this effect as irrelevant and it should be discarded. However, if a 1% change in a parameter affects the location of a pole by 3%, then this is likely to be important to the designer, and the cutoff should be chosen so that it is not discarded.

Note that this routine should always be used with a reduced expression. This is because the computation of all required partial differentials is CPU intensive, so it is wise to try to minimize the CPU time by using a reduced expression as input argument.

Figure 14 Output produced by the *'deltagen'* routine

```
> lsens := deltagen( vo_reduced[1], lookup_110, sensitivities[1], [1, 4, 9], 0.1 ):
```

```
Sensitivities for parameter: 1
-----
```

$$\left[\begin{array}{l} 0.03831 = c_{gd1}, -4.598 = gm_4, -2.492 = g_{ds4}, .9729 = c_{gs9}, .05344 = c_{gd9}, -1.916 = g_{ds1}, .08264 = c_{gd1}, \\ -4.510 = gm_1, 0 = c_{10}, 0 = r_{13}, -.7603 = gm_9, 0 = c_{12}, -2.312 = g_{ds9}, 1.024 = c_{gs1} \end{array} \right]$$

MOS 1 sensitivities using parameters, $\{g_{ds1}, c_{gd1}, gm_1, c_{gs1}\}$, are:

$$[-1.6 = p_1, -1.3 = p_3, .89 = p_4, -.74 = z_5, -.74 = z_4, 1.6 = DCgain, -.66 = p_5, -21 = p_6]$$

MOS 4 sensitivities using parameters, $\{c_{gd4}, gm_4, g_{ds4}\}$, are:

$$[-.32 = p_1, -.64 = p_3, .11 = p_4, -.41 = z_2, -.51 = z_3, -.16 = DCgain, .19 = p_5]$$

MOS 9 sensitivities using parameters, $\{c_{gs9}, c_{gd9}, gm_9, g_{ds9}\}$, are:

$$[-1.4 = p_1, -.22 = p_3, .20 = z_5, -1.1 = z_6, .20 = z_4, 1.4 = DCgain, -1.6 = p_5, .42 = p_6]$$

deltagen. This routine is used to generate the sensitivities based on a physical parameter and the results of an incremental simulation, performed by the '*deltagen*' script. The algorithm was previously defined in section 3.3.5 on page 61. Figure 14 provides an example of how '*deltagen*' can be used and a typical output. The first argument is the symbolic expression while the second argument is the lookup table containing the values of the symbolic parameters from the incremental simulation. The third argument is the list of sensitivities previously computed by '*print_sensitivities*'. The fourth argument is a list of the transistors which the designer requires to obtain the sensitivities, and finally, the fifth argument is the sensitivity cutoff point, which tells the routine to print those roots which are more sensitive to the physical parameter than this cutoff point.

The output states the physical parameter under consideration, and how it affects each symbolic parameter. It also gives the list of symbolic parameters which were used to compute the sensitivities for each transistor i.e. which partial derivatives were used, as well as the sensitivities of the roots as affected by the parameter of the transistor. Note that the list of partials is very important in assessing whether the total sensitivities are accurate. This is because the total net sensitivity is the sum of all partial derivatives contained in that list. Thus, for a reasonable amount of accuracy, the elements comprising the first order small signal MOS model should be part of that list. In other words, the elements '*gm*', '*c_{gs}*', '*g_{ds}*' and '*c_{gd}*' should be included for reasonable accuracy.

bodeplot. This routine generates a phase and magnitude Bode plot of the expression given. The input arguments include the expression to be plotted, the minimum frequency and the maximum frequency. If the keyword '*print*' is specified as the fourth argument, then the numeric transfer function will be printed. The plotting procedure needs to substitute all symbolic elements for their numeric value. These values are found in the '*lookup_table*' file generated by the '*extractmisnan*' script.

bodeplot3d. This routine is similar to the '*bodeplot*' routine except that the third dimension is a symbolic parameter. This symbolic parameter is swept over a given range of values, so that the designer can see the effect of modifying this value on the

bode plots. The input arguments include the expression, the minimum frequency, the maximum frequency, and a sweep range stated in the form:

`<symbolic name> = <minimum value> .. <maximum value>`

If the keyword *'print'* is specified as the fifth parameter, then the numeric transfer function is also printed. The numeric values used to print the transfer function are found in the *'lookup_table'* file, including the numeric value for the parameter swept.

rootlocus. This routine plots the roots (poles and zeroes) in the complex plane. In a manner similar to the *'bodeplot3d'* routine, a parameter with its sweep range is also given. The *'rootlocus'* function plots the roots in the complex plane for a number of values of the sweep parameter within the sweep range. In this way, the user can view the movement of the roots in the complex plane as the parameter is swept.

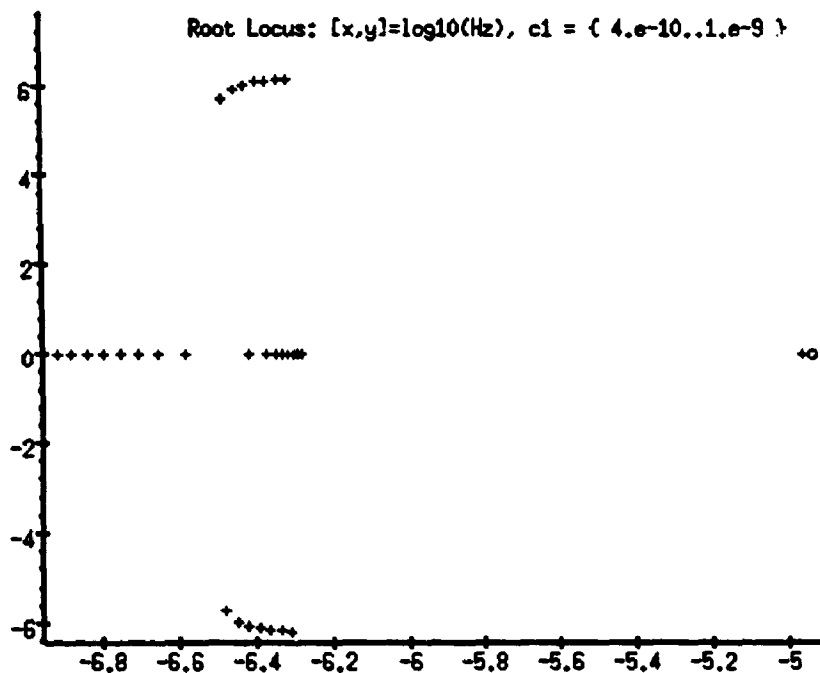
The *'rootlocus'* routine works as follows: There are five parameters which must be specified upon invoking the routine. The arguments include the symbolic expression, the parameter name, the sweep range, the number of frames to be generated, and the display method. The symbolic expression and the sweep parameter name are straightforward. The sweep range is stated as the minimum number, followed by two periods, followed by the maximum number. For instance, *'1..5'* is a valid range, from 1 to 5.

The *'rootlocus'* routine generates a series of graphic frames which it can display in sequence. Each of these frames correspond to a complete view of the complex plane, with all roots, but for a particular value of the sweep parameter. Hence, if the frames are all displayed in a sequence corresponding to, say, ascending values of the sweep parameter, then animation will result - that is, the user will view how the poles and zeros move when the sweep parameter is increased. Hence, the purpose of the *'number of frames'* argument is to specify how many such frames need to be generated. The values of the sweep parameter are then calculated by distributing them in a logarithmic fashion between the smallest value and the largest value within the specified range.

The method can be either the keyword *'inseq'* or the keyword *'animate'*. If *'inseq'* is chosen, then all frames are superimposed on top of each other, in sequence. Thus, the trajectory of the roots can be viewed.

If *'animate'* is chosen, then a special menu appears on top of the graph, which allows the user to animate the plot and display each frame either in ascending order or in descending order, one at a time. The animate method is useful to examine how fast poles and zeros can move, and where they move for specific values of the sweep parameter. Figure 15 provides an example of the plot generated by the *'rootlocus'* routine with the *'inseq'* option. Note that the axes are in logarithmic Hertz.

Figure 15 Example of the *'rootlocus'* plot using the *'inseq'* option



expression_sensitivities. This routine is similar to the *'print_sensitivities'* routine except that it only computes the normalized sensitivities of the expression with respect to all symbolic parameters. The sensitivities are calculated at the numeric point found by substituting the values found in *'lookup_table'*. Figure 16 provides an example

of the output generated by '*expression_sensitivities*'. The first argument is the expression, whereas the second argument is the cutoff point.

Figure 16 Example of the output generated by '*expression_sensitivities*'

```
> expression_sensitivities( zeta, 0.01 ):

Expression value = .3716.  Sensitivities are:
[-.54 = c1, -1.1 = r2, .012 = c2, -.59 = ko, -.55 = kpd]
```

3.4.5 The support routines

display_indets. This routine is used to display the numeric value of all symbolic elements which appear within an expression. The nominal numeric value for each symbolic element is taken from the file '*lookup_table*', and thus corresponds to the element value simulated at nominal condition. The first argument of the routine is the symbolic expression to be analyzed. If the keyword '*pvit*' is specified as the second argument, then the maximum values (taken from the '*lookup_max*' file) and the minimum values (taken from the file '*lookup_min*' file) are also printed. Moreover, the normalized difference for each element is also printed with the '*pvit*' option. This normalized difference gives a percentage variation and is calculated as follows:

$$\frac{\langle \text{maximum value (lookup_max)} \rangle - \langle \text{minimum value (lookup_min)} \rangle}{\langle \text{nominal value (lookup_table)} \rangle} \quad (\text{EQ 6})$$

subst_values. This routine returns an expression with the specified symbolic elements substituted with their corresponding numeric values. The numeric values are again found in the file '*lookup_table*'. The input arguments to this routine include the symbolic expression as the first argument, and a list of symbolic names which the user wants substituted in the expression as the second argument.

3.5 Automated verification and design centering using simulations

It is clear that hybrid symbolic analysis cannot replace the circuit simulator. The purpose of hybrid symbolic analysis is to help the designer to gain insight into the circuit being analyzed. Hence, design knowledge is developed and, using Maple's worksheet interface, this knowledge is captured in the form of an executable document. Circuit simulators can then be used to get a more accurate solution, as the designer is then equipped with the necessary relationships relating observations to theory, and helping him to perform the optimization via numeric simulations.

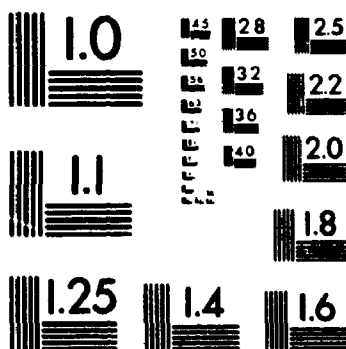
3.5.1 Design centering and circuit characterization

Once a design is well understood, it must be optimized in a way that will make it manufacturable. Yield optimization, for instance, can include several key components including parametric optimization and layout optimization for improved yield. Layout optimization can include the optimization of certain parameters such as compensating for layout mismatch effects, as well as other, perhaps more severe manufacturing considerations. For instance, most complex submicron processes seem to present yield difficulties with their metal to metal vias. One way to help improve yield in such a case is to lay out the circuit with double vias at all place where a via is required. In general, yield optimization is a complex issue which requires design consideration, layout consideration, and manufacturing considerations. For the purpose of this work, only the issue of design centering is discussed.

Design centering can be defined as the optimization of a design so that all of its parameters fall into their respective acceptable window of operation. This implies, for instance, in the

2 of / de 2

PM-1 3 1/2" x 4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT



PRECISIONSM RESOLUTION TARGETS

case of an amplifier that the phase and gain margin will always fall within a given window over all process, voltage, bias currents and temperature variations.

Design centering using automated simulations is often better than simple circuit optimization. This is because several worst case corners can be chosen to help the designer in choosing optimal values for the relevant design parameters, in such a way as to minimize the variation in the relevant design specifications (phase margin, bandwidth), while obtaining a center value that is appropriate for the operating conditions. Once a design is well understood, and it is reasonably optimized under nominal operating conditions, it is important to center its physical and electrical design parameters in order to maximize its parametric yield under all manufacturing and environmental conditions. This process can be done in two steps.

1. First, the design is subjected to all possible simulation corners. This is easily done using the '*autopvit*' script. For each simulation, the simulation output file is post-processed to extract the relevant parameters. For example, in the case of an amplifier, the DC gain, the phase margin, the unity-gain bandwidth and the gain margin can be extracted. These values are then appended to a report file. After all simulations are completed, the designer can view the report file and see how each simulation corners have affected the parameters. This information can then be fed back to the circuit in order to increase its robustness to PVIT variations. The final report can also be used as a characterization report/performance data sheet for the circuit. System designers can refer to this data sheet to evaluate the performance of the resulting circuit building block.
2. Once the circuit has been centered using '*autopvit*', then the actual parametric yield can be estimated using Monte-Carlo analysis. If the circuit is tolerant to all simulation corners, then its parametric yield is theoretically 100%, so Monte-Carlo analysis becomes redundant for statistical simulation over the specified corners.

The corners chosen for the case study developed in section 4.0 on page 91 were automatically generated using a simple C program. These corners include all permutations of best and worst case process for the NMOS, PMOS, resistor and capacitor, as well as best and worst case temperature, supply voltage and reference bias current. Hence, there are a total of 128 permutations. Other corners could be included such as mismatch which could be achieved by modulating the geometry of specific transistors, for instance. The purpose of this effort is to show the importance of the concept, as its practical implementation will vary depending upon what simulation corner is considered important for a given application.

Upon invocation of the *'autopvit'* script, the user needs to specify either three or four arguments. The first argument is the name of the file where the simulation corners are described. This file is what is referred to as the *'pvitcases'* file. The second and third arguments are the start and end simulation numbers. This is useful because the user often will not need to run all 128 simulations. The optional fourth argument is the keyword *'plots'*, which tells the script whether or not it should generate graphical output plots which the designer can view. The plots feature is useful when the designer needs to recreate a particular simulation corner case, and graphically examine the simulation results.

An example output of the report file is shown in Figure 17. The simulation number is used as a cross-reference to the *'pvitcases'* file, where the actual simulator options are defined for the corner case.

An example simulation corner case from the *'pvitcases'* file is shown in Figure 18. The ampersands are used as case delimiters. The simulation corner parameters are then de-

scribed, either as absolute values such as for 'vcc', or as multiplication factors, such as the 'iscale' parameter, which scales the bias current down to 90% of its nominal value.

Figure 17 Example of the autopvit.out report file

Analysis 1: Opamp Open-loop Frequency Characterization

<i>Simulation</i>	<i>DC Gain</i>	<i>UGBW</i>	<i>Phase Margin</i>	<i>Gain Margin</i>
0	64.3	1.26e+08	40.6	8.9
1	68.7	8.41e+07	39.5	8.7
2	69.2	9.44e+07	36.3	8.5
3	60.4	8.41e+07	49.6	10.5
4	61.1	9.44e+07	47.1	11.1
5	68.3	9.44e+07	39.6	8.8
6	68.8	1.06e+08	36.7	9.3
7	60.0	9.44e+07	49.6	10.5
8	60.8	1.06e+08	47.4	11.0
:	:	:	:	:

Figure 18 Example of a simulation corner case as presented in the 'pvitcases' file

#####

Sim 2

```
.param vcc = 4.5
.param cscale = 1.2
.param rscale = 1.3
.temp 105
.options tnom = 105
.param iscale = 0.9
.lib {$techlibpath}/batmos.ntmos LO_N_HI_P
```

** The following are fixed for ALL Simulations*

```
.param ang = 2.5
```

4.0 Case Study - Amplifier Design

This section evaluates the proposed design method using the Maple-Eldo prototype. The circuit topology studied is the traditional basic two-stage CMOS pole-splitting operational transconductance amplifier. Feedback circuits such as amplifiers are good candidates for small signal analysis, since the feedback loop is generally analyzed using linear control theory. The amplifier is designed using Northern Telecom's BATMOS process, a 0.8 μm BiCMOS analog process technology. The aim is to demonstrate how the Maple-Eldo prototype can be used to analyze a circuit, to understand the trade-offs, and to optimize the circuit in order to meet a set of given specifications. Section 4.1 of this chapter describes the specifications and general functionality of the amplifier from a large signal point of view. Section 4.2 is a discussion of the methods used for designing the amplifier using the Maple-Eldo prototype.

4.1 Basic two stage operational transconductance amplifier

An operational transconductance amplifier is, generally speaking, a high gain voltage to current converter. Normally, the performance of the amplifier is limited by a number of parameters [24], such as:

- finite gain and linear range
- offset voltage
- common-mode and power supply rejection
- frequency response
- slew rate
- finite output resistance (transconductance amplifier)
- noise
- input dynamic range

- DC power consumption

At a minimum, a subset of these parameters must be considered during the design and optimization of the amplifier. In the current design, the slew rate, the frequency response, and the finite gain shall be considered. In addition, the amplifier is required to have at least 50 degrees of phase margin and at least 15 dB of gain margin at the worst case simulation corner, in order to ensure stability and a reasonable settling time.

4.1.1 Functional Description of the amplifier

Figure 19 is a schematic representation of the basic two-stage pole-splitting operational amplifier studied. The reference bias current is chosen to be $50\mu\text{A}$ based on the initial design efforts described later in section 4.1.2 on page 94. The principle of operation is described as follows: Transistors $m3$ and $m4$ provide a current-steering mechanism which controls the amount of current feeding into the controlling transistor $m2$ of the current mirror made of $m1$ and $m2$. When the n and p input of the differential pair are at the same voltage, then a current equal to half of the current provided by $m5$ will flow through each of $m3$ and $m4$. Since the dimensions of $m5$ are three times that of $m7$, the current provided by $m5$ is $150\mu\text{A}$, and the current flowing through each of $m3$ and $m4$ will be $75\mu\text{A}$. Since $m1$ and $m2$ are identical transistors, both the driving transistor $m2$ and the mirror transistor $m1$ will see their drain voltage (V_{ds}) to be equal to their gate voltage (V_{gs}) by virtue of the fact that they are driven by identical currents. By varying the voltage seen at the p and n input of the differential pair, it is possible to steer mismatched currents through the branches of the differential pair $m3$ and $m4$. Since the current through $m1$ is essentially controlled by the current in $m2$, a mismatch in $m3$ and $m4$ current will translate into a

gate voltage at m1 and m2 are the same as their drain voltages. This voltage is in turn applied to the gate of m9, as the drain of m1 is connected to the same node. If m5 drives the same current as m8, then, under balanced conditions, m9 should drive twice as much current as m1, because only half of the current from m5 flows into m1. It follows that m9 must be made twice as wide as m1 to maintain balance. Although in theory it is the W/L ratio that is important, we may choose to retain the same gate length in all m1, m2 and m9 transistors as this also affects the output resistance, which in turns affects the DC current of the transistors. However, if a small amount of systematic offset is permissible, then it may be possible to vary the gate length of m9 somewhat away from the m1 and m2 length, if it offers other advantages.

4.1.2 Initial design effort

For the purpose of discussions, let's assume the goal is to design an amplifier for a switched-capacitor application. The specifications of the switched-capacitor circuit is a sampling rate of 5 MHz, thus providing a period of 200 ns. Since switched-capacitor circuits require a reset phase along with an integration phase, only half of that time is available for the integration phase, assuming a 50% duty cycle of the clock. This means that both amplifier slewing and settling must occur within the resulting 100 ns period. A minimum of eight time constants of settling time is required between each samples to ensure virtually no correlation between successive samples. It is therefore reasonable to assume a minimum amplifier bandwidth of 100 MHz when subjected to its normal capacitive load. This implies that the amplifier bandwidth time constant is 10 ns. A settling time of eight time constants thus requires 80 ns. The remaining 20 ns can be used for amplifier slewing

under worst case conditions. The amplifier capacitive loading is chosen to be 0.5 pF, as this provides for a low enough thermal noise floor, while maintaining a large enough geometry to ensure good matching between capacitors. The amplifier operating voltage is 5 volts.

The minimum slew rate required can be calculated as follows:

$$\text{Minimum slew rate} = \frac{dv}{dt} = \frac{5\text{Volts}}{20\text{ns}} = 250 \times 10^6 \frac{\text{Volts}}{\text{sec}} \quad (\text{EQ 7})$$

Let's provide additional margin and assume a slew rate of $300 \times 10^6 \text{ V/s}$. For a 0.5 pF compensation capacitor, this assumes an input stage current of:

$$\text{Input stage current} = C \times \frac{dv}{dt} = 0.5\text{pF} \times 300 \times 10^6 = 150\mu\text{A} \quad (\text{EQ 8})$$

The output stage current needs to slew both the compensation and the load capacitance. Since they are each equal to 0.5 pF, the total capacitance seen at the output node is 1 pF. By following the same development as in (EQ 8), it follows that the output stage current required is $300\mu\text{A}$.

The above slew rate argument combined with the need to eliminate the systematic offset provides the necessary information to compute the relative W/L ratios of all transistors except for the input pair. Actual numbers for W can be obtained once the transistor length is determined. Since m5, m8 and m7 are DC current sources, a longer channel length is warranted, especially since it will help the DC gain by providing a larger output resistance. The channel length need not be set too long as this would translate into a large W, and

hence, a large capacitance coupling from the drain to the V_{dd} supply through the gate, an unwanted scenario. A channel length of 3 μm has been chosen bearing these arguments in mind. On the other hand, transistors m1, m2 and m9 were chosen to have a channel length of 1.5 μm because they need to operate at the amplifier's unity gain frequency, and thus need to be faster, even at the expense of a lower output resistance. The input stage transistors m3 and m4 are chosen to be very wide in order to maximize the input stage transconductance, and thus improve the amplifier DC gain. Another benefit of having large input transistors is a reduction of the flicker noise, also known as 1/f noise. The channel length should not be overly small in order to maintain a reasonable matching between m3 and m4, and thus maintain a lower random offset voltage. Hence, the input transistors are chosen to have a dimension of $200/2$. The sampling error introduced by the finite amplifier DC gain does not affect the accuracy of the switched capacitor system because for most successive samples, the error is in the same polarity as the sample itself, and therefore the consequences are generally a systematic offset in the DC gain of the system. However, if the sampling error is large, an unacceptable amount of correlation between successive samples may result. The minimum acceptable DC gain of the amplifier is specified to be 60dB, a value which is considered to be a reasonable minimum for most industrial switched-capacitor applications.

Note that the above method is for sizing the transistors are merely *educated guesses*. In other words, these numbers are taken strictly from the designer's experience. Clearly, this will not provide an optimized solution of the circuit. It will, however, provide a starting point where the designer can start his analysis of the design using the Maple-Eldo tool.

The design knowledge outlined above can be classified under the categories of large signal design knowledge and small signal design knowledge. The Maple-Eldo tool does not directly help the designer in understanding the large signal issues. Hence, the large signal design issues need to be addressed by hand, as done above by the author, in order to obtain a working amplifier from a large signal point of view.

The small signal design issues, however, were not directly addressed above. The only allusions to small signal issues were the *educated guesses* of the transistor channel lengths. This was done in order to accelerate the design process of the amplifier, and to limit this case study to a single Maple worksheet. It could be argued, however, that a very bad choice of transistor channel lengths would still eventually yield the correct results, because the designer should be able to optimize each transistor based on the results of the various sensitivity analyses done with the Maple-Eldo tool. The designer may need, however, to perform several Maple-Eldo analyses before the optimal solution is reached.

The small signal design knowledge described above is therefore incomplete, because it does not provide any direct means to the designer to obtain an optimized design. The goal of the Maple-Eldo tool is thus to quickly develop a thorough and complete understanding of all the small signal design issues and trade offs implied in the design. The tool can also be used to verify the theory behind the design, and thus help the designer to visualize the effects of key design parameters.

For instance, the compensation theory of the basic two stage operational amplifier states that as Miller compensation is introduced in the driving transistor of the second stage (m_9), the dominant pole will move to a lower frequency, while the non dominant poles

will tend to move to a higher frequency. Moreover, the addition of a resistor in series with the Miller compensation capacitance will bring the right hand plane feed forward zero to the left hand plane, and lower its frequency as the value of the resistance is increased. This theory is easily verifiable with the Maple-Eldo 'rootlocus' routine. Figure 20 illustrates the movement of the roots when c_{10} is swept from 0.1 pF to 1 pF. Note how the dominant pole moves to a lower frequency, while the non-dominant roots stay above 100 MHz. Figure 21 shows the Miller zero moving to a lower frequency as the value of the Miller resistor r_{13} is increased. Note that the second right hand plane zero is also pushed to a higher frequency, which should further improve the phase margin. This observation is interesting as the phase margin improvement is often attributed to the Miller zero being first located in the right hand plane, and eventually moving to the left hand plane as the Miller resistor is increased. In this case, however, two separate zeros are involved, where the right hand plane zero has been created by the gate-drain overlap capacitance of the transistor. This observation can be easily verified by the reader using the Maple-Eldo tool, and assuming an ideal transconductance gain stage (representing the transistor), along with the Miller capacitor and resistor in parallel with a capacitance, representing the gate-drain overlap capacitance of the transistor. This is left to the reader as an exercise in using the tool.

Figure 20 Pole splitting effect when $c[10]$ is increased

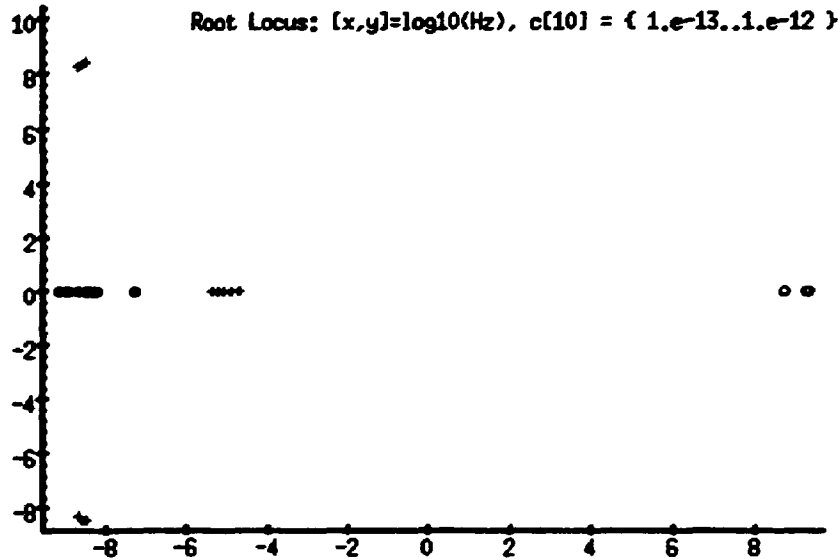
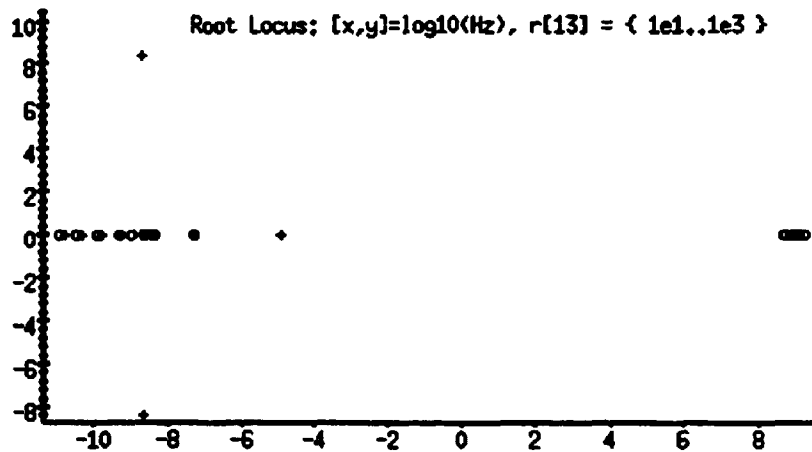


Figure 21 Miller zero moving to a lower frequency as $r[13]$ is increased



As seen from the above figures, this theoretical analysis normally done by hand by the designer, and later verified through numerous simulations, can largely be converted into computerized executable knowledge using the Maple-Eldo tool. Moreover, the analysis done by Maple is much more accurate than the theoretical analysis since it directly uses

the numeric values computed by the simulator. The next section is a detailed analysis of the amplifier using the various '*smallsignal*' package routine of the Maple-Eldo tool.

4.2 Design of the amplifier using the Maple-Eldo tool

Since the Maple worksheet is rather lengthy, it was instead included as APPENDIX B to this thesis. Note that the Maple worksheet contains designer's notes and comments regarding the analyses. These notes and comments, combined with the automatic chronological log of each analysis constitutes the essence of the worksheet concept. Although simple in concept, this adds a great deal of value to the designer because it automatically keeps track of all analyses along with his thoughts, conclusions and postulates as he proceeds with his own analysis. The reader is therefore urged to consult the appendix to get a better perspective on how a designer can use the worksheet efficiently in a typical application. Sections 4.2.1 and 4.2.2 provide instead a summarized discussion of the worksheet results, where the designer's notes and comments were paraphrased in the main text.

4.2.1 Maple worksheet for initial analysis

Once the initial sizing of the transistors is completed by hand, the designer needs to capture the schematic and perform a first simulation. This is done using the Eldo circuit simulator. The results are viewed using the Eldo waveform viewer, Xelga.

Next, the designer wishes to analyze the circuit using hybrid symbolic analysis and the Maple '*smallsignal*' package. Using the '*createnet*' script, a small signal netlist ('*netlist.ss*') is created. The resulting netlist is modified to specify which small signal pa-

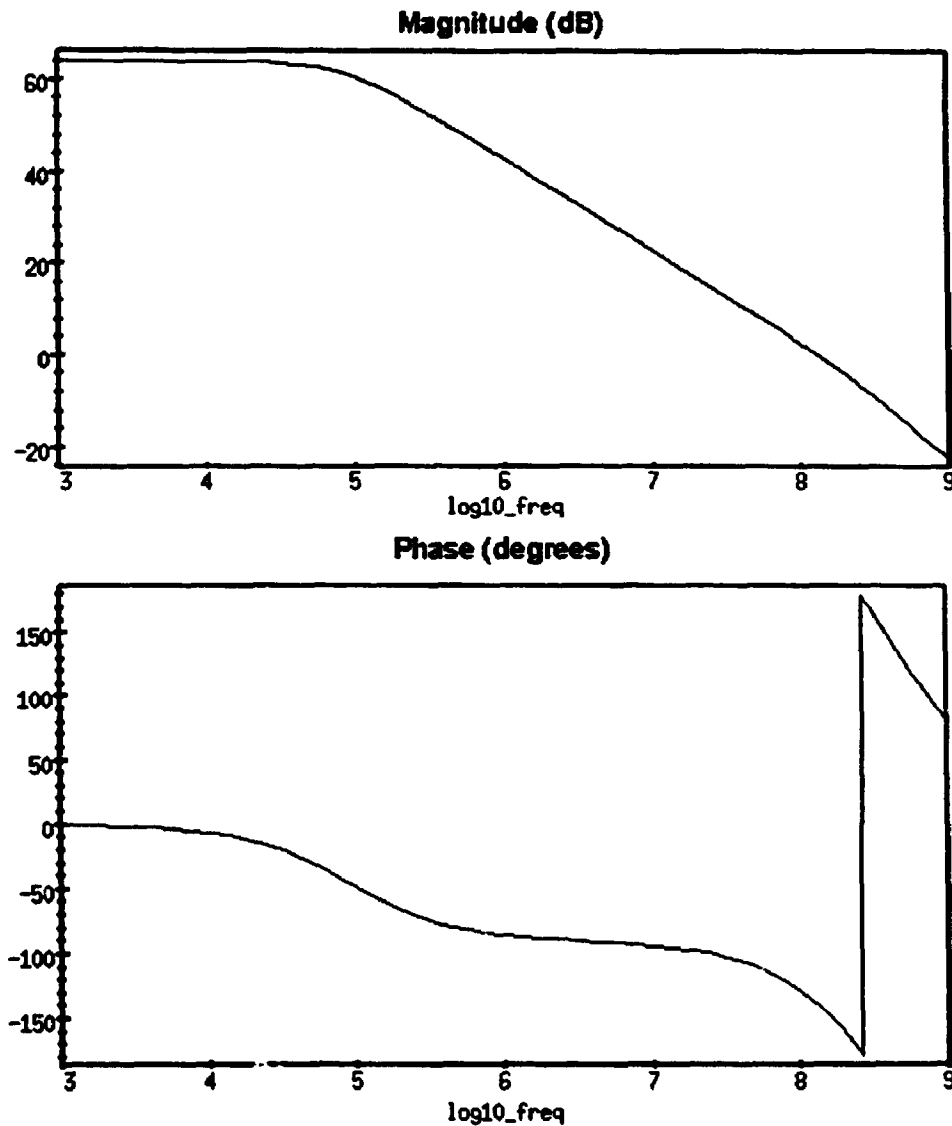
parameter is to be solved symbolically by Maple. Appendix A.2 lists the user-modified '*netlist.ss*' file.

The first phase is completed after the script '*extractmisnan*' and the '*maplenet*' program are executed. The '*extractmisnan*' script generates the '*lookup_table*' file along with the '*netlist.maple*' file, as previously explained in section 3.4.1 on page 71. The output of the '*maplenet*' program is the file '*matrix.maple*' which contains the hybrid symbolic matrix.

The second phase is to invoke Maple, and to create the worksheet, '*analysis1.ms*'. Before trying to solve the matrix, the user must tell Maple to use the '*smallsignal*' package, and he must also load in the '*matrix.maple*' file. The '*mnasolve*' routine is invoked, which returns the exact hybrid symbolic solution. At this point, the user should perform a traditional phase and magnitude Bode plot, to compare against the simulator's results. This Bode plot is reproduced in Figure 22. The author has carefully verified the similarity between this plot compared to what has been produced by the simulator.

The next step is to reduce the expression to exclude all minterms which do not contribute much to the '*s*' coefficients of the transfer function. Section 3.3.4 provides a complete discussion of the algorithm used to reduce the expression.

Figure 22 Phase and magnitude Bode plots for the initial analysis of the amplifier



The 'rootlocus' routine can be used to view the root placement in the complex plane. Figure 23 is the resulting root placement graph from invoking the 'rootlocus' routine without sweeping the compensation resistor value ($r_{13} = 500 \Omega$). The poles are illustrated with the

'+' character, and the zeroes with the 'o' character. The X-axis is the real part, whereas the Y-axis is the imaginary part. Both axes are in logarithmic Hertz. Note how the dominant pole is located at about 100KHz. There are two right-hand plane zeroes located over 400MHz. These right-hand plane zeroes probably affect the phase margin of the amplifier because of their relatively low frequency with respect to the unity gain bandwidth. Moreover, there are non-dominant poles and zeroes in the neighborhood of 100MHz, which is about the unity-gain bandwidth of the amplifier, as seen in Figure 22 on page 102.

Figure 23 Root placement of the amplifier with initial values as shown in Figure 19

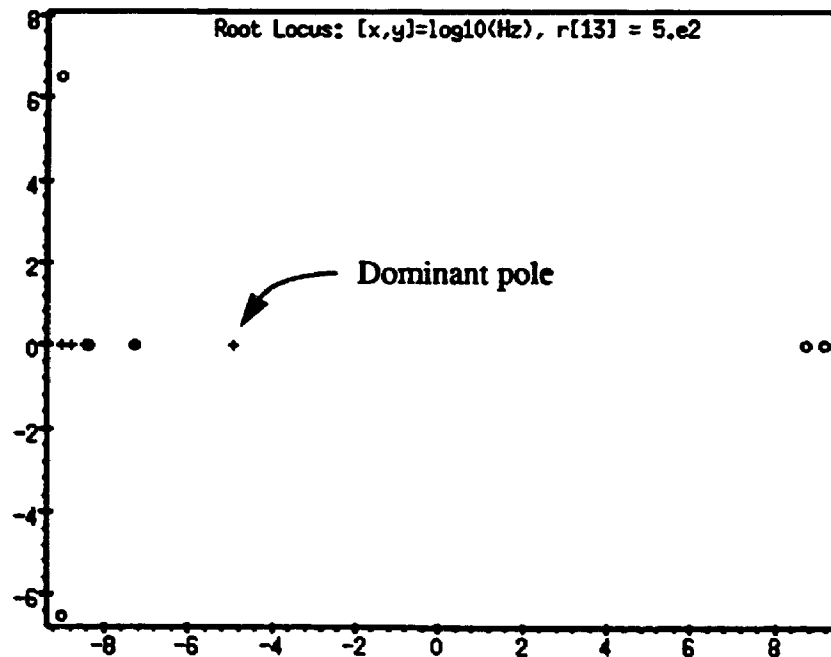


Figure 24 Root locations

Sensitivity Cutoff = 10.0 %

DC gain = 64.28 dB

Root Frequencies:

$$\left[z_1 = -202 \cdot 10^8, z_2 = -232 \cdot 10^9, z_3 = -509 \cdot 10^9, z_4 = -0.104 \cdot 10^{10} + 0.320 \cdot 10^7 i, z_5 = -0.104 \cdot 10^{10} - 0.320 \cdot 10^7 i, \right. \\ \left. z_6 = 0.134 \cdot 10^{10} \right]$$

$$\left[p_1 = -87000., p_2 = -0.197 \cdot 10^8, p_3 = -0.226 \cdot 10^9, p_4 = -0.285 \cdot 10^9, p_5 = -0.599 \cdot 10^9, p_6 = -0.106 \cdot 10^{10} \right]$$

Figure 25 Root and DC gain Sensitivities

Sensitivities:

```
table([
  c_gd9 = [.17 = p3, -.79 = p5]
  c_10 = [-.45 = z4, -.45 = z5, -.18 = z6, -.76 = p1, .28 = p3, .88 = p5, -1.4 = p6]
  r_13 = [-.78 = z4, -.78 = z5, .38 = z6, -.19 = p3, .17 = p4, 1.2 = p5, -2.2 = p6]
  gm_1 = [.47 = z4, .47 = z5, 1.2 = p3, -.76 = p4, .45 = p5]
  s_dsd = [-.12 = DCgain, .13 = p1, .17 = p3, -.17 = p4]
  s_dsg = [-.93 = DCgain, .90 = p1, .11 = p3, -.11 = p4]
  s_dsl = [-.85 = DCgain, .85 = p1, .15 = p3, -.15 = p4]
  c_gd9 = [-.44 = z6, -.18 = p1, -.55 = p3, .83 = p4, -.10 = p5, -.31 = p6]
  c_gd4 = [-.43 = z3, .33 = p3, -.41 = p6]
  gm_9 = [.99 = DCgain, -.23 = z4, -.23 = z5, 1.4 = z6, -.93 = p1, .13 = p3, .32 = p4, 1.0 = p5, -.61 = p6]
  c_gdl = [-.22 = p3, .27 = p4, -.12 = p6]
  c_gsl = [-.52 = z4, -.52 = z5, -.43 = p3, .23 = p4, -.41 = p5, -.22 = p6]
  c_id = [.10 = p4, -.65 = p5, -.18 = p6]
  gm_4 = [1.0 = DCgain, .82 = z2, 1.0 = z3, .12 = p2, .50 = p3, .69 = p4, -.54 = p5, .15 = p6]
])
```

In order to understand how to properly place the roots in the complex plane, a sensitivity analysis is performed. By invoking the 'print_sensitivities' routine, the user can get the

normalized symbolic sensitivities of each root with respect to each symbolic parameter. Since the routine needs to compute the symbolic partial differentials for each root, the sensitivity analysis is performed on the reduced expression, in order to save on CPU resources. Only parameters which affect the roots by more than 10% are shown. Figure 24 shows the location of the roots, along with the DC gain, while Figure 25 shows how each symbolic parameter can affect the roots and the DC gain.

The theory of the two-stage pole splitting operational amplifier tells us that the Miller compensation zero is to be affected by the feedforward resistance, r_{13} . A close examination of the sensitivities of the roots for r_{13} does not reveal the Miller zero in an obvious fashion. The sensitivity numbers are normalized, which means that a 1% increase in r_{13} will result in a 0.78% decrease in the location of both zeros of the complex zero pair, z_4 and z_5 . The negative sign implies that the direction of movement is opposite that of the symbolic parameter. Since the Miller zero cannot be identified by the sensitivity analysis, a root locus analysis, where r_{13} is the parameter swept over a wide range, should make it clear which zero is affected by the Miller effect. Figure 26 illustrates the movements of the roots for values of r_{13} in the range {500, 665, 3120}. Notice how the complex zero pair, at $r_{13} = 500$ Ohms, becomes real for larger values of r_{13} . Also, one of the two zeros of the complex pair effectively becomes the Miller zero as it moves to a lower frequency, and consequently enhances the phase margin of the amplifier. This phase enhancement is better seen in Figure 27, where a three dimensional magnitude and phase Bode plots is illustrated. This unique Bode plot is parametric, where the parameter is r_{13} , swept over the range {500..5000} Ohms. The magnitude plot clearly demonstrates the 'bump' near the

unity-gain frequency created by the zero. The phase plot also shows the phase enhancement expected from a lower frequency zero.

These observations indicate that with $r_{13} = 500$ Ohms, the Miller zero is located at a high frequency, not useful to help the phase margin. Consequently, the analysis shall continue with a higher value for r_{13} , which will place the Miller zero in the neighborhood where it improves stability. A value of $r_{13} = 1200$ Ohms has been chosen to meet this purpose. Since r_{13} has now changed, the sensitivity analysis needs to be recalculated. Figure 28 shows the results of the root location and the root sensitivities to the symbolic parameters for a value of $r_{13} = 1200$ Ohms.

A close look at Figure 28 reveals that the Miller zero is z_3 , and its sensitivity to r_{13} is -1.4. This value implies that a 1% increase in r_{13} will lead to a 1.4% decrease in the location of the zero. The negative sign means that the zero decreases in frequency as r_{13} is increased.

Note that c_{10} can also be increased to improve the stability as it brings z_3 down, but it also brings down the dominant pole (p_1), an undesirable situation as this would lower the unity-gain bandwidth. Also note that a doublet (p_2 and z_1) exists around 20 MHz. This doublet is ignored in the subsequent analysis as it is not affected by any symbolic parameter. Also of importance, note that the complex pole pair moves down with an increasing r_{13} , although not as quickly as z_3 . Consequently, it may be important to resize the active devices of the circuit to push this complex pole pair to a higher frequency.

Figure 26 Root loci for values of $r_{13} = \{500, 665, 3120\}$ Ohms

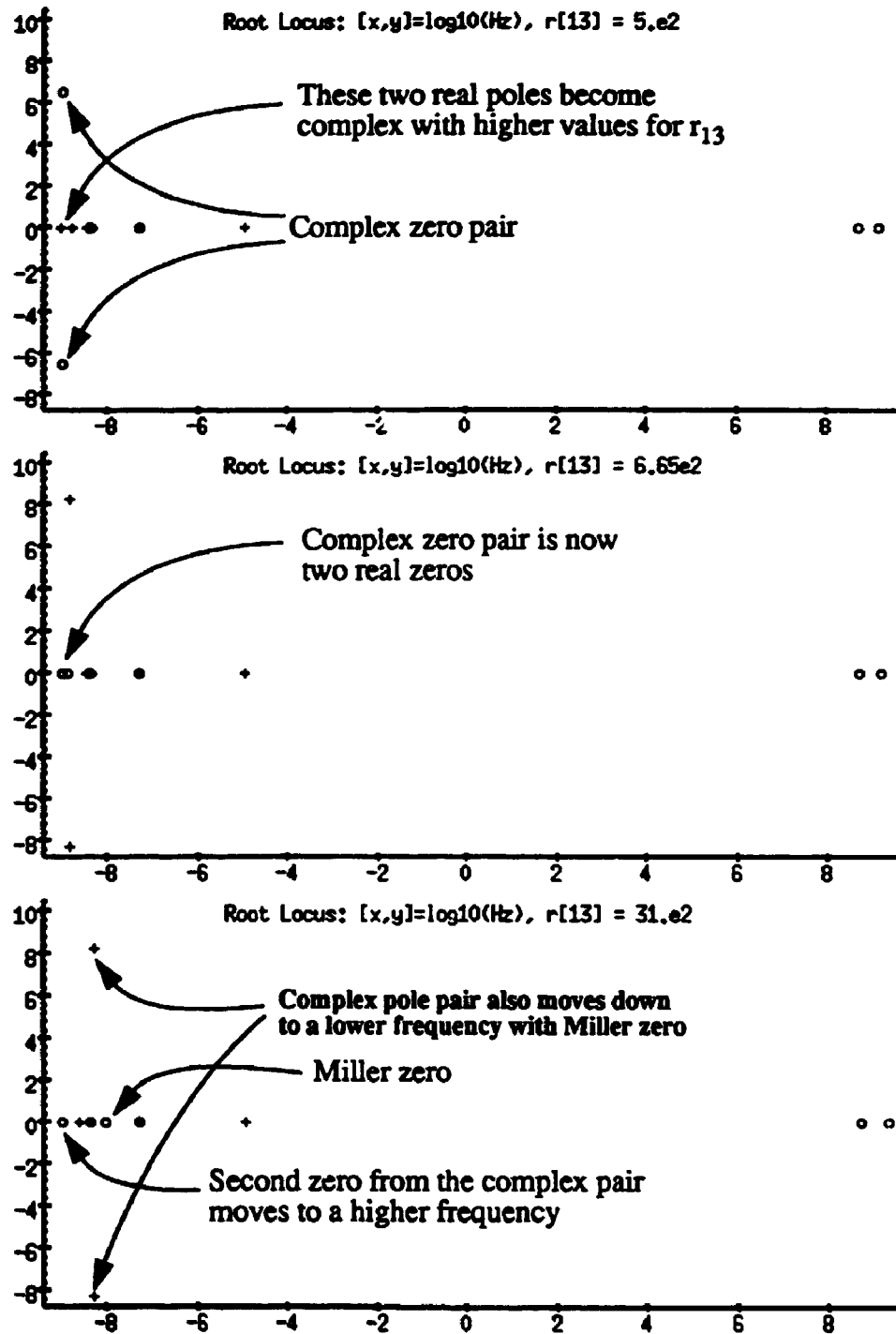


Figure 27 Parametric Bode plots. Parameter $r[13] = (500..5000)$

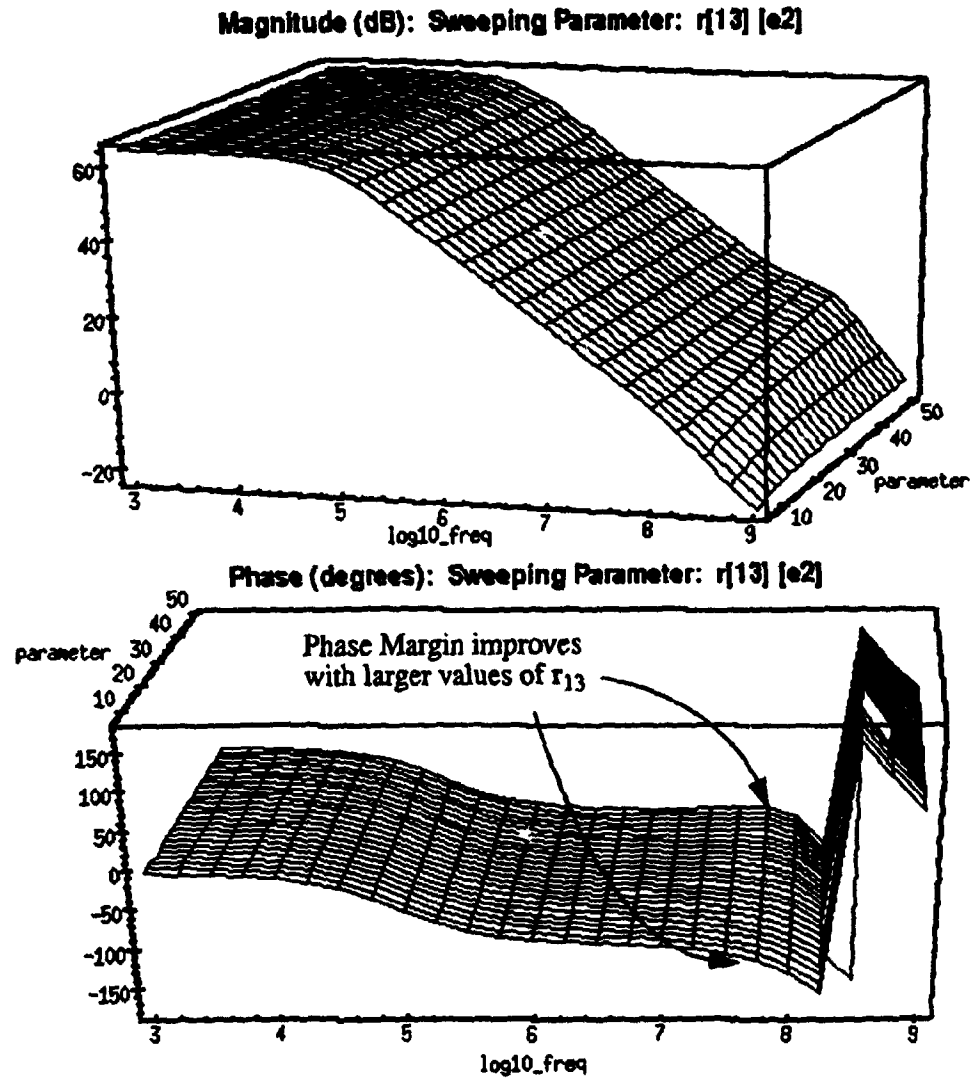


Figure 28 Root and DC gain locations and parametric sensitivities for $r_{13} = 1200$

Sensitivity Cutoff = 10.0 %

DC gain = 64.28 dB

Root Frequencies:

=====

$$\left[z_1 = -202 \cdot 10^8, z_2 = -240 \cdot 10^9, z_3 = -324 \cdot 10^9, z_4 = .514 \cdot 10^9, z_5 = -.100 \cdot 10^{10}, z_6 = .176 \cdot 10^{10} \right]$$

$$\left[p_1 = -87000, p_2 = -.199 \cdot 10^8, p_3 = -.194 \cdot 10^9, p_4 = -.313 \cdot 10^9, p_5 = -.363 \cdot 10^9 + .385 \cdot 10^9 i, \right.$$

$$\left. p_6 = -.363 \cdot 10^9 - .385 \cdot 10^9 i \right]$$

Sensitivities:

=====

```
table([
  c10 = [.11 = z2, -1.1 = z3, -.76 = p1, .15 = p3, -.22 = p5, -.22 = p6]
  r13 = [.16 = z2, -1.4 = z3, .23 = z6, -.18 = p3, -.45 = p5, -.45 = p6]
  gm9 = [.99 = DCgain, -.16 = z3, 1.2 = z6, -.93 = p1, .24 = p3, .29 = p5, .29 = p6]
  cgs1 = [-1.2 = z5, -.20 = p3, -.24 = p4, -.20 = p5, -.20 = p6]
  cid = [.10 = p4, -.41 = p5, -.41 = p6]
  sds9 = [-.93 = DCgain, .90 = p1]
  cds4 = [-.43 = z4, .10 = p2, .18 = p3, -.19 = p5, -.19 = p6]
  gm4 = [1.0 = DCgain, .84 = z2, 1.0 = z4, .12 = p2, .42 = p3, .51 = p4]
  sds4 = [-.12 = DCgain, .13 = p1]
  cgs9 = [-.40 = p5, -.40 = p6]
  cds9 = [-.53 = z6, -.18 = p1, -.29 = p3, .41 = p4, -.13 = p5, -.13 = p6]
  sds1 = [-.85 = DCgain, .85 = p1]
  gm1 = [-.13 = z2, .11 = z3, .99 = z5, .55 = p3, .30 = p4]
])
```

At this point, it becomes necessary to calculate the sensitivities of the roots with respect to the geometry of the active devices. Three separate Eldo simulations are required to achieve this. The first simulation is used to extract the values of the symbolic parameters

when the lengths of all transistors are increased by 10%. The second simulation is used to extract the values of the symbolic parameters when the widths of all transistors are increased by 10% while the third simulation is used to extract the values of the symbolic parameters when both the lengths and the widths of all transistors are increased simultaneously by 10%. The variation of the symbolic parameters, combined with the nominal sensitivities of the roots with respect to each symbolic parameter, provides all the information necessary to calculate the net sensitivity of the roots with respect to transistor geometry, that is, either the length alone, the width alone, or the length and width simultaneously. Note that this method is accurate only when all symbolic parameters of the small signal model of the transistor are included in the calculation. Since this is not the case in the current case study, the results are accordingly approximate. For transistor m1 and m9, only parameters in the set $\{g_{ds}, g_m, c_{gd}, c_{gs}\}$ have been included in the calculations, whereas the set $\{g_{ds}, g_m, c_{gs}\}$ has been used for m4. Since these parameters correspond to a simplified small signal model, the results of the analysis can still be considered as reasonably accurate for our purposes.

Figure 29 shows the root sensitivities to the lengths and widths of the important transistors of the circuit. Figure 30 illustrates the sensitivities to the size (correlated lengths and widths) of the important transistors of the circuit.

Figure 29 Sensitivities to the length and width when r13 = 1200

Sensitivities for parameter: l

=====

$$\left[\begin{array}{l} 03831 = c_{gd4} - 4598 = gm_4 - 2.492 = g_{ds4} \quad 9729 = c_{gs9} \quad 65344 = c_{gd9} - 1916 = g_{ds1} \quad .08264 = c_{gd1} \\ -4510 = gm_1, 0 = c_{10}, 0 = r_{13}, -7603 = gm_9, 0 = c_{12} - 2.312 = g_{ds9} \quad 1.024 = c_{gs1} \end{array} \right]$$

MOS1 sensitivities using parameters, $\{g_{ds1}, c_{gd1}, gm_1, c_{gs1}\}$, are:

$$[-1.6 = p_1, -58 = p_3, -30 = p_4, -1.6 = z_5, 1.6 = DCgain, -21 = p_5, -21 = p_6]$$

MOS4 sensitivities using parameters, $\{c_{gd4}, gm_4, g_{ds4}\}$, are:

$$[-32 = p_1, -37 = p_3, -10 = p_4, -39 = z_2, -50 = z_4, -16 = DCgain]$$

MOS9 sensitivities using parameters, $\{c_{gs9}, c_{gd9}, gm_9, g_{ds9}\}$, are:

$$[-1.4 = p_1, -26 = p_3, -96 = z_6, .11 = z_3, 1.4 = DCgain, -61 = p_5, -61 = p_6]$$

Sensitivities for parameter: w

=====

$$\left[\begin{array}{l} 1.002 = c_{gd4} \quad 4264 = gm_4 \quad .2809 = g_{ds4} \quad 9246 = c_{gs9} \quad 1.000 = c_{gd9} \quad 3345 = g_{ds1} \quad 9986 = c_{gd1} \quad .4108 = gm_1 \\ 0 = c_{10}, 0 = r_{13}, .4819 = gm_9, 0 = c_{12} \quad .4365 = g_{ds9} \quad 9104 = c_{gs1} \end{array} \right]$$

MOS1 sensitivities using parameters, $\{g_{ds1}, c_{gd1}, gm_1, c_{gs1}\}$, are:

$$[.29 = p_1, -.12 = p_4, -.65 = z_5, -.27 = DCgain, -.18 = p_5, -.18 = p_6]$$

MOS4 sensitivities using parameters, $\{c_{gd4}, gm_4, g_{ds4}\}$, are:

$$[.15 = p_2, .38 = p_3, .22 = p_4, .29 = z_2, .10 = z_3, .41 = DCgain, -.22 = p_5, -.22 = p_6]$$

MOS9 sensitivities using parameters, $\{c_{gs9}, c_{gd9}, gm_9, g_{ds9}\}$, are:

$$[-.25 = p_1, -.11 = p_3, .42 = p_4, -.13 = z_3, -.36 = p_5, -.36 = p_6]$$

Figure 30 Sensitivities to the size (correlated length and width) when $r_{13} = 1200$ Sensitivities for parameter: lw

$$\left[\begin{array}{l} 1.043 = c_{gd4} \quad -0.04831 = g_{m4} \quad -2.302 = g_{ds4} \quad 1.987 = c_{gs9} \quad 1.061 = c_{gd9} \quad -1.648 = g_{ds1} \quad 1.088 = c_{gd1} \\ -0.05601 = g_{m1} \quad 0 = c_{10} \quad 0 = r_{13} \quad -3263 = g_{m9} \quad 0 = c_{12} \quad -1.993 = g_{ds9} \quad 2.026 = c_{gs1} \end{array} \right]$$

MOS 1 sensitivities using parameters, $\{g_{ds1}, c_{gd1}, g_{m1}, c_{gs1}\}$, are:

$$[-1.4 = p_1, -62 = p_3, -44 = p_4, -2.4 = z_5, 1.4 = DCgain, -41 = p_5, -41 = p_6]$$

MOS 4 sensitivities using parameters, $\{c_{gd4}, g_{m4}, g_{ds4}\}$, are:

$$[-28 = p_1, .11 = p_4, -.11 = z_2, -.52 = z_4, 24 = DCgain, -19 = p_5, -19 = p_6]$$

MOS 9 sensitivities using parameters, $\{c_{gs9}, c_{gd9}, g_{m9}, g_{ds9}\}$, are:

$$[-1.7 = p_1, -.39 = p_3, .48 = p_4, -.95 = z_6, 1.5 = DCgain, -1.0 = p_5, -1.0 = p_6]$$

By looking closely at the results shown in Figure 29 and Figure 30, the following conclusions can be made about transistors m1, m4 and m9:

1. The width of m4 should be increased. This will result in a higher DC gain, and it will push the first and second non-dominant poles (p_3 and p_4) to a higher frequency. However, beware of the complex pole pair (p_5 and p_6) coming down in frequency. The sensitivities clearly demonstrate this in Figure 29.
2. The length of m4 should be reduced. This will push the dominant pole (p_1) and the first and second non-dominant poles (p_3 and p_4) to a somewhat higher frequency, which improves the bandwidth of the amplifier. Stability is not affected much since z_2 also moves up. Again, Figure 29 shows these relationships.
3. The length of m1 should be increased. This will improve the DC gain, while pushing the dominant pole p_1 down. As a consequence, the unity-gain bandwidth is not affected. The first non-dominant pole (p_3) will also come down, which effectively counteracts the increase created by reducing the length of m4 in step 2. The end result of these steps is a higher DC gain.

4. The length of m_9 should be kept small, within reasons. This will reduce the already improved DC gain, and trade it off with a better stability, as the complex pole pair (p_5 and p_6) is pushed to a higher frequency. A shorter length implies less DC gain, but it pushes the dominant pole higher, thus resulting in an unchanged unity-gain bandwidth. Since the complex pole pair is pushed to a higher frequency, the Miller zero is thus capable to maintain a good phase margin up to a higher frequency, thus resulting in improved stability.
5. The final step would then be to adjust r_{13} such that the phase and the gain margins are optimal. This is because as seen in Figure 28, the value of r_{13} affects mostly the Miller zero (z_3) and the complex pole pair (p_5 and p_6). Hence, the active components can first be optimized to place the roots in the best possible location, and then the Miller zero can be placed properly to improve the stability. The general idea is to first try to optimize the active components to place the complex pole pair at the highest frequency possible, and then adjust the placement of the Miller zero over all simulation corners, by tuning the value of r_{13} .

It is important to realize that normally the exploration of the design space would be done via several sets of simulations, and the resulting collection of plots would be used to help the designer understand the relationships between the various roots of the amplifier, and how each root is affected by the small signal parameters and the physical design parameters. Moreover, in order to properly interpret such plots, the designer would have needed to do a fair bit of preliminary analysis to understand the theory behind the movement of the roots. This often translates itself into a long and tedious exercise, especially for designers with limited experience.

The traditional design process thus puts emphasis on the designer's ability to relate such theories to the actual frequency domain plots, and also to relate the conclusions reached to the physical size of the transistors. This is a difficult task that only designers with many

years of relevant experience can do efficiently. Using the Maple-Eldo tool, however, allows the designer to easily reach the same or better conclusions through close examination of the symbolic and device size sensitivities. The root locus plots, along with the 3D Bode plots also allow the designer to visualize the impact of specific symbolic parameters, and how they affect the overall amplifier frequency response. And finally, the Maple worksheet provides an easy medium to capture the results of these analyses, along with comments from the designer, in a chronological order.

4.2.2 Root placement optimization and design centering

The rules stated in the previous section were used by the author to optimize the amplifier under nominal conditions. Once the specifications were met under nominal conditions, the '*autopvit*' script was invoked. As discussed in section 3.5 on page 87, the '*autopvit*' script is used to simulate the design over a number of simulation corners. In this particular case, the following corners were chosen to be:

- Voltage from 4.5 to 5.5 volts
- Current bias from 45 to 55 μ A.
- Best and Worst process technology for both NMOS and PMOS independently
- Temperature from -40 to 105 degrees Celsius
- Capacitor variation from 80% to 120%
- Resistor variation from 70% to 130%

These seven parameters produced a total of $2^7 = 128$ simulations. A report was then produced containing the important data from these simulations. The phase margin, the DC gain, the gain margin and the unity-gain bandwidth were extracted for each simulation corner. The results were then examined carefully by the author, and the device sizes were

further tuned according to the observations and bearing in mind the rules stated in the previous section. The final netlist obtained was compared to the initial netlist, and the component values for both netlists are shown in Table 1.

Table 1 Initial netlist compared to the final netlist

Device name	Initial sizes	Final sizes
R13	500 Ω	1500 Ω
C1d	0.5 pF	0.5 pF
C10	0.5 pF	0.5 pF
M8	L=3 μm , W=150 μm	L=3 μm , W=150 μm
M7	L=3 μm , W=25 μm	L=3 μm , W=25 μm
M5	L=3 μm , W=75 μm	L=3 μm , W=75 μm
M4	L=2 μm , W=200 μm	L=1 μm , W=450 μm
M3	L=2 μm , W=200 μm	L=1 μm , W=450 μm
M9	L=1.5 μm , W=300 μm	L=1 μm , W=200 μm
M2	L=1.5 μm , W=75 μm	L=3 μm , W=150 μm
M1	L=1.5 μm , W=75 μm	L=3 μm , W=150 μm

The final netlist shown in the table meets the original set of specifications over all simulation corners. The full final report produced by 'autopvit' is listed in APPENDIX D.

4.2.3 Conclusions

The role of the important roots present in the circuit became obvious to the designer through the usage of the parametric Bode plots, the sensitivity analyses, and the root locus

parametric plots. For instance, the relationships between the device sizes, the Miller zero, and the complex pole pair was clearly identified, and a set of design rules was easy to derive from observations of the Maple analysis results. Moreover, this information was captured in the form of an executable worksheet, so the knowledge gained can be easily reused by another designer at a later time. Although excerpts of the worksheet were shown in several figures in this chapter, a complete listing of the worksheet including designer's comments is provided in APPENDIX B for reference purposes.

It is concluded that the two important goals of this work have been achieved. Firstly, the Maple-Eldo prototype has helped the designer to quickly master the important small signal design trade offs of the circuit in an efficient manner, thus meeting the requirement for efficiently increasing the designer's understanding of the circuit. Secondly, the worksheet provides a recorded log of the designer's efforts, postulates, observations, and conclusions. These recorded details can later be reused by a different designer, should the circuit be re-optimized for a different application. In essence, the design knowledge has been captured through the recorded log and is fully reusable and transferable. Thus, the goals of design knowledge development, capture and reuse have been achieved.

5.0 Evaluations and Conclusions

5.1 Summary and discussions

The business costs related to productivity, predictability and design quality were discussed. This provided the motivation for finding a design methodology which would allow for a faster and more efficient design knowledge development, capture, and reuse approach.

A relatively simple model for describing analog circuit design knowledge was defined. The traditional approach using SPICE was described and its limitations were examined. Some of the major limitations discussed include:

- No direct insight into the circuit is provided using SPICE.
- The link between theory and observations remains very much a manual task to be performed by the designer.
- The SPICE stand-alone approach does not provide any sense of history, let alone a method to capture and transfer the design knowledge for later reuse.

It was proposed that knowledge reuse and transfer could be achieved through a framework that allows the designer to better understand how the circuit works, and to capture his reasoning while exploring the design space.

The existing literature addressing the topics of analog circuit knowledge development and reuse was reviewed and criticized. The existing work was found to fall under either the optimizer-based approach, or under the spreadsheet oriented *a priori* knowledge development method. The major drawback of both approaches was found to be that they hide the design knowledge from the user of the tool during the reuse phase. Hence, design knowl-

edge can not be transferred *per se*. Moreover, no means of improving productivity were given for the approach requiring *a priori* knowledge development.

The proposed method was exposed as an attempt to address these issues for the special case of small signal analysis. The goal was to provide better insights into the circuit, and to identify a framework to help capture, reuse and transfer the design knowledge developed by the designer. This was believed to be achievable provided the tool had the following features:

- The ability to navigate within the mathematical and physical design space,
- The ability to filter out the irrelevant details,
- The ability to capture the design knowledge as it is developed, during design space exploration,
- A systematic, flexible, error-free, and physically accurate method,
- A user-friendly way to edit the information captured,
- A simple and efficient way to link to a SPICE-like circuit simulator.

A prototype based on the Maple V symbolic analysis engine was developed in order to prove the method. The prototype was described in its entirety, including its UNIX-based link to Eldo, a SPICE-like circuit simulator. Important issues such as small signal modeling were exposed, since the design knowledge developed can only be as good as the accuracy of the analyses. Some of the major limitations of the tool were exposed, namely, the computing limitations common to symbolic circuit analysis.

A case study was performed to assess the value that the prototype added to an overall operational amplifier design project. It was found that the tool made it much easier on the designer to visualize the performance of the amplifier in terms of its key design parameters,

such as the transistor sizes and the values of the various small signal parameters. Moreover, since the tool provides an easy means (via the Maple worksheets) to capture the circuit design knowledge as it is developed, it is anticipated that this knowledge could be reused efficiently at a later time by a fellow engineer in a subsequent project.

One of the most important advantage of the tool is that it allows the designer to navigate at will within the small signal design space, in order to fully visualize any possible relationship between the physical and small signal parameters. The designer should therefore be in a better capacity to quickly assess any potential limitations of the circuit, and thus to be in a better position to understand the technical or scheduling risk of the design project. As a result, the circuit design project can be more predictable than it would be otherwise.

5.2 Recommendations for future work

There are many topics which deserve special attention in any subsequent work following this thesis. For instance, it would be useful to find a faster and more memory efficient algorithm for finding the symbolic solution of a large circuit matrix. This would benefit the designer in the sense that large analog circuits could be solved efficiently.

The tool is currently limited by Maple's crude word processing capabilities, its lack of user-made graphics (such as a circuit schematic), and its lack of integration into other simulation tools. For instance, it would be much better, for documentation and viewing purpose, if the designer could paste the results of external simulations into the worksheet, thus creating the ability to make comparisons in the worksheet between the Maple results and the simulated results.

Another aspect would be to make the '*maplenet*' program hierarchical, so that hierarchical netlists could be read by the program, and the equivalent MNA matrix could be built from it. Moreover, the current prototype has models only for MOS transistors. In a BiCMOS process, a model for bipolar transistors might also be useful. Similarly, models for capacitors and resistors might be useful, especially in the case where the distributed capacitance of a diffused resistor affects the performance of the circuit.

The '*smallsignal*' package itself could be improved much further. For example, a data structure could be added in '*lookup_min*' and '*lookup_max*' files which stores the simulation run number for the minimum and maximum value of each parameter. Ideally, the simulation number would also be cross-referenced to the actual simulation corner parameters, so that the designer would know which corner parameters generated the minimum and maximum values for the small signal parameter.

The '*reduce_order*' routine could be improved with an algorithm that computes the sensitivity of each symbolic parameter with respect to each of the '*s*' coefficients, for both the reduced coefficients and the exact coefficients. Thus, each exact sensitivity could be compared with its corresponding approximated one in order to maintain the accuracy of the transfer function and its first order partial differentials. This therefore implies that the root sensitivities of the reduced transfer function will be similar to the root sensitivities of the original transfer function, and thus accuracy can be guaranteed in the sensitivity analyses.

The '*deltagen*' routine could be extended with the infrastructure to compute the sensitivity of other physical quantities such as temperature, bias current conditions, voltage or process technology corner. This would provide the designer with a clearer understanding of

the effects of the simulation corners on the design. In a similar light, an algorithm should be found to determine the sensitivity of each root with respect to each simulation corner, as well as the correlation between the movement of the roots with respect to one another. Also, the current approach of finding sensitivities using partials and perturbations could be improved by implementing the algorithm based on the adjoint systems. This is described in reference [25].

A set of routines that can extract such information as gain and phase margins from the given transfer function could be useful in the assessment of the impact of a change in the circuit. In general, the tool should provide more infrastructure supporting the design of analog circuit using feedback, such as phase-locked loops, voltage regulators and amplifiers.

The possibility to extend this work into full large signal analysis should be examined. One concept worth investigating is the possibility of modelling the large signal domain in terms of a set of piecewise linear lookup tables of various DC points. Again, the movement of the roots, and their respective sensitivities could be computed as the DC point is changed. Although this concept could be useful for the analysis of non linear oscillator circuits, it still does not preclude the use of a full transient analysis on a simulator.

A set of support routines to analyze the linearized circuit in the time domain should also be added to the '*smallsignal*' package. This would allow the designer to relate such quantities as the settling time and settling accuracy to the overall location of the frequency domain roots.

As a general rule, the research and development focus should be on methods that provide the designer with a better interface between Maple and the design framework used during design.

6.0 REFERENCES

- [1] E. Girczyc, S. Carlson, Increasing Design Quality and Engineering Productivity through Design Reuse, 30th ACM/IEEE Design Automation Conference, 1993.
- [2] R.A. Rutenbar, Analog Design Automation: Where are we? Where are we going?, IEEE 1993 Custom Integrated Circuits Conference.
- [3] B. Preas, P. Karger, Automatic Placement: A Review of Current Techniques, Proc. DAC, June 1986.
- [4] E.S. Kuh, T. Ohtsuki, Recent Advances in VLSI Layout, Proc. IEEE, Vol. 78, No. 2, Feb. 1990.
- [5] R.K. Brayton, et al., Multi-Level Logic Synthesis, Proc. IEEE, Vol. 78, No. 2, Feb. 1990.
- [6] M.G.R. Degrauwe, et al., IDAC: An interactive design tool for analog CMOS circuits, IEEE Journal of Solid-State Circuits, Vol. 22, pp. 1106-1114, Dec. 1987.
- [7] R. Harjani, R. Rutenbar, L.R. Carley, OASYS: A framework for analog circuits synthesis, IEEE Trans. on CAD, Vol. 8, pp. 1247-1265, Dec. 1989.
- [8] T. Morie, H. Onodera, K. Tamaru, A System for Analog Circuit Design that Stores and Reuses Design Procedures, Proc. IEEE Custom Integrated Circuits Conference, pp. 13.4.1-13.4.4, 1993.
- [9] R.K. Henderson, et al., A Spreadsheet Interface for Analog Design Knowledge Capture and Reuse, Proc. IEEE Custom Integrated Circuits Conference, pp. 13.3.1-13.3.4, 1993.
- [10] F. Medeiro, et al., Global design of analog cells using statistical optimization techniques, Analog Integrated Circuit and Signal Processing, Vol. 6, No. 3, pp. 179-195, Kluwer, Nov. 1994.

- [11] F.V. Fernandez, et al., Interactive AC modeling and characterization of analog circuits via symbolic analysis, *Analog Integrated Circuit and Signal Processing*, Vol. 1, No. 3, pp. 183-208, Kluwer, Nov. 1991.
- [12] K.K. Wee, R.J. Mack, Towards expandable and generalised analogue design automation, *Proceedings ISCAS '94*, Vol. 1, pp. 359-362.
- [13] D.A. Pintur, *MathEdge: The application development toolkit for Maple*, Waterloo Maple Software, Waterloo, Ontario, Canada.
- [14] F.V. Fernandez et al., Symbolic analysis of large analog integrated circuits by approximation during expression generation, *Proceedings ISCAS '94*, vol. 1, pp. 25-28.
- [15] B.J. Hosticka, et al., Design Methodology for Analog Monolithic Circuits, *IEEE Transactions on Circuits and Systems - I: Fundamental Theory and Applications*. Vol. 41, No. 5, May 1994.
- [16] H.Y. Koh, et al., OPASYN: A compiler for MOS operational amplifiers, *IEEE Transactions on CAD*, vol. 9, no. 2, February 1990.
- [17] J. Jongsma et al., An open design tool for analog circuits, *Proceedings IEEE ISCAS 1991*, pp. 2000-2003.
- [18] G. Gielen, and W. Sansen, *Symbolic analysis for automated design of analog integrated circuits*, Kluwer, Boston, 1991.
- [19] C. Meixenberger, et al., Tools for analog design, *Proc. Workshop on Advances in Analog Circuits Design*, pp. 357-368, Scheveningen, The Netherlands, 1992.
- [20] C. Michael, and M. Ismail, Statistical modelling of device mismatch for analog MOS integrated circuits, *IEEE Journal of Solid State Circuits*, vol. 27, pp. 154-166, Feb. 1992.

- [21]** H. N. Gabow, Two algorithms for generating weighted spanning trees in order, *SIAM J. of Computing*, vol. 6, no. 1, pp. 139-150, March 1977.
- [22]** F. V. Fernandez et al., Accurate simplification of large symbolic formulae, *Proc. IEEE ICCAD*, pp. 318-321, Nov. 1992.
- [23]** S. J. Seda, M. G. R. DeGrauwe, and W. Fichtner, Lazy-expansion symbolic expression approximation in SYNAP, *Proc. IEEE ICCAD*, pp. 310-317, Nov. 1992.
- [24]** R. Gregorian, and G. C. Temes, *Analog MOS integrated circuits for signal processing*, John Wiley & Sons, 1986.
- [25]** J. Vlach, K. Singhal, *Computer Methods for Circuit Analysis and Design*, Second Edition, Van Nostrand Reinhold, New York, 1994.
- [26]** P. V. O'Neil, *Advanced Engineering Mathematics*, Second Edition, Wadsworth Publishing Company, Belmont, California, 1987.
- [27]** A. S. Sedra, K. C. Smith, *Microelectronic Circuits*, Second Edition, Holt, Rinehart and Winston, New York, 1987.
- [28]** P. R. Gray, R. G. Meyer, *Analysis and Design of Analog Integrated Circuits*, Second Edition, John Wiley & Sons, New York, 1984.
- [29]** P. R. Gray, R. G. Meyer, MOS Operational Amplifier Design - A Tutorial Overview, *IEEE Journal of Solid State Circuits*, vol. SC-17, no.6, pp. 969-982, Dec. 1982.

APPENDIX A Examples of Important Files

A short example of the netlist used by the Maple-based prototype tool is given. This is the netlist used in the case study of section 4.0 on page 91.

A.1 Transistor-level netlist: 'netlist'

```

.....
*
* Netlist for Analysis 1: Two-stage OTA
*
.....
Rxx13 vc1 vc2 500
Cxx14 avss vout 0.5P
Cxx10 vc2 vout 0.5P
Mxm8 vout vbias avdd avdd MPCH L=3U W=150U
Mxm7 vbias vbias avdd avdd MPCH L=3U W=25U
Mxm5 vx vbias avdd avdd MPCH L=3U W=75U
Mxm6 vc1 p vx vx MPCH L=2.0U W=200U
Mxm3 vn n vx vx MPCH L=2.0U W=200U
Mxm9 vout vc1 avss gnd MNCH L=1.5U W=300U
Mxm2 vn vn avss gnd MNCH L=1.5U W=75U
Mxm1 vc1 vn avss gnd MNCH L=1.5U W=75U
*
.....
* EXTERNAL MODEL REFERENCES:
*
*   MNCH  MPCH
*
.....

```

A.2 User-modified 'netlist.ss'

```

=====
*
* This is the header used for the netlist.ss created by
* createnet and createnet?.awk.
*
* It contains the excitation source(s) used for Maple.
*
=====
input p n l

=====
*
* Note that you must make some effort to reduce the
* number of nodes in your netlist before attempting
* to solve the resulting matrix with Maple. The higher
* the number of nodes, the more difficult it is for Maple

```

- * to solve the system. As a simple rule-of-thumb, the
- * user can get rid of the source series resistance as this
- * generally makes little difference on the transfer function,
- * but it makes it enormously difficult for Maple to solve.
- *
- * Also, keep the number of symbolic variables to within reason.
- *

```
*****
* resistor: rrx13
*****
r13    vc1  vc2  name
```

```
*****
* capacitor: cxcld
*****
cld    gnd  vout  name
```

```
*****
* capacitor: cxc10
*****
c10    vc2  vout  name
```

```
*****
* mos: mzm8  model: mpch
*****
gm8    vbias  gnd  vout  gnd      srvalue
gds8   vout   gnd                srvalue
cgs8   vbias  gnd                srvalue
cgs8   vbias  vout            srvalue
cgb8   vbias  gnd                srvalue
cbd8   gnd    vout            srvalue

mcdg8  vbias  gnd  vout  gnd      srvalue
```

```
*****
* mos: mzm7  model: mpch
*****
gm7    vbias  gnd  vbias  gnd      srvalue
gds7   vbias  gnd                srvalue
cgs7   vbias  gnd                srvalue
cgb7   vbias  gnd                srvalue
cbd7   gnd    vbias            srvalue

mcdg7  vbias  gnd  vbias  gnd      srvalue
```

* mos: mzm5 model: mpch

gm5 vbias gnd vx gnd srvalue

gds5 vx gnd srvalue

cgs5 vbias gnd srvalue

cgd5 vbias vx srvalue

cgb5 vbias gnd srvalue

cbd5 gnd vx srvalue

mcdg5 vbias gnd vx gnd srvalue

* mos: mzm6 model: mpch

gm6 p vx vcl vx name

gds6 vcl vx name

cgs6 p vx srvalue

cgd6 p vcl name

cgb6 p vx srvalue

cbd6 vx vcl srvalue

mcdg6 p vx vcl vx srvalue

* mos: mzm3 model: mpch

gm3 n vx vm vx gm6

gds3 vm vx gds6

cgs3 n vx srvalue

cgd3 n vm cgd6

cgb3 n vx srvalue

cbd3 vx vm srvalue

mcdg3 n vx vm vx srvalue

* mos: mzm9 model: mnch

gm9 vcl gnd vout gnd gm9

gds9 vout gnd gds9

cgs9 vcl gnd cgs9

cgd9 vcl vout cgd9

cgb9 vcl gnd srvalue

cbd9 gnd vout srvalue

mcdg9 vcl gnd vout gnd srvalue

* mos: mzm2 model: mnch

```

*****
gm2      vm  gnd  vm  gnd      gm1
gds2     vm  gnd                gds1
cgs2     vm  gnd                cgs1
cgb2     vm  gnd                ssvalue
cbd2     gnd  vm                ssvalue

mcdg2    vm  gnd  vm  gnd      ssvalue
*****
* mos: mzm1  model: mnch
*****
gm1      vm  gnd  vcl  gnd      name
gds1     vcl  gnd                name
cgs1     vm  gnd                name
cgd1     vm  vcl                name
cgb1     vm  gnd                ssvalue
cbd1     gnd  vcl                ssvalue

mcdg1    vm  gnd  vcl  gnd      ssvalue

```

APPENDIX B Maple worksheet

This is a listing of the Maple worksheet developed in the case study of section 4.0 on page 91.

analysis1.ms

This worksheet performs the initial analysis of the amplifier. The following three operations are performed in order:

1. The circuit matrix is first solved, using `mnsolve`. The solution is graphically compared with the simulator's results using bode plots, as a sanity check.
2. The resulting transfer function is reduced, in order to facilitate the computation of symbolic sensitivities.
3. Conclusions are drawn from the sensitivity analyses. These will help the designer to understand the numerous trade offs relating the important small signal parameters.

NOTE: Please refer to the discussion on each routine in the thesis document for a complete explanation of each routine, and the meaning of their arguments.

Large Signal Analysis of the Amplifier:

Before performing the small signal analysis on Maple, we outline the specifications this amplifier is to be designed for:

- Minimum unity gain bandwidth: 100MHz
- Maximum time allowed for slewing: 20 ns
- Time allowed for linear settling: 80 ns
- Output capacitive load: 0.5pF
- Minimum DC gain: 60dB

This amplifier is designed for a switched capacitor application, where the sampling rate is at 5 MHz. Thus, one complete cycle is 200ns, implying that half of this cycle (100ns) is for the integration phase, whereas the other half cycle is the reset phase. Hence, if 20 ns is the maximum slewing time allowed, 80 ns remains to achieve linear settling.

The amplifier is of the basic two stage OTA type, that is, the 7 transistor traditional design. The output load is defined to be 0.5 pF. Thus, under slewing conditions, the output stage current source must supply enough current to be able to slew both the output capacitive load AND the internal compensation capacitor simultaneously. The input stage, however, must be able to drive enough current to slew the compensation capacitor only.

The minimum slew rate is defined as follows, for a voltage swing of 5 volts and a 20 ns slew time period:

> min_slew_rate := dv/dt = 5 / 20e-9;

$$\text{min_slew_rate} := \frac{dv}{dt} = 250000000 \cdot 10^9$$

Thus, a minimum slew rate of 250×10^6 volts/sec is required. Let's assume this to be 300×10^6 V/sec, to allow for a bit of margin. The current drive required by the second stage to drive the output capacitance component is therefore:

> Output_stage_current_load_component := C * dv/dt = 0.5e-12 * 300e6;

$$\text{Output_stage_current_load_component} := \frac{C \cdot dv}{dt} = .0001500$$

The 0.5 pF load requires a slewing current of 150 uA. But the output stage also needs to supply slewing current to the compensation capacitor, which is, in this analysis, also 0.5 pF. Thus, another 150 uA is needed for the compensation capacitor slewing, and thus the total output stage current required is 300 uA.

The input stage needs only to slew the compensation capacitor. Therefore, it requires a current of 150 uA, by virtue of the above analysis.

The transistor netlist used for this analysis is shown here, for reference:

```
*****
*
* Netlist for Analysis 1: Two-stage OTA
*
*****
Rxx13 vc1 vc2 500
Cxcld avss vout 0.5P
Cxc10 vc2 vout 0.5P
Mxm8 vout vbias avdd avdd MPCH L=3U W=150U
Mxm7 vbias vbias avdd avdd MPCH L=3U W=25U
Mxm5 vx vbias avdd avdd MPCH L=3U W=75U
Mxm4 vc1 p vx vx MPCH L=2.0U W=200U
Mxm3 vm n vx vx MPCH L=2.0U W=200U
Mxm9 vout vc1 avss gnd MNCH L=1.5U W=300U
Mxm2 vm vm avss gnd MNCH L=1.5U W=75U
Mxm1 vc1 vm avss gnd MNCH L=1.5U W=75U
*
```

```
*****
* EXTERNAL MODEL REFERENCES:
*
*   MNCH  MPCH
*
*****
```

Resistor Rxr13 is the compensation resistor, whereas capacitor Cxc10 is the compensation capacitor. Cxcl0 is the load capacitor. Transistors Mxm3 and Mxm4 form the P channel differential pair. Transistor Mxm7 forms a current mirror driver transistor to generate the gate voltage reference for the input stage bias current source (Mxm5) and the output stage bias current source (Mxm8). Transistors Mxm1 and Mxm2 form the input stage current mirror, where the drain of Mxm1 is the output of the first stage. Transistor Mxm9 is the second stage driving transistor. The compensation network is designed around Mxm9.

We can now proceed with our analysis in the small signal domain, with the aim of understanding the issues and important parameters affecting the roots (poles and zeroes) of the opamp transfer function.

```
> with(smallsignal):
> read 'matrix.maple':
Warning: new definition for norm
Warning: new definition for trace
```

```
unknowns := [ p n iinput vc1 vc2 ir13 vout vbias vx vm ]
```

Use mnalist as input argument to mnasolve.

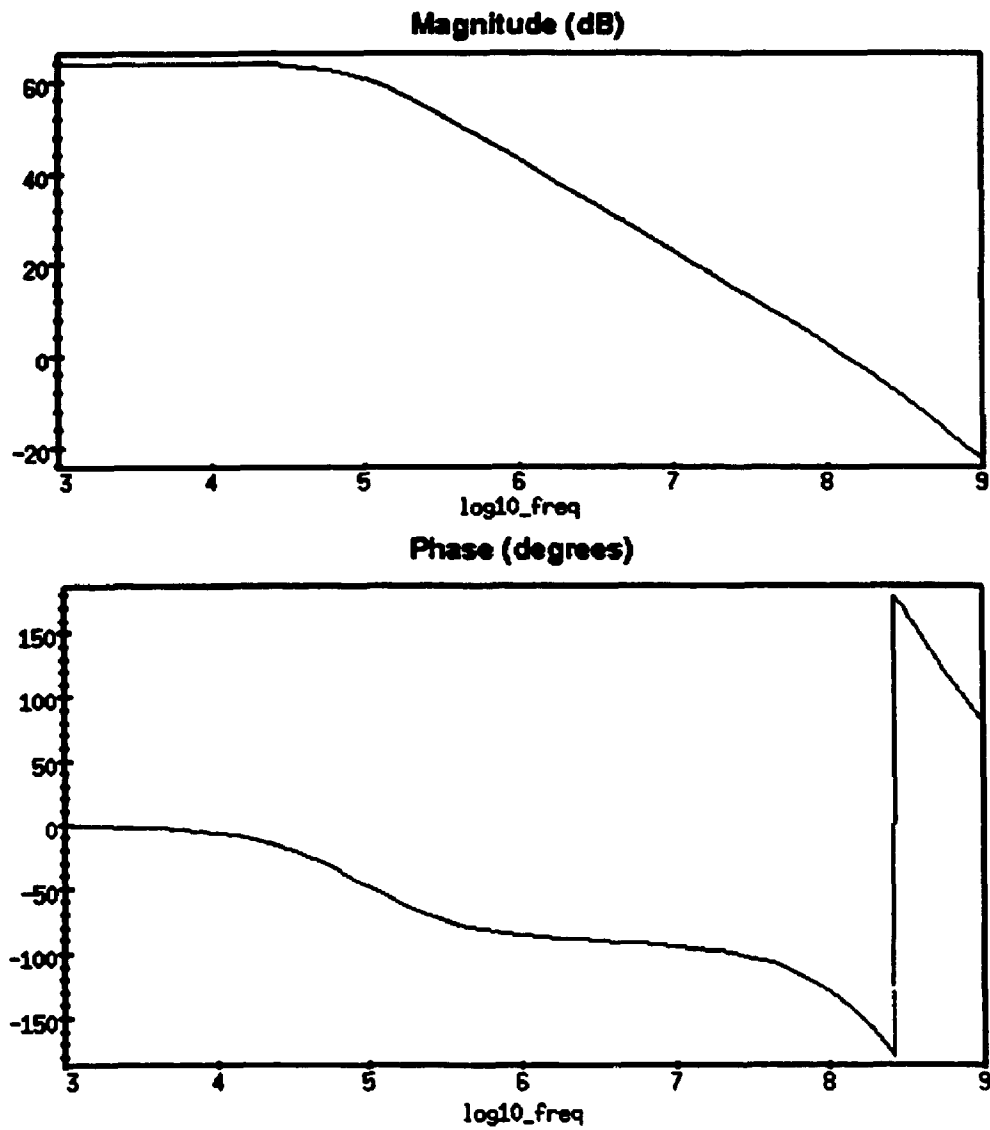
```
> vo := mnasolve( mnalist, vout ):
Solving for parameter vout in row #7
```

NOTE: The mnasolve routine does not simplify the expression at all. This has been chosen to be the case because of the enormous amounts of data produced. Only the reduced expression generated later in this worksheet will be simplified.

Now, let's compare the resulting transfer function with what the simulator produces. The comparison can be done graphically by examining the bode plots produced by Maple and by the simulator (eldo).

I have carefully examined that this Bode plot done in Maple looks identical to what Eldo produces under nominal simulation conditions.

> bodeplot(vo, 1e3, 1e9);



Next, the transfer function can be reduced using `reduce_order`. This routine examines each minterm present within each 's' coefficient, and substitutes it for its numerical value if the minterm is judged to have little effect over the coefficient value. The numbers used for substitution of minterms and to compute the coefficient value are found in the `lookup_table` routine.

The routine can also eliminate a complete 's' term if its effect over the frequency range of interest is observed to be of little importance. Hence the specification of a frequency range in the list of parameters. Note that the resulting expression is numerically identical to the original expression if no 's' terms are dropped. The routine simply substitutes the numeric value for each minterms that do not affect the expression very much (less than the `mtol` value). The expression thus becomes an approximation only if 's' terms are dropped.

For our amplifier, we want to maintain a high degree of accuracy. Therefore, we do not want to drop 's' terms, if possible. By specifying a value of 0.01 for `ctol`, the routine ends up keeping all 's' coefficients as the output below shows. We also want to keep symbolic all those minterms which affect the coefficient by more than 1%. Hence, let's specify `mtol` to also be 0.01.

```
> vo_reduced := reduce_order( vo, 0.01, 0.01, [log, 1e3, 1e10, 15] );
```

```
fmin: .1e4
fmax: .1e11
Number of Frequency Points: 15
```

```
List of Frequency Points:
```

```
=====
```

```
[1000., 3162., 10000., 31620., 100000., 316200., 1000 107, 3162 107, 1000 108, 3162 108, 1000 109, 3162 109,
.1000 1010, 3162 1010, .1000 1011]
```

The s terms kept in the numerator are: [s, s², s³, s⁴, s⁵, s⁷, s⁶]

The corresponding absolute maximum minterm sensitivity error is:

```
[.0087981895, .0114408574, .054441101, .0278842216, .062724012, .008578377, .0408742816]
```

The s terms kept in the denominator are: [s, s², s³, s⁴, s⁵, s⁷, s⁶]

The corresponding absolute maximum minterm sensitivity error is:

```
[.0233314769, .0493538851, .1003243939, .2081364647, .2459570891, .0208187807, .1792412700]
```

```
At frequency 1.000000e+03: Relative Error is 1.15e-12
At frequency 3.162000e+03: Relative Error is 2.23e-12
At frequency 1.000000e+04: Relative Error is 9.01e-12
At frequency 3.162000e+04: Relative Error is 1.28e-11
At frequency 1.000000e+05: Relative Error is 3.55e-11
At frequency 3.162000e+05: Relative Error is 2.11e-10
At frequency 1.000000e+06: Relative Error is 7.89e-13
```

```

At frequency 3.162000e+06: Relative Error is 2.00e-10
At frequency 1.000000e+07: Relative Error is 4.84e-12
At frequency 3.162000e+07: Relative Error is 1.11e-11
At frequency 1.000000e+08: Relative Error is 1.00e-10
At frequency 3.162000e+08: Relative Error is 1.07e-10
At frequency 1.000000e+09: Relative Error is 4.81e-12
At frequency 3.162000e+09: Relative Error is 1.46e-10
At frequency 1.000000e+10: Relative Error is 4.93e-12

```

Maximum relative error observed, 2106105221 10⁻⁹

Clearly, the reduction of the transfer function has not introduced any approximation in the resulting expression, as made obvious by the maximum relative error observed (10^{-9}). The explanation for this is because no 's' term has been dropped. We still get a small error due to roundoffs as the routine is internally limited to 10 digits of accuracy. The only reduction performed was the substitution of minterns for their numeric equivalents. Nevertheless, the reduced expression clearly is a very precise approximation of the original 'vo'. We can compare the two graphically by looking at the Bode plots. Since the plots look identical, I did not bother pasting them in the worksheet.

Note that the order of the s terms are presented here as they came out of Maple's internal data structures, and are therefore not in a proper ascending order.

```
> bodeplot( vo_reduced[1], 1e3, 1e9 );
```

We can now save the exact and reduced transfer functions, so that if we need to use them later in a different worksheet, we can read them directly from disk instead of recomputing them.

```
> save vo, vo_reduced, 'vo.m';
```

In order to view these roots graphically in the complex plane, a root locus plot can be performed. Since our goal is not to sweep any particular parameter, we can specify a single valued root locus. However, we need to identify a parameter, and find out its numeric value under typical PVT conditions. The routine `display indets` serves this purpose.

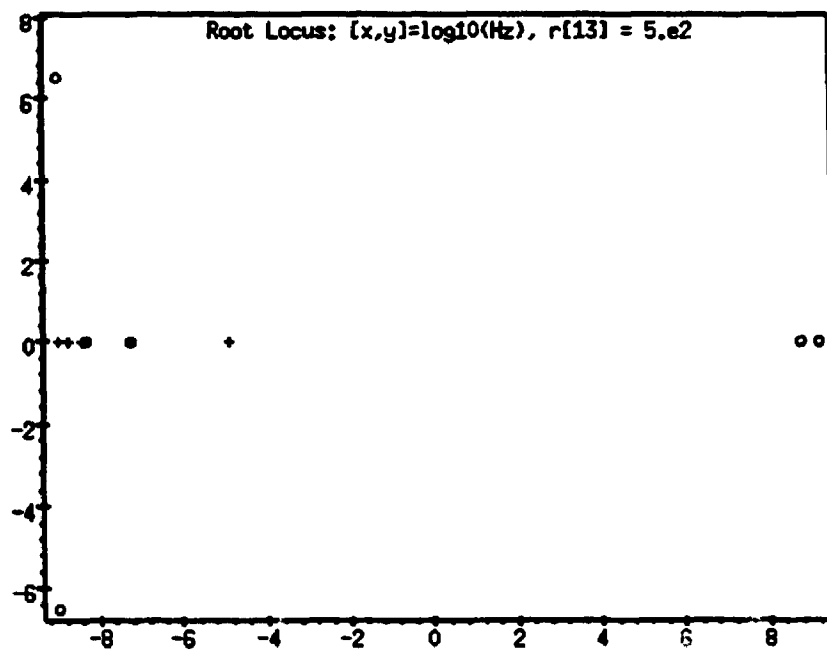
> display_indets(vo);

The following are the nominal PVIT values.

$$\left[\begin{array}{l} c_{gd1} = .8091400 \cdot 10^{-13}, g_{m4} = .0006003100, g_{ds4} = 2740800 \cdot 10^{-5}, c_{gs9} = .5386200 \cdot 10^{-12}, \\ c_{gd9} = .1309800 \cdot 10^{-12}, g_{ds1} = .00001644000, c_{gd1} = 2904000 \cdot 10^{-13}, g_{m1} = .0006962700, \\ c_{10} = .5000000 \cdot 10^{-12}, r_{13} = 500.0000, g_{m9} = .003340800, c_{1d} = .5000000 \cdot 10^{-12}, g_{ds9} = .00005956600, \\ c_{gs1} = .1318100 \cdot 10^{-12} \end{array} \right]$$

> rootlocus(vo, r[13], 500..500, 2, animate);

List of parametric points = [.500., 500.]



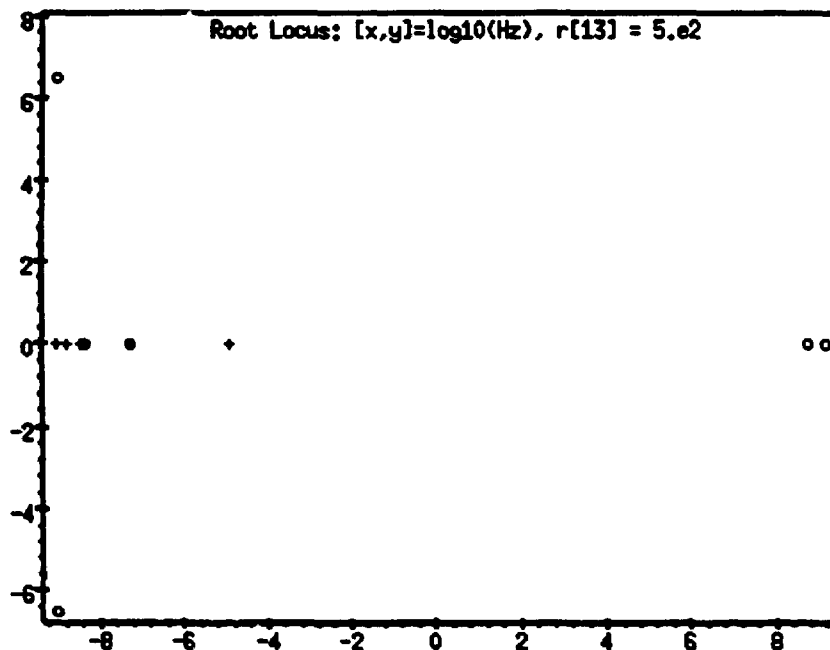
The x-axis is the frequency in $\log_{10}(\text{Hz})$, whereas the y axis is the imaginary frequency in $\log_{10}(\text{Hz})$.

There are 2 right-plane zeros, in the 500MHz range, and the dominant pole is roughly at 100 KHz, from the root locus plot. The right hand plane zeros may have an effect on the phase margin of the amplifier given that the amplifier UGBW is over 100MHz. Moreover, there are zeros and non-dominant poles in the neighborhood of the 100MHz range. It will be interesting to see the exact frequencies of these roots, and their sensitivity to the various symbolic parameters.

Note that it is not clear yet where the Miller compensation zero is located. The simplified theory of compensation states that for small values of the resistor, the Miller zero is in the right hand plane, whereas it will move into the left hand plane as $r[13]$ is increased. This simplified theory however assumes that there is no gate-drain capacitance on Mxm9, which is not the case in reality.

The immediate concern is, however, to convince ourselves that 'vo_reduced' has a similar root placement to 'vo', so that we can use 'vo_reduced' in computing our symbolic sensitivities, and use these results as hard facts to improve the performance of the amplifier.

```
> rootlocus( vo_reduced[1], r[13], 500..500, 2, animate );
      List of parametric points = [500., 500.]
```



The roots of 'vo_reduced' are placed similarly to 'vo'. Hence, we can use 'vo_reduced' to compute the symbolic sensitivities.

Next, the sensitivity of all roots with respect to all symbolic parameters is computed. The cutoff is specified to be 10%, meaning that only those parameters which affect the root by more than 10% are printed.

```
> read 'vo.m';
> sensitivities := print_sensitivities( vo_reduced[1], 0.1 );
```

Sensitivity Cutoff = 10.0 %

DC gain = 64.28 dB

Root Frequencies:

$$\left[z_1 = -202 \cdot 10^8, z_2 = -232 \cdot 10^9, z_3 = 509 \cdot 10^9, z_4 = -104 \cdot 10^{10} + 339 \cdot 10^7 i, z_5 = -104 \cdot 10^{10} - 339 \cdot 10^7 i, \right. \\ \left. z_6 = 134 \cdot 10^{10} \right]$$

$$\left[p_1 = -87000., p_2 = -197 \cdot 10^8, p_3 = -226 \cdot 10^9, p_4 = -285 \cdot 10^9, p_5 = -599 \cdot 10^9, p_6 = -106 \cdot 10^{10} \right]$$

Sensitivities:

```
table([
```

$$gm_4 = [1.0 = DCgain, .82 = z_2, 1.0 = z_3, .12 = p_2, .50 = p_3, .69 = p_4, -.54 = p_5, .15 = p_6]$$

$$s_{dB4} = [-.12 = DCgain, .13 = p_1, .17 = p_3, -.17 = p_4]$$

$$c_{g29} = [.17 = p_3, -.79 = p_5]$$

$$c_{g29} = [-.44 = z_6, -.18 = p_1, -.55 = p_3, .83 = p_4, -.10 = p_5, -.31 = p_6]$$

$$s_{dB1} = [-.85 = DCgain, .85 = p_1, .15 = p_3, -.15 = p_4]$$

$$c_{gdl} = [-.22 = p_3, .27 = p_4, -.12 = p_6]$$

$$c_{gsl} = [-.52 = z_4, -.52 = z_5, -.43 = p_3, .23 = p_4, -.41 = p_5, -.22 = p_6]$$

$$gm_1 = [.46 = z_4, .46 = z_5, 1.2 = p_3, -.76 = p_4, .45 = p_5]$$

$$c_{10} = [-.45 = z_4, -.45 = z_5, -.18 = z_6, -.76 = p_1, .28 = p_3, .88 = p_5, -1.4 = p_6]$$

$$r_{13} = [-.78 = z_4, -.78 = z_5, .38 = z_6, -.19 = p_3, .17 = p_4, 1.2 = p_5, -2.2 = p_6]$$

$$gm_9 = [.99 = DCgain, -.24 = z_4, -.24 = z_5, 1.4 = z_6, -.93 = p_1, .13 = p_3, .32 = p_4, 1.0 = p_5, -.61 = p_6]$$

$$c_{k2} = [.10 = p_4, -.65 = p_5, -.17 = p_6]$$

$$s_{dB9} = [-.93 = DCgain, .90 = p_1, .11 = p_3, -.11 = p_4]$$

$$c_{gdl} = [-.43 = z_3, .33 = p_3, -.41 = p_6]$$

```
])
```

These sensitivities were chosen to be printed out in the above format, which state how a given parameter affects all roots of the transfer function. An alternate way would be to reverse the relationship, and thus give a table that prints all roots, and for each root, show how the parameters affect it. Although there are arguments for both formats, we prefer the above because as a designer, we are generally interested in knowing how a given design parameter can affect the overall circuit, not just a single root.

There are some obvious conclusions which can be drawn from this table:

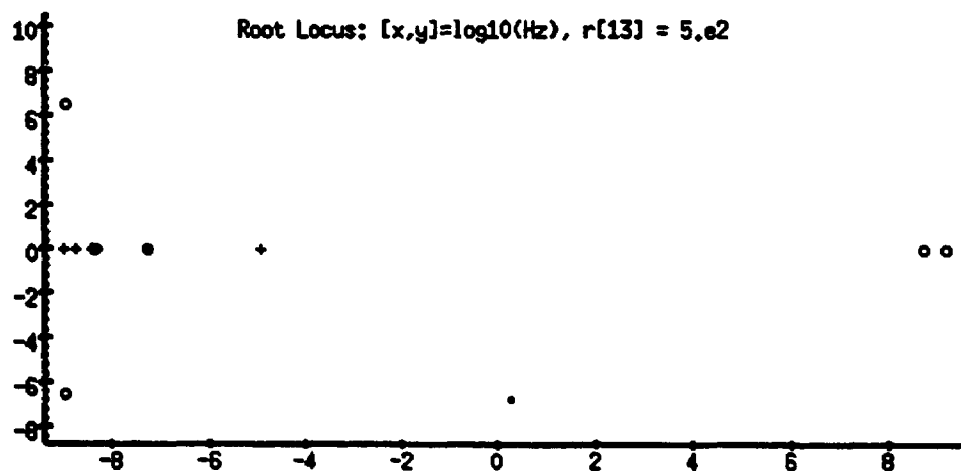
- There is a complex zero pair (rather unexpected).
- Looking at r[13], it is not clear which zero is the Miller zero. Will want to do a root locus where r[13] is swept, in order to view the Miller zero, perhaps at a different value of r[13].
- g[ds9] affects the DC gain in almost a one-to-one relationship. Specifically, a 1% increase in g[ds9] will result in a 0.93% decrease in DC gain. Simplified theory would expect 1% => -1%.
- gm[9] affects the DC gain directly. As expected.
- c[10] affects the dominant pole (p1) as -0.75% for a 1% increase. Thus, most of the dominant pole location comes from the effects of c[10]. c[gd9] affects the dominant pole by -0.18%. The addition of these two components yields a value that is close to -1, (-0.18 + -0.75 = -0.93) which makes sense, as they together form the effective Miller capacitor. The other 0.07% contribution probably comes from other parasitics.

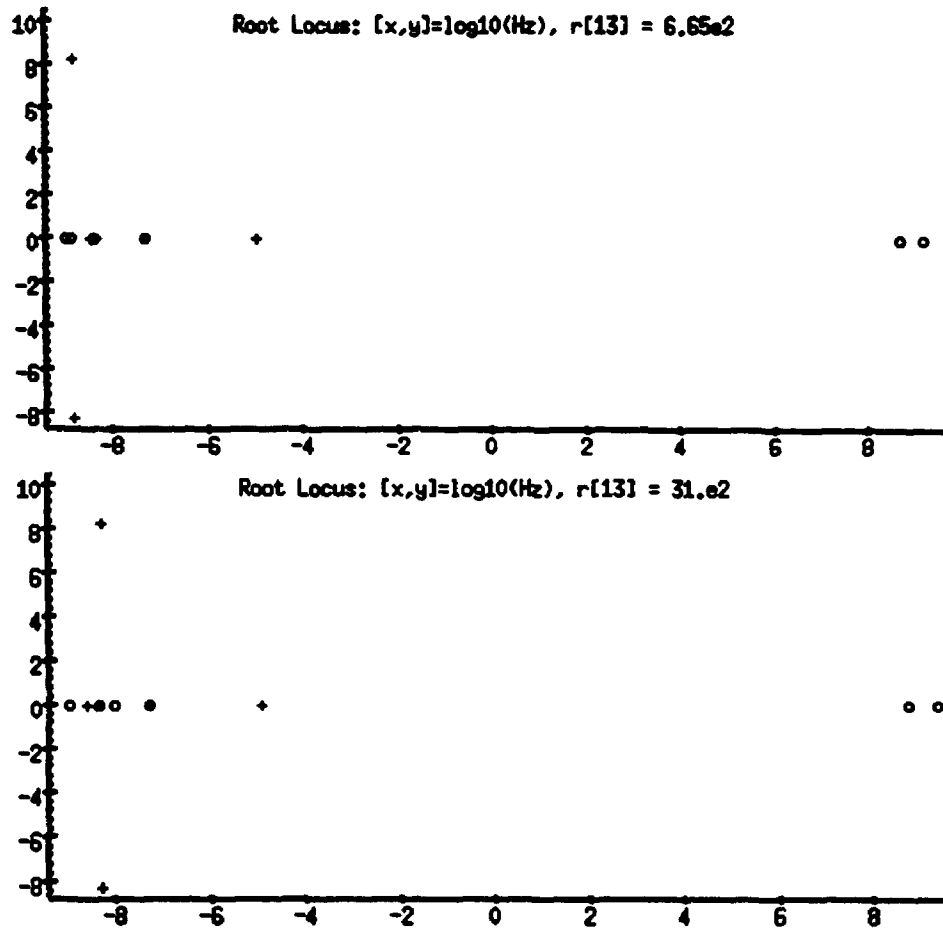
Before we analyze this table in too much depth, it would be instructive to find out where the Miller zero is located. This is done using the root locus with the animate mode.

The root locus is performed with a range of r[13] from, say, 500 ohms to 5000 Ohms. A close look at the results from the root locus will indicate what happens to the location of the roots, and, in particular, the Miller zero.

> rootlocus(vo, r[13], 500..5000, 25, animate);

List of parametric points = [500., 550., 605., 665., 735., 805., 890., 980., 1080., 1190., 1310., 1440., 1580., 1740., 1920., 2110., 2330., 2560., 2810., 3100., 3410., 3750., 4130., 4540., 5000.]

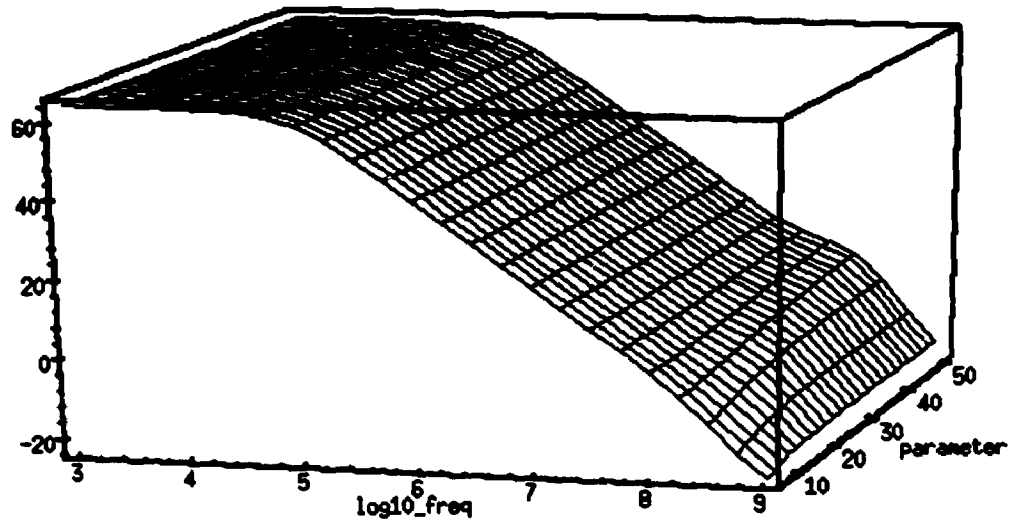




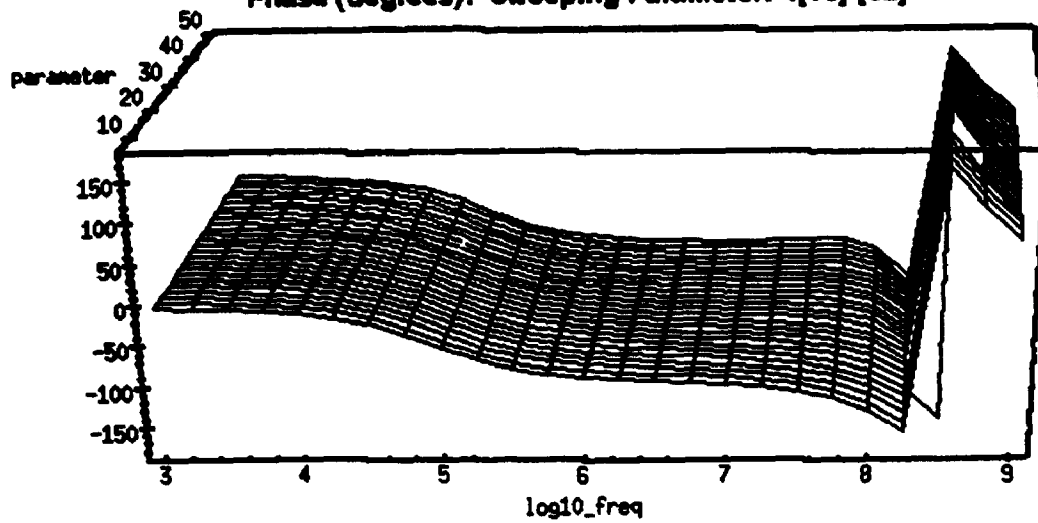
The results from the root locus indicate that the complex zero pair is complex only when $r[13]$ is about 500 Ohms. Moreover, as $r[13]$ is increased, one zero moves in to a lower frequency, bringing with itself a complex pole pair. However, the zero moves in faster than the complex pair, so this should definitely help the phase margin. This is our Miller zero. This effect is best visualized using a parametric (3D) Bode plot.

> bodeplot3d(vo, 1e3, 1e9, r[13]=500..5000);

Magnitude (dB): Sweeping Parameter: r[13] [e2]



Phase (degrees): Sweeping Parameter: r[13] [e2]



For the purpose of a sensitivity analysis, we can set $r[13]$ to be equal to 1200 Ohms, and perform the sensitivity analysis again. The 1200 Ohms value is chosen because from looking at the 3D Bode plot, this is where we start seeing the traditional 'bump' in the magnitude plot. We would therefore expect our final value for $r[13]$ to be in the neighborhood of 700 Ohms ~ 2000 Ohms, at a value where the bump can be seen, but yet not so much as to seriously affect the amplifier's gain margin. So, let's take an educated guess and plug 1200 Ohms for $r[13]$, and then recalculate the sensitivities. Thus, we set the lookup table value for $r[13]$ to be 1200. There is no need to recalculate $vo_reduced$ as it currently holds $r[13]$ as a symbolic parameter, which can therefore be altered.

```
> lookup_table[r[13]]:
                                500.0000
> lookup_table[r[13]] := 1200:
                                lookup_table := 1200
                                13
> lookup_table[r[13]]:
                                1200
> sens2 := print_sensitivities( vo_reduced[1], 0.1 );
```

Sensitivity Cutoff = 10.0 %

DC gain = 64.28 dB

Root Frequencies:

$$\left[z_1 = -202 \cdot 10^8, z_2 = -240 \cdot 10^9, z_3 = -324 \cdot 10^9, z_4 = .514 \cdot 10^9, z_5 = -.100 \cdot 10^{10}, z_6 = .176 \cdot 10^{10} \right]$$

$$\left[p_1 = -87000., p_2 = -.199 \cdot 10^9, p_3 = -.194 \cdot 10^9, p_4 = -.313 \cdot 10^9, p_5 = -.363 \cdot 10^9 + .385 \cdot 10^9 i, \right.$$

$$\left. p_6 = -.363 \cdot 10^9 - .385 \cdot 10^9 i \right]$$

Sensitivities:

```

table([
  gm4 = [1.0 = DCgain, 84 = z2, 1.0 = z4, .12 = p2, .42 = p3, .51 = p4]
  sds4 = [-.12 = DCgain, .13 = p1]
  cgs9 = [-.40 = p5, -.40 = p6]
  cgs10 = [-.53 = z6, -.18 = p1, -.29 = p3, .41 = p4, -.13 = p5, -.13 = p6]
  sds1 = [-.85 = DCgain, .85 = p1]
  cgs1 = [-1.2 = z5, -.20 = p3, -.24 = p4, -.20 = p5, -.20 = p6]
  gm1 = [-.13 = z2, .11 = z3, .99 = z5, .55 = p3, .30 = p4]
  c10 = [.11 = z2, -1.1 = z3, -.76 = p1, .15 = p3, -.22 = p5, -.22 = p6]
  r13 = [.16 = z2, -1.4 = z3, .23 = z6, -.18 = p3, -.45 = p5, -.45 = p6]
  gm9 = [.99 = DCgain, -.16 = z3, 1.2 = z6, -.93 = p1, .24 = p3, .29 = p5, .29 = p6]
  c12 = [.10 = p4, -.41 = p5, -.41 = p6]
  sds9 = [-.93 = DCgain, .90 = p1]
  cgs11 = [-.43 = z4, .10 = p2, .18 = p3, -.19 = p5, -.19 = p6]
])

```

Interestingly enough, now the Miller zero is clearly seen. Let's summarize our observations:
 - r[13] affects z3 by -1.4% for a 1% increase. Clearly, z3 is the Miller zero, and it is currently located at 324MHz.

- c[10] affects the dominant pole (p1) as -0.76% for a 1% increase. Thus, most of the dominant pole location comes from the effects of c[10]. c[gs9] affects the dominant pole by -0.18%. The addition of these two components yields a value that is close to -1, (-0.18 + -0.76 = -0.94) which makes sense, as they together form the effective Miller capacitor. The other 0.06% contribution probably comes from other parasitics.

- c[10] also affects the Miller zero (z3) as -1.1% for a 1% increase. It also affects the complex pole pair as -0.22% for a 1% increase. Thus, increasing c[10] would improve stability as the complex pair moves down in frequency slower than the Miller zero. However, it also decreases the dominant pole by 0.76, thus decreasing the bandwidth, as would be expected.

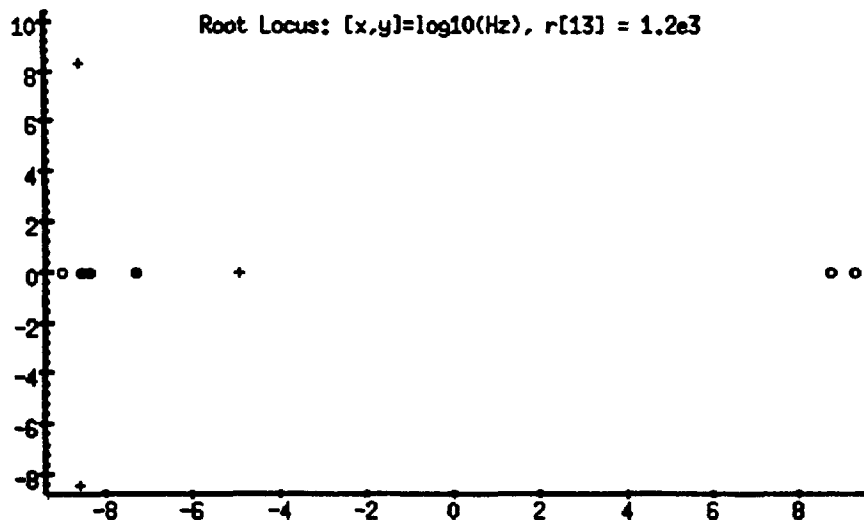
- $g[ds9]$ affects the DC gain in almost a one-to-one relationship. Specifically, a 1% increase in $g[ds9]$ will result in a 0.93% decrease in DC gain. Simplified theory would expect 1% \rightarrow -1%. It also affects the dominant pole in the opposite direction, by 0.9. Thus, $g[ds9]$ does not affect the unity gain bandwidth directly.
- $gm[9]$ affects the DC gain directly. As expected.
- $gm[9]$ also affects the complex pole pair, pushing them to a higher frequency by 0.29% for a 1% increase. So, improving $gm[9]$ would improve the stability, while increasing the DC gain.
- However, $c[gs9]$ and $c[gd9]$ both reduce the frequency of the complex pair by -0.4 and -0.13 respectively.

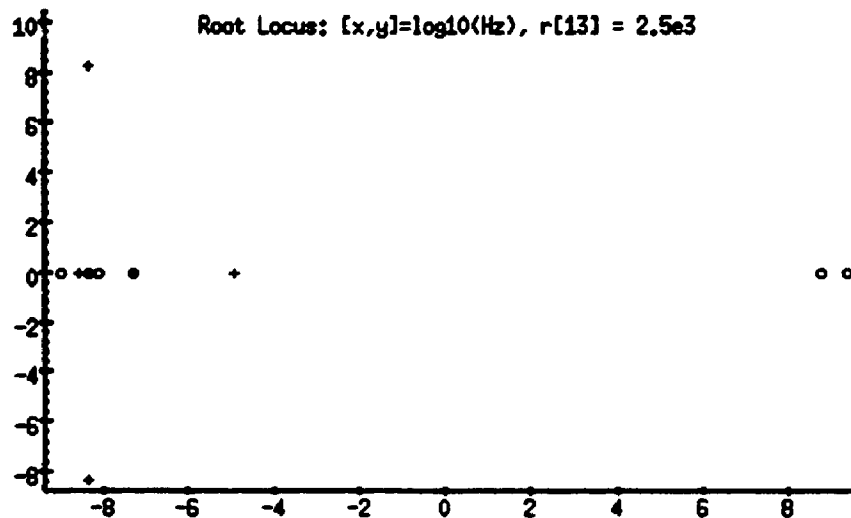
It would be instructive to obtain physical sensitivities for transistor M9, in particular. This way, we could see exactly how the length and the width (or their ratio) would affect the roots, and more easily reach conclusions on how to properly size M9.

But first, let's have a closer look at our root locus plot, and let's look closely at the position of all the relevant roots.

```
> rootlocus( vo, r[13], 1200..2500, 2, animate );
```

```
List of parametric points = [1200., 2500.]
```





These plots show the position of the roots for $r[13] = 1200$ and 2500 respectively. There are two high frequency doublets (one in the low MHz and one in the 100's of MHz) which do not seem to be affected by the Miller resistor. As the Miller zero moves down in frequency, so does the complex pole pair, but only slightly. Also, there is another high frequency pole which will erode the phase margin. In total, we have the complex pole pair and a high frequency pole (3 poles) which we need to consider in our stability analysis. To compensate for this, all we really have is the Miller zero, since the other high frequency zero is in the GHz range.

Referring back to our sensitivity table above, we see that one high frequency doublet is composed of $z2$ (240MHz) and $p3$ (194MHz). The Miller zero (for $r[13] = 1200$ ohms) is $z3$ at 324 MHz, which is currently cancelled out by $p4$ at 313MHz. The complex pole pair is at about 529 MHz (after taking the magnitude of the complex vector).

Pushing out all poles $p3$, $p4$ and the complex pole pair is impossible, as all these elements need to be traded off. This can be readily seen by looking at the sensitivity numbers in the table.

One tradeoff we could do is to try to push out $p3$ and $p4$ to a higher frequency. This would allow us to have a stable amplifier even with a lower complex pole pair frequency. The phase margin would be very good, because one of the complex poles would be cancelled by the Miller zero, while the other can only give a 90 degree phase shift maximum, in addition to the initial 90 degrees phase shift from the dominant pole. From this point of view, then $p3$ and $p4$ are the ultimate limiting factor in the stability of the amplifier, as long as the complex pole pair doesn't come too far down in frequency. Note that in such a case, the amplifier would still be stable, but the phase margin would be very low.

Another scenario would be to try to push the complex pole pair to a much higher frequency, at the expense of bringing down either one or both of $p3$ and $p4$. This is an unlikely scenario, however, because it implies a much more substantial change in the position of the poles from their current locations.

In order to gain insight on the effects of the transistor lengths and widths, the sensitivity with respect to geometry is computed. That is, the deltagen system is used in UNIX to generate 3 lookup tables: lookup l10, lookup w10, and lookup lw10. The first lookup table was created with a netlist where all transistor lengths were increased by 10%. The second table was created with a netlist where all transistor widths were increased by 10%, and the third table, where both lengths and widths were increased by 10% simultaneously.

The idea is to compute the sensitivity of each symbolic parameter with respect to geometry. Then, these sensitivities are multiplied with the above table, to see how changing a particular transistor size will affect the roots. Finally, all sensitivities affecting a given transistor are added together, to give the net resulting sensitivity. Note that in order for this to be accurate, the sensitivities with respect to all relevant symbolic parameters must be computed a priori, so that the aggregate sensitivity includes all important parameters.

Since the location of the roots has changed, we must recalculate the sensitivity with respect to the physical parameters. It is not necessary to run again the UNIX script deltagen since the only parameter that was modified is the value for r[13]. Since r[13] does not affect the DC bias conditions of the amplifier, the small signal parameters of all active devices will not change. Hence, we can simply modify the value of r[13] in all three lookup tables, and calculate the physical sensitivities using the deltagen routine.

```

> lookup_l10[r[13]]:
                    500.0000
> lookup_l10[r[13]] := 1200:
                    lookup_l10 := 1200
                      r13
> lookup_lw10[r[13]]:
                    500.0000
> lookup_lw10[r[13]] := 1200:
                    lookup_lw10 := 1200
                      r13
> lookup_w10[r[13]]:
                    500.0000
> lookup_w10[r[13]] := 1200:
                    lookup_w10 := 1200
                      r13

```

> lsens2 := deltagen(vo_reduced[1], lookup_l10, sens2[1], [1, 4, 9], 0.1):

Sensitivities for parameter: l

=====

$$\left[\begin{array}{l} 03831 = c_{gd4} - 4598 = gm_4, -2.492 = g_{db4}, 9729 = c_{gs9}, 05344 = c_{gd9}, -1.916 = g_{ds1}, 18264 = c_{gd1}, \\ -4510 = gm_1, 0 = c_{10}, 0 = r_{13}, -7603 = gm_9, 0 = c_{d2}, -2.312 = g_{ds9}, 1.024 = c_{gs1} \end{array} \right]$$

MOS1 sensitivities using parameters, $\{g_{ds1}, c_{gd1}, gm_1, c_{gs1}\}$, are:

$$[1.6 = DCgain, -58 = p_3, -21 = p_5, -21 = p_6, -30 = p_4, -1.6 = p_1, -1.6 = z_5]$$

MOS4 sensitivities using parameters, $\{c_{gd4}, gm_4, g_{db4}\}$, are:

$$[-.16 = DCgain, -.39 = z_2, -.37 = p_3, -.11 = p_4, -.32 = p_1, -.50 = z_4]$$

MOS9 sensitivities using parameters, $\{c_{gs9}, c_{gd9}, gm_9, g_{ds9}\}$, are:

$$[1.4 = DCgain, -.26 = p_3, -.61 = p_5, -.61 = p_6, -1.4 = p_1, -.96 = z_6, .11 = z_3]$$

> wsens2 := deltagen(vo_reduced[1], lookup_w10, sens2[1], [1, 4, 9], 0.1):

Sensitivities for parameter: w

=====

$$\left[\begin{array}{l} 1.002 = c_{gd4}, 4264 = gm_4, 2809 = g_{db4}, 9246 = c_{gs9}, 1.000 = c_{gd9}, 3345 = g_{ds1}, 9986 = c_{gd1}, 4108 = gm_1, \\ 0 = c_{10}, 0 = r_{13}, 4819 = gm_9, 0 = c_{d2}, 4365 = g_{ds9}, 9104 = c_{gs1} \end{array} \right]$$

MOS1 sensitivities using parameters, $\{g_{ds1}, c_{gd1}, gm_1, c_{gs1}\}$, are:

$$[-27 = DCgain, -18 = p_5, -18 = p_6, -12 = p_4, 29 = p_1, -.65 = z_5]$$

MOS4 sensitivities using parameters, $\{c_{gd4}, gm_4, g_{db4}\}$, are:

$$[.41 = DCgain, 29 = z_2, .38 = p_3, -.22 = p_5, -.22 = p_6, .22 = p_4, .15 = p_2, .10 = z_3]$$

MOS9 sensitivities using parameters, $\{c_{gs9}, c_{gd9}, gm_9, g_{ds9}\}$, are:

$$[-.11 = p_3, -.36 = p_5, -.36 = p_6, .42 = p_4, -.25 = p_1, -.13 = z_3]$$

> hwsens2 := deltagen(vo_reduced[1], lookup_hw10, sens2[1], [1, 4, 9], 0.1):

Sensitivities for parameter: lw

=====

$$\left[\begin{array}{l} 1.043 = c_{gd4} - 0.4831 = gm_4, -2.302 = g_{ds4}, 1.987 = c_{gs9}, 1.061 = c_{gd9}, -1.648 = g_{ds1}, 1.088 = c_{gd1}, \\ -0.05601 = gm_1, 0 = c_{10}, 0 = r_{13}, -3.263 = gm_y, 0 = c_{kx}, -1.993 = g_{ds9}, 2.026 = c_{gs1} \end{array} \right]$$

MOS1 sensitivities using parameters, $\{g_{ds1}, c_{gd1}, gm_1, c_{gs1}\}$, are:

$$\left[1.4 = DCgain, -62 = p_3, -41 = p_5, -41 = p_6, -44 = p_4, -1.4 = p_1, -2.4 = z_5 \right]$$

MOS4 sensitivities using parameters, $\{c_{gd4}, gm_4, g_{ds4}\}$, are:

$$\left[24 = DCgain, -11 = z_2, -19 = p_5, -19 = p_6, .11 = p_4, -28 = p_1, -52 = z_4 \right]$$

MOS9 sensitivities using parameters, $\{c_{gs9}, c_{gd9}, gm_y, g_{ds9}\}$, are:

$$\left[1.5 = DCgain, -.39 = p_3, -1.0 = p_5, -1.0 = p_6, .48 = p_4, -1.7 = p_1, -.95 = z_6 \right]$$

OBSERVATIONS:

1. Increasing the width of M4 will improve the DC gain, and it will push the first and second non-dominant poles (p3 and p4) to a higher frequency. However, beware of the complex pole pair (p5 and p6) coming down in frequency as $W4$ is increased too much.
2. The length of M4 should be reduced. This will push the dominant pole (p1) and the first non-dominant pole (p3) to a somewhat higher frequency, which improves the bandwidth of the amplifier. Stability remains mostly unaffected since $z2$ also moves up.
3. The length of M1 should be increased, as it will improve the DC gain while pushing the dominant pole (p1) down. Hence, the UGBW will not be affected. The first non-dominant pole (p3) will also come down, which effectively counteracts its previous increase created by a reduction of the length of M4 in step 2. The end result should be a higher DC gain.
4. The length of M9 should be kept reasonably small. This will reduce the already improved DC gain, and trade it off with a better overall stability, as the complex pole pair (p5 and p6) will be pushed to a higher frequency. A shorter length implies less DC gain, but it also pushes the dominant pole higher, and therefore the overall UGBW will remain unchanged. Since the complex pole pair is pushed to a higher frequency, the Miller zero is thus capable to maintain a good phase margin up to a higher frequency, and the end result is an improvement in stability.
5. Finally, the value of r13 should be adjusted such that the phase and gain margins are optimal. From the above sensitivity analysis, the value of r13 affects mostly the location of the Miller zero and the complex pole pair (p5 and p6). Thus, the idea was to first push the complex pole pairs as high as possible while bearing in mind other specifications (DC gain, etc), and lastly, the value of r13 can be optimized to place the zero at the best location for proper design centering.

The end result is therefore to move p3 and p4 to a higher frequency, at the cost of bringing down the complex pole pair somewhat. This will result in higher performance from a gain and bandwidth perspective (due in part to more gain via the input pair), while maintaining a high phase margin. It is also expected that the phase will drop off very quickly beyond the unity gain frequency because of the effects of p3 and p4.

APPENDIX C Selected code from the *smallsignal* package

The following is a listing of some selected pieces of code from the '*smallsignal*' Maple package, implementing some of the key algorithms used by the package. The complete '*smallsignal*' package comprises over 2500 lines of code, in addition to the maplenet C program (1500 lines) and several *awk* and *cshell* scripts. The interested reader is urged to contact the author for further information about the scripts, the Maple code and the algorithms used in the package.

```

#=====
#
# deltagen:
# This routine is used to generate sensitivities
# based on simulated results as opposed to symbolic
# information. For instance, the normalized sensitivities
# of each symbolic parameter can be found for a variation
# in the respective transistor's length, or the temperature.
# The reference table is lookup_table. A delta table is
# passed as argument. This delta table contains the name
# of the parameter varied as an entry called [param], and
# the amount of variation [factor].
#=====
smallsignal(deltagen) := proc(expression, lookup_delta,
                             sensitivities, moslist, cutoff)
    local ijk, indets_list, myfactor, nv, nv_zip, index,
          nsens, OldDigits, moszip, mosvector, mosprint, mos,
          mosparam, index2, tablesens, rootset, param_namelist,
          param_name, rootlist, param_used, printlist;

    indets_list := [op(indets(expression, name) minus {s})];
    printf("\n ");
    printf("Sensitivities for parameter: %a\n ", lookup_delta[param]);
    printf("===== \n ");

    OldDigits := Digits;
    Digits := 4;

    #-----
    # Get the normalized variations.
    #-----
    myfactor := lookup_delta[factor] - 1.0;
    nv := map((x,y,z) -> ((z[x] - lookup_table[x] / (lookup_table[x] * y)),
              indets_list, myfactor, lookup_delta);

    nv_zip := zip((x,y) -> x=y, nv, indets_list);

    print(nv_zip);

```

```

#-----
# Generate the new sensitivities based on
# the normalized variations. i.e. gm*gm(length)
# for each root.
#-----
for i from 1 to nops(indets_list) do
  index := indets_list[i];
  nsens[index] := map((x,y)->op(1,x)*y=op(2,x),
    sensitivities[index], rv[i]);
od;

#-----
# Now, we need to multiply the parametric
# sensitivities for all MOS transistors
# First, define the MOS param list and convert the
# sensitivities list to a 2D table for easy reference.
# This is to find out if the sensitivity is assigned.
#-----
mosparam := [c[gs], c[gd], gm, g[ds] ];

for i from 1 to nops(indets_list) do
  index := indets_list[i];
  for k from 1 to nops(nsens[index]) do
    index2 := op(2, nsens[index][k]);
    tablesens[index][index2] := op(1, nsens[index][k]);
  od;
  tablesens[index][indices] := map(x->op(x),
    {indices(tablesens[index])});
od;

#-----
# Now build the physical sensitivities.
# First, we must extract all the root's name affecting
# a given MOS transistor. This is stored in rootlist[i].
# Also, the relevant parameters affecting each MOS
# are extracted and stored in param_namelist[i].
#-----
for i in moslist do
  rootset[i] := { };
  param_namelist[i] := [ ];
  for j from 1 to nops(mosparam) do
    if type(mosparam[j], indexed) then
      index := op(1, mosparam[j]);
      param_name := op(0, mosparam[j])["`index.i"];
    else
      param_name := op(1, mosparam[j])[i];
    fi;
    param_namelist[i] := [op(param_namelist[i]), param_name];
    if assigned(tablesens[param_name][indices]) then
      rootset[i] := rootset[i] union convert(tablesens[param_name][indices], set);
    fi;
  od;

  rootlist[i] := [op(rootset[i])];

#-----

```

```

# Now build the physical sensitivities
#-----
param_used[i] := {};
for j in param_namelist[i] do
  for k in rootlist[i] do
    if assigned(tablesens[j][k]) then
      param_used[i] := {op(param_used[i]), j};
      if assigned(mos[i][k]) then
        mos[i][k] := tablesens[j][k] + mos[i][k];
      else
        mos[i][k] := tablesens[j][k];
      fi;
    fi;
  od;
od;

print('MOS `i.` sensitivities using parameters `', param_used[i], ` are:');

mosprint[i] := {};
mosvector := map(x->op(x), [indices(mos[i])]);
for j in mosvector do
  mosprint[i] := [op(mosprint[i]), mos[i][j]];
od;
moszip[i] := zip((x,y)->x=y, mosprint[i], mosvector);

#-----
# Don't print those below cutoff.
#-----
printlist := [ ];
for j from 1 to nops(moszip[i]) do
  if abs(op(1, moszip[i][j])) >= cutoff then
    printlist := [ op(printlist), moszip[i][j] ];
  fi;
od;
print(evalf(eval(printlist), 2));

od;

Digits := OldDigits;

RETURN( [nv_zip, moszip, eval(nsens)] );

end:

#=====
#
# The following is a routine to compute the sensitivities
# of a polynomial in terms of 's' (var).
#
#=====
smallsignal[root_sensitivities] := proc( pol, var, values )
  local i, j, inds, ref, part_diffs, roots, ans, max_mag,
        returnlist, nlist, OldDigits;
#-----

```

```

# Get the indeterminates minus 's', and put them in a list.
#-----
inds := [op(indets(pol, name) minus (var))];

#-----
# Calculate the partial derivatives. Since Maple can't
# handle floats, convert to fractions (rationals).
#-----
rof := RootOf(convert(pol, rational), var);
part_diffs := [seq(diff(rof, i), i=inds)];

#-----
# Now, find the numerical roots, and put them in a list.
# But first, set accuracy to 10 digits.
#-----
OldDigits := Digits;
Digits := 10;
roots := [fsolve(subs(values, pol), var, complex)];
Digits := OldDigits;

#-----
# Now, find the values of the partial derivatives
# at each of the roots.
# First, we plug in the roots, followed by the numerical
# value for each of the indeterminates.
# We obtain a list of lists, where each sublist corresponds
# to a root, and each element within that sublist is the
# numerical value of the sensitivity at the root for a
# particular indeterminate (symbolic parameter).
#-----
ans := map((x,y,z)->eval(subs(y=x,x)), roots, rof, part_diffs);
ans := evalf(eval(subs(values, ans)), 25);

#-----
# Now a bit of math. We are interested in:
#
#      d |root|          | 1  d root \
#      ----- = |root| * Re{ ---- * ----- }
#      d parm           \root  d parm /
#
# The above formula can be proved using natural logs.
#
# The idea is to obtain a real relative sensitivity from a complex
# sensitivity.
#
# To get a relative sensitivity, we multiply the above result by
# abs(parm/root). This gives a % variation in the root for each %
# variation in the parameter.
#-----
ans := [seq( [seq( map((x,y,z)->abs(y)*Re(x/z), ans[j])[i], lookup_table[inds[i]], roots[j]),
              i=1..nops(inds))], j=1..nops(ans))];
#-----

```

```

# Combine the sensitivity values to the indeterminate
# symbolic name.
#-----
nlist := map((x,y) -> zip((z,w) -> z=w,x,y), ans, inds);

returnlist := [ roots, nlist, part_diffs ];

end:

#=====
#
# This routine takes a polynomial coefficient as input,
# and substitute numerically those minterms which have an
# effect less than mtol over the coefficient value.
#
# Since the routine comprises a single for loop, it is fast
# and efficient, wha. . the computing time is proportional to
# the number of minterms appearing in the coefficient.
#
# The returned list comprises two elements. The symbolic/hybrid
# coefficient, and the maximum error assumed if each substitution would
# have been with a zero. Note that the returned coefficient is
# exact, as those symbolic minterms have been replaced by their
# equivalent numeric value. The error returned thus gives a feel
# for the sensitivity lost over a variation of the symbolic elements,
# since from a sensitivity perspective, this is equivalent to
# neglecting these terms.
#
#=====

smallsignal[reduce_single_coeff] := proc( minterm, mtol )
  local mint_num, mint_symb, mint_val, abs_mint_val,
        mint_count, i, mint_error;

  #-----
  # first, convert the minterm to a list form.
  # subs into mint_num, and evaluate minterm.
  #-----
  mint_symb := [op(minterm)];
  mint_num := subs(op(eval(lookup_table)), mint_symb );

  mint_val := convert( mint_num, '+' );
  abs_mint_val := abs(mint_val);
  mint_count := nops(mint_num);

  #-----
  # Now, loop through every minterm, and choose
  # to either keep it symbolic or make it numeric.
  # To evaluate the total error, substitute 0 into
  # the numeric list.
  #-----
  for i from 1 to mint_count do
    if abs(mint_num[i]) < mtol*abs_mint_val then
      mint_symb := [ mint_symb[1..i-1], mint_num[i], mint_symb[i+1..mint_count] ];
      mint_num := [ mint_num[1..i-1], 0, mint_num[i+1..mint_count] ];
    end if;
  end for;
end proc;

```

```

    ft;
  od;

  mint_error := abs( 1 - (convert(mint_num, '+')/mint_val) );

  RETURN( ( convert(mint_symb, '+'), mint_error ) );

end:

#####
#
# This routine takes a polynomial in the following
# polylist format:
# [ [ c1, c2, c3, ..., cn ], [ s^0, s^1, ..., s^n ] ]
#
# and reduces each coefficients so that it contains
# a minimum number of minterms.
#
#####
smallsignal[reduce_all_coeffs] := proc( polylist, msol )
  local i, coeff_count, polylist2, new_coeff, max_error;

  polylist2 := copy(polylist);
  coeff_count := nops(polylist[1]);
  max_error := [ xxx ];
  for i from 1 to coeff_count do
    if polylist[1][i] = 0 then
      new_coeff := [0,0];
    else
      new_coeff := smallsignal[reduce_single_coeff]( polylist[1][i], msol );
    fi;
    polylist2 := [ [ polylist2[1][1..i-1], new_coeff[1], polylist2[1][i+1..coeff_count] ],
      polylist2[2] ];
    max_error := [ op(max_error), new_coeff[2] ];
  od;
  max_error := [ max_error[2], nops(max_error) ];
  RETURN( ( polylist2, max_error ) );
end:

```

APPENDIX D Autovit system

The following is a listing over all 128 simulation corners of the amplifier designed in the case study (section 4.0 on page 91) after design centering and optimization.

Opamp Open-loop Frequency Characterization

Simulation	DC Gain	UGBW	Phase Margin	Gain Margin
0	61.8	1.88e+08	67.0	-0.0
1	64.1	1.19e+08	62.8	41.4
2	63.3	1.33e+08	65.0	38.5
3	60.4	1.26e+08	72.1	-0.0
4	59.8	1.41e+08	75.2	-0.0
5	63.9	1.41e+08	63.7	35.6
6	63.0	1.58e+08	65.6	38.8
7	60.2	1.50e+08	73.3	-0.0
8	59.7	1.78e+08	76.2	-0.0
9	63.5	2.37e+08	71.3	36.0
10	63.3	2.82e+08	71.2	-0.0
11	60.4	2.37e+08	80.3	-0.0
12	60.3	2.99e+08	81.7	-0.0
13	63.4	2.82e+08	68.2	27.7
14	63.1	3.55e+08	63.9	-0.0
15	60.3	2.99e+08	77.5	-0.0
16	60.2	3.76e+08	76.3	-0.0
17	64.1	1.06e+08	51.6	41.2
18	63.3	1.19e+08	53.2	38.5
19	60.4	1.72e+08	60.4	-0.0
20	59.8	1.26e+08	63.0	-0.0
21	63.9	1.19e+08	52.6	36.9
22	63.0	1.33e+08	54.3	38.8
23	60.2	1.26e+08	61.5	-0.0
24	59.7	1.41e+08	64.7	-0.0
25	63.5	1.78e+08	63.3	38.1
26	63.3	2.00e+08	66.1	-0.0
27	60.4	1.78e+08	70.8	-0.0
28	60.3	2.11e+08	74.7	-0.0
29	63.4	2.00e+08	64.3	31.6
30	63.1	2.37e+08	67.0	-0.0
31	60.3	2.11e+08	72.1	-0.0
32	60.2	2.51e+08	76.0	-0.0
33	64.1	1.33e+08	57.3	39.3
34	63.3	1.58e+08	60.0	36.1
35	60.4	1.41e+08	65.9	-0.0
36	59.8	1.58e+08	69.6	-0.0
37	63.9	1.58e+08	58.1	35.4
38	63.0	1.78e+08	60.7	36.4
39	60.2	1.68e+08	67.1	-0.0
40	59.7	1.88e+08	70.8	-0.0
41	63.5	2.37e+08	67.6	36.4
42	63.3	2.99e+08	69.2	-0.0
43	60.4	2.51e+08	74.9	-0.0

44	60.3	2.99e+08	78.7	-0.0
45	63.4	2.99e+08	65.7	28.7
46	63.1	3.55e+08	66.1	-0.0
47	60.3	2.99e+08	74.3	-0.0
48	60.2	3.76e+08	76.5	-0.0
49	64.1	1.26e+08	47.7	39.1
50	63.3	1.41e+08	49.8	36.1
51	60.4	1.33e+08	56.0	-0.0
52	59.8	1.50e+08	59.1	-0.0
53	63.9	1.41e+08	48.3	36.0
54	63.0	1.58e+08	50.5	36.4
55	60.2	1.50e+08	56.7	-0.0
56	59.7	1.68e+08	59.9	-0.0
57	63.5	2.00e+08	57.7	37.2
58	63.3	2.27e+08	60.9	-0.0
59	60.4	2.11e+08	64.7	-0.0
60	60.3	2.37e+08	68.9	-0.0
61	63.4	2.24e+08	58.4	31.5
62	63.1	2.66e+08	61.7	-0.0
63	60.3	2.37e+08	65.6	-0.0
64	60.2	2.82e+08	70.2	-0.0
65	64.3	1.19e+08	62.9	41.1
66	63.4	1.41e+08	64.7	38.3
67	60.5	1.26e+08	72.1	-0.0
68	59.9	1.50e+08	75.3	-0.0
69	64.0	1.41e+08	63.8	35.4
70	63.2	1.68e+08	65.1	38.6
71	60.3	1.50e+08	73.3	-0.0
72	59.7	1.78e+08	76.0	-0.0
73	63.6	2.37e+08	71.2	35.8
74	63.3	2.82e+08	70.8	-0.0
75	60.4	2.51e+08	79.6	-0.0
76	60.4	3.16e+08	80.5	-0.0
77	63.4	2.99e+08	66.4	27.5
78	63.2	3.55e+08	63.6	-0.0
79	60.3	3.16e+08	76.1	-0.0
80	60.3	3.76e+08	75.9	-0.0
81	64.3	1.06e+08	51.7	41.0
82	63.4	1.19e+08	53.2	38.3
83	60.5	1.12e+08	60.5	-0.0
84	59.9	1.26e+08	62.9	-0.0
85	64.0	1.26e+08	52.3	36.6
86	63.2	1.41e+08	54.0	38.6
87	60.3	1.26e+08	61.6	-0.0
88	59.7	1.41e+08	64.0	-0.0
89	63.6	1.78e+08	63.3	37.3
90	63.3	2.00e+08	66.0	-0.0
91	60.4	1.88e+08	71.0	-0.0
92	60.4	2.11e+08	74.5	-0.0
93	63.4	2.00e+08	64.4	31.4
94	63.2	2.37e+08	66.8	-0.0
95	60.3	2.11e+08	72.1	-0.0
96	60.3	2.51e+08	75.8	-0.0
97	64.3	1.41e+08	57.2	34.0
98	63.4	1.58e+08	59.8	35.9
99	60.5	1.41e+08	65.8	-0.0
100	59.9	1.68e+08	69.8	-0.0

101	64.0	1.58e+08	58.1	35.2
102	63.2	1.88e+08	60.3	36.2
103	60.3	1.68e+08	67.0	-0.0
104	59.7	2.00e+08	70.8	-0.0
105	63.6	2.51e+08	67.0	35.7
106	63.3	2.99e+08	68.8	-0.0
107	60.4	2.51e+08	74.7	-0.0
108	60.4	3.16e+08	78.2	-0.0
109	63.4	2.99e+08	65.5	28.6
110	63.2	3.55e+08	65.8	-0.0
111	60.3	3.16e+08	73.6	-0.0
112	60.3	3.76e+08	76.0	-0.0
113	64.3	1.26e+08	47.7	38.9
114	63.4	1.41e+08	49.7	35.9
115	60.5	1.33e+08	56.0	-0.0
116	59.9	1.50e+08	58.9	-0.0
117	64.0	1.50e+08	48.0	35.7
118	63.2	1.58e+08	50.3	36.2
119	60.3	1.50e+08	56.7	-0.0
120	59.7	1.68e+08	59.7	-0.0
121	63.6	2.00e+08	57.6	36.9
122	63.3	2.37e+08	60.9	-0.0
123	60.4	2.11e+08	64.6	-0.0
124	60.4	2.37e+08	68.6	-0.0
125	63.4	2.37e+08	58.3	31.4
126	63.2	2.66e+08	61.5	-0.0
127	60.3	2.37e+08	65.5	-0.0
128	60.3	2.82e+08	69.9	-0.0

NOTE: A gain margin of -0.0 implies that no gain margin was found by the script