Automated Design Synthesis of CMOS Operational Amplifiers

by

Ognen J. Nastov

S.B. Elect. Eng., Massachusetts Institute of Technology (1991)

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Science

at the Massachusetts Institute of Technology

February 1994

© Ognen J. Nastov, 1994

The author hereby grants to MIT permission to reproduce and to distribute copies of this thesis document in whole or in part.

Signature of Author ....
Department of Electrical Engineering and Computer Science
January 19, 1994

Certified by ....
Gerald J. Sussman
Matsushita Professor of Electrical Engineering
Thesis Supervisor

Accepted by ....
Frederic R. Morgenthaler
Chairman, Department Committee on Graduate Students

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
WITHDRAWN
APR 06 1994
MIT LIBRARIES

Automated Design Synthesis of CMOS Operational Amplifiers

by

Ognen J. Nastov

Submitted to the Department of Electrical Engineering and Computer Science
January 19, 1994
In Partial Fulfillment of the Requirements for the Degree of
Master of Science

Abstract

In this thesis, I designed and implemented a system which autonomously designs optimal CMOS operational amplifiers. Throughout the design search, my system assembles the op amps by composing subcircuits. I evaluate op amps' performances by applying symbolic transformations and numerical techniques to the set of asserted approximate design equations. I also implemented a simulator-based performance evaluator that verifies the optimal designs.

My system controls the optimal design quest with a robust genetic algorithm. I integrate the searches through the topological and design parameter spaces in a novel, unique way. As a result, my system's design search is efficient, exhaustive, global, unbiased, and autonomous.

My system also designs op amps following an alternative strategy inspired by observation of human designers. In addition, my system autonomously generates such strategies.

Finally, I demonstrate my system's ability to autonomously design optimal CMOS op amps through a number of successful design experiments: a minimal area, high speed, micropower, high bandwidth, and high gain optimal designs.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for this research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-92-J-4097 and by the National Science Foundation under grant number MIP-9001651.

Thesis Supervisor: Gerald J. Sussman
Title: Matsushita Professor of Electrical Engineering

In Memory of my Father Jovan Nastov.

Acknowledgements

I would like to express my thanks to all the people who have contributed in one way or another to the realization of this thesis:

My thesis advisor Gerry Sussman for his continued support, skillful guidance, and valuable advice on all aspects of this thesis.

Hal Abelson, for his constant understanding and support, and for overseeing my early circuit optimization work.

Dr. Amar Bose, for his friendship, and for sharing his unique vision of the worlds of teaching, acoustics, and linear circuits with me.

Mark Jakiela, for introducing me to the world of genetic algorithms.

Greg Allen, for his friendship and review of an early draft of this thesis.

Phillip Alvelda, Bill Rozas, Chris Hanson, and other AI lab members for answering questions on MOS technology, scheme, and other topics.

The Neilys, the Browns, the Sarginsons, and others, for their friendship.

M.I.T. Varsity Swimming, for helping me stay in excellent physical shape.

The U.S. Army, for stationing 300 peacekeeping troops in Macedonia.

Finally, my mother Svetlana and sister Vesna, for their endless love, encouragement, and unwavering support in every part of my life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this thesis, I designed and implemented a system which autonomously designs optimal CMOS operational amplifiers. Throughout the design search, my system assembles the op amps by composing subcircuits. I evaluate op amps' performances by applying symbolic transformations and numerical techniques to the set of asserted approximate design equations. I also implemented a simulator-based performance evaluator that verifies the optimal designs.

My system controls the optimal design quest with a robust genetic algorithm. I integrate the searches through the topological and design parameter spaces in a novel, unique way. As a result, my system's design search is efficient, exhaustive, global, unbiased, and autonomous.

My system also designs op amps following an alternative strategy inspired by observation of human designers. In addition, my system autonomously generates such strategies.

Finally, I demonstrate my system's ability to autonomously design optimal CMOS op amps through a number of successful design experiments: a minimal area, high speed, micropower, high bandwidth, and high gain optimal designs.

1.1 Operational Amplifier Design Overview

The integration of analog and digital systems in the same package has become a widespread accepted engineering practice due to the continual increase in chip complexity. A number of feature-packed, sophisticated, and robust CAD tools can today be used to design the digital gate arrays, cells, and macro cells on such chips. In contrast, most analog circuit blocks still have to be designed by hand because: (1) analog circuits are not as easily decoupled into a set of simple interconnected basic blocks as digital circuits are; (2) device non-linearities are more important in analog circuit design, making the design process a knowledge-intensive task, and requiring experts with an accumulated set of techniques, approximation tricks, and circuit intuition; and (3) the performance specifications for analog circuits are more numerous, varied, and complicated than those for digital circuits.

The analog and digital system integration constrains the designers to using the same VLSI processing technology for both the analog and digital parts. The domination of the CMOS technology in today's monolithic digital systems has thus singled out CMOS as a dominant technology for integrated analog circuit systems.

One of the most important and widely used analog blocks is the operational amplifier (op amp), shown in Figure 1-1. The objective of the op amp is to amplify the difference between its two input signals v_+ and v_-. The op amp has to exhibit large amplification (gain) since its primary use is in implementing signal processing functions through the use of negative feedback.

Op amps are the functional core of many mixed analog/digital VLSI systems, particularly interface circuits, such as A/D and D/A converters, and switched capacitor filters.

Op amps are characterized by a large number of mutually conflicting performance specifications with variable priorities which depend on the op amp functionality within the analog system. For example, op amps used as comparators in A/D converters are primarily required to have high slew rates, small input offsets, and restricted area

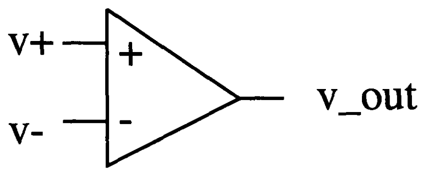


Figure 1-1: An operational amplifier

and power dissipation (an N-bit flash A/D converter employs 2^N op amps on a single chip). In contrast, op amps used in filtering applications or other negative-feedback configurations have to be compensated, i.e., are required to have high gain, and solid phase margins.

The human analog circuit designer goes through several phases when designing an op amp. In the first phase, the designer considers the intended application of the op amp, sets the performance specifications, and decides upon an appropriate circuit topology. In the second phase, the topology is sized and biased using analytical first-order design equations. In the third phase, the design is evaluated and optimized by adjustments of the design parameters and repeated circuit simulation. At the completion of the third phase, the op amp design is ready to be laid out and fabricated for testing. A survey of analog VLSI circuit designers [14] has indicated that the third design phase is the primary bottleneck of the analog circuit design process.

1.2 Previous VLSI Design Automation Efforts

During the past 10-15 years, varied methodologies have been employed in systems that automate the analog VLSI design process. These systems differ in three main areas: the circuit topology selection scheme (if any), the choice of a circuit performance evaluator, and the featured optimization method (if any). The optimization is typically carried out by having one performance function (such as the circuit chip area, power dissipation, or unity gain bandwidth) assume the role of an objective function, while the rest of the circuit performances are set up as constraints.

In 1983, Nye et al [20] developed *Delight.Spice*, a system which uses the data output from a general-purpose circuit simulator to evaluate each design point (iteration) of a fixed-topology analog circuit. This system employs a feasible directions optimization scheme to guide the iterations.

The advantage of using a circuit simulator as a performance evaluator lies in its accuracy, generality, and applicability to a broad range of circuits. Circuit simulators accurately predict circuit behavior because they employ complex and extensive device models, and use a variety of numerical techniques to solve the large systems of non-linear differential equations that describe the circuit. On the other hand, using circuit simulators as performance evaluators results in high computational intensity, especially for large circuits. In addition, circuit simulators can easily fail to converge for unorthodox sets of design parameters produced by the design search of the outer optimization driver loop.

Another approach in evaluating the circuit performance is to use approximate analytic 1-st order design equations. This approach is less intensive regarding computer resources, but requires development of such equations for each particular circuit topology or circuit building block. In addition, this scheme is less accurate than the method of using a circuit simulator. This in turn means that the solutions achieved in this scheme may in fact be only near-optimal.

Sometimes the accuracy of the equations used in this approach are enhanced by introducing fitting parameters. The results of a particular equation are fitted with the results obtained from repeated circuit simulations. Nevertheless, this fitting scheme has met limited success [15].

A number of authors developed systems built around a performance evaluator based on a set of approximate equations. In some cases, for example in the CIROP system (1984) by Ressler [23], the IDAC system (1987) by DeGrauwe et al [7], and in OASYS (1988) by Harjani et al [13], the authors design op amps by repeatedly composing subcircuit modules and solving the appropriate design equations until the designed circuit satisfies the design specifications, thus closely mimicking the human design approach. These systems do not feature any optimization schemes. The problem with this approach is that it does not yield optimal designs, although it does provide a good starting point for a circuit optimizer.

The approaches that employ an optimization algorithm include OPASYN by Koh (1989) [15], and the work done by Maulik and Carley (1991) [17]. Koh first heuristically selects an op amp topology from a library of four topologies, and then uses a steepest descent unconstrained method to optimize the design. Maulik and Carley also start from a fixed circuit topology and feature a constrained optimization method.

A common problem that arises from applying a general-purpose optimizer is the need for a good initial guess. When a circuit simulator is used as a performance evaluator, the computed performance functions tend to be noisy and non-differentiable. This constrains the optimization methods to be of zero-order only, which somewhat alleviates the problems of converging to a local minimum (or even to a local noise spike) when the optimization is started from a random initial point.

When a carefully crafted set of 1-st order design equations is used in circuit performance evaluation, the restriction on the order of the optimization methods can usually be lifted, and standard constrained NLP methods can then be used [17]. Nevertheless, the requirement for a good initial guess, and the problem of converging to a true local minimum remain. In addition, as mentioned above, at best constrained NLP methods arrive only in the vicinity of the actual optimal solution due to the inaccuracies in the approximate circuit performances from the analytic design equations.

In the cases when the performance functions are approximately computed, and are continuous and differentiable, in addition to the problem of converging to local minima, constrained optimization methods may also fail because they may be unable to find the feasible region from the initial point.

To the best of my knowledge, my application of *genetic algorithms* (GAs) to the problem of finding an optimal analog circuit design in this thesis and in my previous fixed topology optimal study [19] is original. Genetic algorithms are robust iterative general purpose zero-order unconstrained search strategies based on ideas from population genetics and natural selection [10]. They maintain a population that evolves and improves with each generation (iteration). These algorithms are intrinsically global in nature and thus somewhat alleviate the local minimum convergence problem. GAs do not require a good initial point to start the search. Genetic algorithms also exhibit potential for a full parallelization, which would make them even more attractive as the current trend of proliferation of parallel computer hardware continues.

In this thesis, I present a system for automated synthesis of CMOS op amps, whose main features are a modular subcircuit composition technology, an approximate design equations performance evaluator, and a genetic algorithm optimization scheme. The equation-based performance evaluator is built around a mixed symbolic and numerical algebraic equation solving scheme. My system also includes a circuit simulator-based performance evaluator, which verifies the designed op amps. This evaluator is based on a two-way interface to a circuit simulator which performs various op amp experiments, and a package that processes raw simulator data. A featured genetic algorithm evolves a population of topologically and parametrically varied op ams, and allows the survival and reproduction of the fittest among the op amps through a structured and partially randomized genetic information exchange. My system also features a design strategy inspired by observation of human designers synthesizing an op amp with a pre-selected modularly composed topology.

1.3 Organization of Thesis

Chapter 2 demonstrates an op amp design example. Chapter 3 presents the modular composition technology, a core idea in my system's design methodology. Chapter 4 describes the schemes used to evaluate the performance of the op amps. Chapter 5 details on the main genetic algorithm based design strategy. Chapter 6 discusses some additional design strategies featured in my system. Chapter 7 features additional op amp design experiments. In Chapter 8 I summarize my work and detail on possible future research directions.

The Appendix contains the circuit diagrams and the LISP code for all subcircuit modules (objects) and circuit components, a summary of the MOS device models, tabulation of process and design constants and parameters, and a summary overview of the op amp performance functions.

Chapter 2

Design Example: An Optimal High Speed Op Amp

Op amps used in interface circuits, such as A/D and D/A converters, are primarily required to be of high speed, i.e. to switch as fast as possible. This design goal can be achieved by maximizing the slew rate of the op amp. Thus I gave my system the set of design specifications shown in Table 2.1 and Table 2.2. The penalty forms and coefficients are further discussed in Chapter 5. The technology constants for the simple $5\mu m$ CMOS process used are given in Table B.2 in the Appendix.

Design Parameters	
Design Parameter(s)	Allowable Range
Transistor sizes M	$0.5 \leadsto 100$
Compensation capacitor C_C (pF)	$0.1 \leadsto 20$
Nulling resistor R_Z $(k\Omega)$	$0 \leadsto 100$
Bias voltages V_{BIAS} (V)	$V_{SS} \leadsto V_{DD}$

Table 2.1: High speed op amp design parameter specifications

Design Performance Specifications			
Performance	Specification	Penalty form	Penalty coefficient
CMOS Technology	SIMPLE $5\mu m$	–	–
Ambient temperature (C)	25	–	–
Positive power supply V_{DD} (V)	5	–	–
Negative power supply V_{SS} (V)	-5	–	–
MOS channel lengths L (μm)	10	–	–
Loading capacitance C_L (pF)	20	–	–
Active area (μm^2)	< 50000	linear	.01
DC power (mW)	< 20	linear	10
Input offset (mV)	$= 0$	linear	10
Input CMR (V)	$-3 \leadsto 3$	linear	10 & 30
Output swing (V)	$-3 \leadsto 3$	linear	10 & 30
Differential gain @ DC (dB)	> 80	linear	10
CMRR @ DC (dB)	> 80	linear	1
Unity gain bandwidth (MHz)	> 5	linear	60
Phase margin (degrees)	> 60	linear	10
PSRR+ @ DC (dB)	> 80	linear	1
PSRR– @ DC (dB)	> 80	linear	1
Slew rate $(V/\mu s)$	MAXIMIZE	goal: 100	5

Table 2.2: High speed op amp performance specifications

After randomly generating an initial population of thirty op amps with varied modularly composed topologies and parameter values, my system evolved three hundred generations of this population. The topologies, design parameters, and performances of the currently best op amps from a few selected generations in the design run are given in Figures 2-1 – 2-6, and Tables 2.3 – 2.14. (Note that the SPICE verification data is missing from most of these tables. The SPICE circuit simulator based performance evaluator could not verify the performances computed by the approximate equations based performance evaluator due to convergence failures for most of these op amps. The simulator's convergence failures are due to the unorthodoxy of the design parameters for these intermediate circuits, and are further discussed in Chapter 4.)

The genetic algorithm convergence profile is shown in Figure 2-8. The plot shows the continual converging improvement in the fitness function values of the best op amp in each generation. The fitness function summarizes the quality of the op amp's performance in a single number, and is further discussed in Chapter 5.

The fittest individual in the initial generation was a two-stage op amp with cascoded current mirror loads, cascoded differential input stage, and cascoded output stage, and no compensation scheme. During the initial fifty generations this topology turned out to be inadequate in satisfying the active area, phase margin, and voltage swing requirements. The evolution mechanisms thus engineered a sequence of better topological alternatives by virtue of introducing a nulling resistor compensation during the first 10 generations as well as removing the cascode devices from (1) the current mirror loads after thirty generations; (2) from the differential input stage (between generations 30 and 40); and (3) from the output stage (between generations 40 and 50).

The next sixty generations indicated the inability of the currently best op amp topology to achieve the required DC differential gain levels previously obtained by its cascoded predecessors. Between generations 110 and 120, the evolutionary mechanisms induced the expected change in the output stage genes in order to switch back to op amps with a cascoded output stage.

In the remaining 190 generations, this op amp topology, shown in Figure 2-7 enjoyed a stable adjustment in its design parameters inducing a continual improvement of its overall performance, and converged to its optimal parametric configuration summarized in Table 2.15. The achieved performance of this optimal high speed op amp is given in Table 2.16, and its simulation plots in Figure 2-9, Figure 2-10, and Figure 2-11. The fast switching capability of the designed op amp is readily visible from Figure 2-11.

As it can be seen from Table 2.16, the achieved design violates the design requirements for the input and output voltage swings, and the PSRRs. It is possible to instruct my design system to emphasize these requirements. I have chosen not to do so because in my experience (1) the voltage swing requirements are frequently underestimated by the approximate equations used in my system, and (2) the PSRRs, as well as other frequency domain performances are only crudely approximated with the 1-st order equations. I will discuss these and other related issues in the following chapters.

Table 2.16 indicates a large discrepancy between SPICE and design equations computed values for the output resistance. At the same time, the SPICE and design equations computed values of the small signal channel conductances for the devices in the cascoded output stage did match each other very well. I have observed this paradox in other circuits simulated with the particular version of the SPICE program I used (HPSPICE 3.4c). I thus concluded that the SPICE calculation of the output resistance is unreliable due to a bug in the program.

Table 2.16 also shows a discrepancy in the slew rate values. While the design equations determine the slew rate by considering the maximum rate of charging of the compensation capacitor by the first stage current, in this particular circuit it is the charging of the output capacitive load by the second stage current that sets the lower slew rate limit. The ratio of the second stage bias current to the loading capacitance gives a 51 $V/\mu s$ slew rate figure, which is much closer to the SPICE computed value of 41 $V/\mu s$.

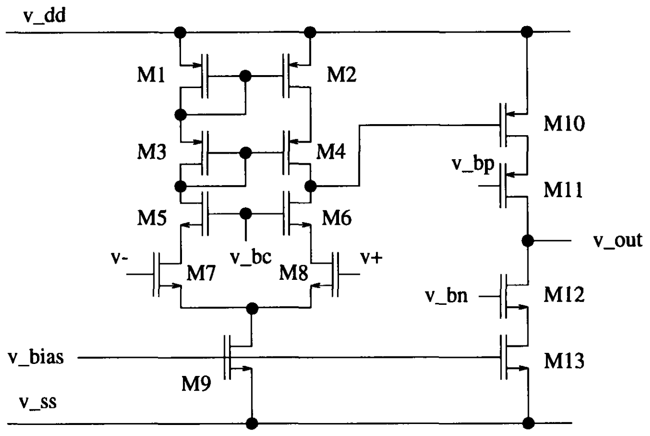


Figure 2-1: Topology of the best op amp in the initial generation of the high speed design run

Design Parameters		
Design Parameter	Allowable Range	Value
M_1, M_2	0.5 $\leadsto$ 100	36.4
M_3, M_4	0.5 $\leadsto$ 100	94.6
M_5, M_6	0.5 $\leadsto$ 100	83.0
M_7, M_8	0.5 $\leadsto$ 100	54.5
M_9	0.5 $\leadsto$ 100	22.1
M_{10}, M_{11}	0.5 $\leadsto$ 100	76.7
M_{12}, M_{13}	0.5 $\leadsto$ 100	56.3
V_{BIAS} (V)	-5 $\leadsto$ 5	-2.20
V_{BC} (V)	-5 $\leadsto$ 5	2.04
V_{BP} (V)	-5 $\leadsto$ 5	1.82
V_{BN} (V)	-5 $\leadsto$ 5	.43

Table 2.3: Design parameters for the best op amp in the initial generation of the high speed design run

Design Performance Summary			
Performance	Specification	Design	SPICE
Active area (μm^2)	< 50000	55364	–
DC power (mW)	< 20	21.62	–
Input offset (mV)	$= 0$	4.51	–
Input CMR (V)	-3 $\leadsto$ 3	-1.4 $\leadsto$ 1.7	–
Output swing (V)	-3 $\leadsto$ 3	-1.4 $\leadsto$ 0.5	–
Differential gain @ DC (dB)	> 80	109	–
CMRR @ DC (dB)	> 80	71	–
Unity gain bandwidth (MHz)	> 5	6.0	–
Phase margin (degrees)	> 60	3	-
PSRR+ @ DC (dB)	> 80	87	–
PSRR– @ DC (dB)	> 80	87	–
Slew rate (V/μs)	MAXIMIZE	78	–
Area of C_C (%)	–	0	–
Area of R_Z (%)	–	0	–
First stage bias current $I_{(CS)}$ (μA)	–	611	–
Second stage bias current $I_{(OS)}$ (μA)	–	1551	–
Output resistance R_{OUT} ($K\Omega$)	–	796	–
Dominant pole $pole_I$ (Hz)	–	275990	–
Output pole $pole_{II}$ (MHz)	–	9.9e-3	–
Compensation pole $pole_{III}$ (MHz)	–	–	–
Zero (MHz)	–	–	–

Table 2.4: Performance summary of the best op amp in the initial generation of the high speed design run

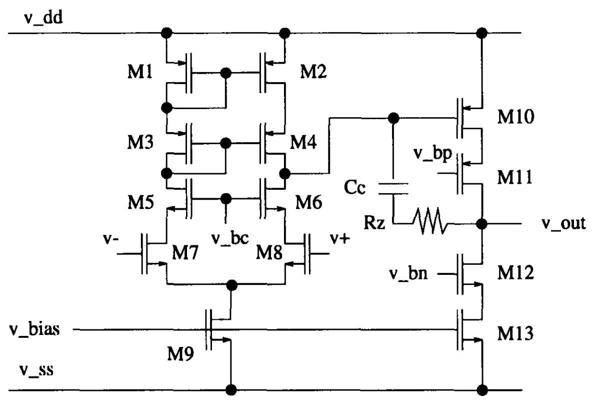


Figure 2-2: Topology of the best op amp in generation 10 of the high speed design run

Design Parameters		
Design Parameter	Allowable Range	Value
M_1, M_2	$0.5 \rightsquigarrow 100$	24.7
M_3, M_4	$0.5 \rightsquigarrow 100$	56.9
M_5, M_6	$0.5 \rightsquigarrow 100$	4.89
M_7, M_8	$0.5 \rightsquigarrow 100$	27.6
M_9	$0.5 \rightsquigarrow 100$	30.0
M_{10}, M_{11}	$0.5 \rightsquigarrow 100$	55.0
M_{12}, M_{13}	$0.5 \rightsquigarrow 100$	56.1
C_C (pF)	$0.1 \rightsquigarrow 20$	12.9
R_Z $(k\Omega)$	$0 \rightsquigarrow 100$	4.22
V_{BIAS} (V)	$-5 \rightsquigarrow 5$	-3.19
V_{BC} (V)	$-5 \rightsquigarrow 5$	1.40
V_{BP} (V)	$-5 \rightsquigarrow 5$	-2.42
V_{BN} (V)	$-5 \rightsquigarrow 5$	-4.57

Table 2.5: Design parameters of the best op amp in generation 10 of the high speed design run

Design Performance Summary			
Performance	Specification	Design	SPICE
Active area (μm^2)	< 50000	78739	–
DC power (mW)	< 20	4.74	–
Input offset (mV)	$= 0$	.06	–
Input CMR (V)	-3 $\leadsto$ 3	-2.6 $\leadsto$ 1.1	–
Output swing (V)	-3 $\leadsto$ 3	-3.4 $\leadsto$ 2.6	–
Differential gain @ DC (dB)	> 80	159	–
CMRR @ DC (dB)	> 80	118	–
Unity gain bandwidth (MHz)	> 5	3.4	–
Phase margin (degrees)	> 60	76	–
PSRR+ @ DC (dB)	> 80	131	–
PSRR– @ DC (dB)	> 80	131	–
Slew rate (V/μs)	MAXIMIZE	12.7	–
Area of C_C (%)	–	38	–
Area of R_Z (%)	–	11	–
First stage bias current $I_{(CS)}$ (μA)	–	165	–
Second stage bias current $I_{(OS)}$ (μA)	–	309	–
Output resistance R_{OUT} ($K\Omega$)	–	8136	–
Dominant pole $pole_I$ (Hz)	–	.04	–
Output pole $pole_{II}$ (MHz)	–	4.1	–
Compensation pole $pole_{III}$ (MHz)	–	29.5	–
Zero (MHz)	–	5.3	–

Table 2.6: Performance summary of the best op amp in generation 10 of the high speed design run

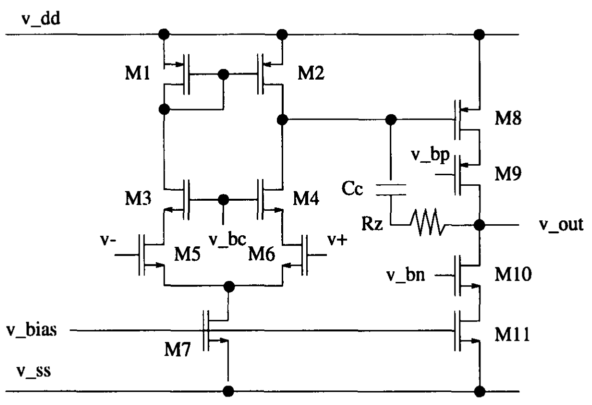


Figure 2-3: Topology of the best op amp in generation 30 of the high speed design run

Design Parameters		
Design Parameter	Allowable Range	Value
M_1, M_2	0.5 $\rightsquigarrow$ 100	24.7
M_3, M_4	0.5 $\rightsquigarrow$ 100	4.48
M_5, M_6	0.5 $\rightsquigarrow$ 100	27.6
M_7	0.5 $\rightsquigarrow$ 100	78.1
M_8, M_9	0.5 $\rightsquigarrow$ 100	55.2
M_{10}, M_{11}	0.5 $\rightsquigarrow$ 100	56.1
C_C (pF)	0.1 $\rightsquigarrow$ 20	10.5
R_Z $(k\Omega)$	0 $\rightsquigarrow$ 100	4.27
V_{BIAS} (V)	-5 $\rightsquigarrow$ 5	-3.19
V_{BC} (V)	-5 $\rightsquigarrow$ 5	1.39
V_{BP} (V)	-5 $\rightsquigarrow$ 5	-2.41
V_{BN} (V)	-5 $\rightsquigarrow$ 5	-2.17

Table 2.7: Design parameters for the best op amp in generation 30 of the high speed design run

Design Performance Summary			
Performance	Specification	Design	SPICE
Active area (μm^2)	< 50000	67635	–
DC power (mW)	< 20	7.39	–
Input offset (mV)	$= 0$	2.83	–
Input CMR (V)	-3 $\rightsquigarrow$ 3	-2.2 $\rightsquigarrow$ 1.1	–
Output swing (V)	-3 $\rightsquigarrow$ 3	-3.4 $\rightsquigarrow$ 2.6	–
Differential gain @ DC (dB)	> 80	113	–
CMRR @ DC (dB)	> 80	119	–
Unity gain bandwidth (MHz)	> 5	6.8	–
Phase margin (degrees)	> 60	65	–
PSRR+ @ DC (dB)	> 80	85	–
PSRR− @ DC (dB)	> 80	85	–
Slew rate (V/μs)	MAXIMIZE	41	–
Area of C_C (%)	–	36	–
Area of R_Z (%)	–	13	–
First stage bias current $I_{(CS)}$ (μA)	–	430	–
Second stage bias current $I_{(OS)}$ (μA)	–	309	–
Output resistance R_{OUT} ($K\Omega$)	–	8145	–
Dominant pole $pole_I$ (Hz)	–	15.4	–
Output pole $pole_{II}$ (MHz)	–	4.15	–
Compensation pole $pole_{III}$ (MHz)	–	29.0	–
Zero (MHz)	–	6.43	–

Table 2.8: Performance summary of the best op amp in generation 30 of the high speed design run

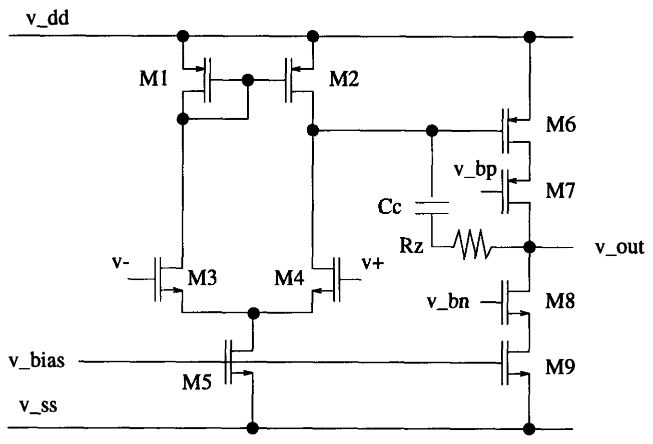


Figure 2-4: Topology of the best op amp in generation 40 of the high speed design run

Design Parameters		
Design Parameter	Allowable Range	Value
M_1, M_2	0.5 $\rightsquigarrow$ 100	24.7
M_3, M_4	0.5 $\rightsquigarrow$ 100	27.1
M_5	0.5 $\rightsquigarrow$ 100	53.1
M_6, M_7	0.5 $\rightsquigarrow$ 100	55.2
M_8, M_9	0.5 $\rightsquigarrow$ 100	68.6
C_C (pF)	0.1 $\rightsquigarrow$ 20	10.5
R_Z $(k\Omega)$	0 $\rightsquigarrow$ 100	4.66
V_{BIAS} (V)	-5 $\rightsquigarrow$ 5	-2.57
V_{BP} (V)	-5 $\rightsquigarrow$ 5	-2.26
V_{BN} (V)	-5 $\rightsquigarrow$ 5	-2.17

Table 2.9: Design parameters for the best op amp in generation 40 of the high speed design run

Design Performance Summary			
Performance	Specification	Design	SPICE
Active area (μm^2)	< 50000	67657	–
DC power (mW)	< 20	21.1	16.0
Input offset (mV)	$= 0$	-3.4	8.97
Input CMR (V)	$-3 \leadsto 3$	$-1.2 \leadsto 2.8$	$-4.0 \leadsto 4.1$
Output swing (V)	$-3 \leadsto 3$	$-2.1 \leadsto .4$	$-4.8 \leadsto 4.1$
Differential gain @ DC (dB)	> 80	95	77
CMRR @ DC (dB)	> 80	101	73
Unity gain bandwidth (MHz)	> 5	9.86	9.73
Phase margin (degrees)	> 60	89	38
PSRR+ @ DC (dB)	> 80	72	74
PSRR– @ DC (dB)	> 80	72	37
Slew rate (V/μs)	MAXIMIZE	87.6	46.6
Area of C_C (%)	–	36	–
Area of R_Z (%)	–	14	–
First stage bias current $I_{(CS)}$ (μA)	–	922	937
Second stage bias current $I_{(OS)}$ (μA)	–	1192	658
Output resistance R_{OUT} ($K\Omega$)	–	1119	192
Dominant pole $pole_I$ (Hz)	–	182	–
Output pole $pole_{II}$ (MHz)	–	8.17	–
Compensation pole $pole_{III}$ (MHz)	–	26.6	–
Zero (MHz)	–	4.1	–

Table 2.10: Performance summary of the best op amp in generation 40 of the high speed design run

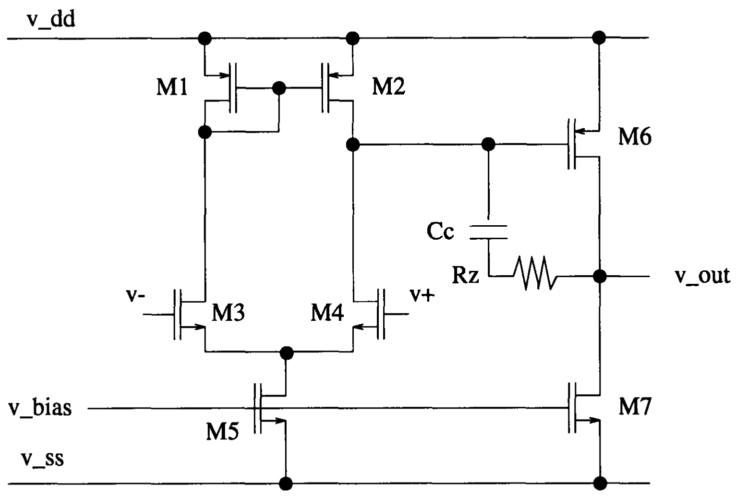


Figure 2-5: Topology of the best op amp in generation 50 of the high speed design run

Design Parameters		
Design Parameter	Allowable Range	Value
M_1, M_2	0.5 $\rightsquigarrow$ 100	24.6
M_3, M_4	0.5 $\rightsquigarrow$ 100	27.1
M_5	0.5 $\rightsquigarrow$ 100	53.1
M_6	0.5 $\rightsquigarrow$ 100	55.2
M_7	0.5 $\rightsquigarrow$ 100	56.2
C_C (pF)	0.1 $\rightsquigarrow$ 20	10.7
R_Z $(k\Omega)$	0 $\rightsquigarrow$ 100	4.27
V_{BIAS} (V)	-5 $\rightsquigarrow$ 5	-3.27

Table 2.11: Design parameters for the best op amp in generation 50 of the high speed design run

Design Performance Summary			
Performance	Specification	Design	SPICE
Active area (μm^2)	< 50000	55810	–
DC power (mW)	< 20	4.9	5.1
Input offset (mV)	$= 0$	.33	.46
Input CMR (V)	$-3 \rightsquigarrow 3$	$-2.6 \rightsquigarrow 3.9$	$-4.0 \rightsquigarrow 4.7$
Output swing (V)	$-3 \rightsquigarrow 3$	$-4.3 \rightsquigarrow 3.9$	$-4.9 \rightsquigarrow 4.9$
Differential gain @ DC (dB)	> 80	75	77
CMRR @ DC (dB)	> 80	81	87
Unity gain bandwidth (MHz)	> 5	4.93	3.66
Phase margin (degrees)	> 60	63	54
PSRR+ @ DC (dB)	> 80	79	81
PSRR− @ DC (dB)	> 80	85	60
Slew rate (V/μs)	MAXIMIZE	22.3	13.9
Area of C_C (%)	–	44	–
Area of R_Z (%)	–	15	–
First stage bias current $I_{(CS)}$ (μA)	–	238	243
Second stage bias current $I_{(OS)}$ (μA)	–	252	265
Output resistance R_{OUT} ($K\Omega$)	–	132	136
Dominant pole $pole_I$ (Hz)	–	852	–
Output pole $pole_{II}$ (MHz)	–	3.76	–
Compensation pole $pole_{III}$ (MHz)	–	29.0	–
Zero (MHz)	–	6.9	–

Table 2.12: Performance summary of the best op amp in generation 50 of the high speed design run

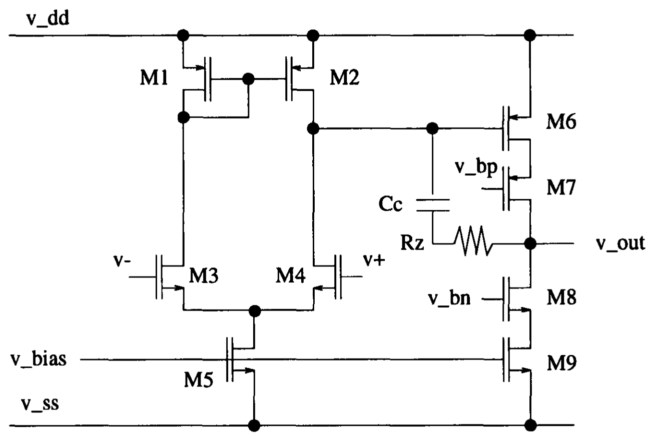


Figure 2-6: Topology of the best op amp in generation 120 of the high speed design run

Design Parameters		
Design Parameter	Allowable Range	Value
M_1, M_2	0.5 $\rightsquigarrow$ 100	14.6
M_3, M_4	0.5 $\rightsquigarrow$ 100	27.1
M_5	0.5 $\rightsquigarrow$ 100	32.4
M_6, M_7	0.5 $\rightsquigarrow$ 100	52.4
M_8, M_9	0.5 $\rightsquigarrow$ 100	65.9
C_C (pF)	0.1 $\rightsquigarrow$ 20	4.98
R_Z $(k\Omega)$	0 $\rightsquigarrow$ 100	5.05
V_{BIAS} (V)	-5 $\rightsquigarrow$ 5	-2.66
V_{BP} (V)	-5 $\rightsquigarrow$ 5	3.3
V_{BN} (V)	-5 $\rightsquigarrow$ 5	-1.98

Table 2.13: Design parameters for the best op amp in generation 120 of the high speed design run

Design Performance Summary			
Performance	Specification	Design	SPICE
Active area (μm^2)	< 50000	51269	–
DC power (mW)	< 20	15.1	–
Input offset (mV)	$= 0$	-2.1	–
Input CMR (V)	-3 $\rightsquigarrow$ 3	-1.6 $\rightsquigarrow$ 2.9	–
Output swing (V)	-3 $\rightsquigarrow$ 3	-2.3 $\rightsquigarrow$ .6	–
Differential gain @ DC (dB)	> 80	98	–
CMRR @ DC (dB)	> 80	104	–
Unity gain bandwidth (MHz)	> 5	15.3	–
Phase margin (degrees)	> 60	58	–
PSRR+ @ DC (dB)	> 80	75	–
PSRR– @ DC (dB)	> 80	75	–
Slew rate (V/μs)	MAXIMIZE	100	–
Area of C_C (%)	–	22	–
Area of R_Z (%)	–	20	–
First stage bias current $I_{(CS)}$ (μA)	–	498	–
Second stage bias current $I_{(OS)}$ (μA)	–	1013	–
Output resistance R_{OUT} ($K\Omega$)	–	1394	–
Dominant pole $pole_I$ (Hz)	–	186	–
Output pole $pole_{II}$ (MHz)	–	7.33	–
Compensation pole $pole_{III}$ (MHz)	–	25.8	–
Zero (MHz)	–	8.06	–

Table 2.14: Performance summary of the best op amp in generation 120 of the high speed design run

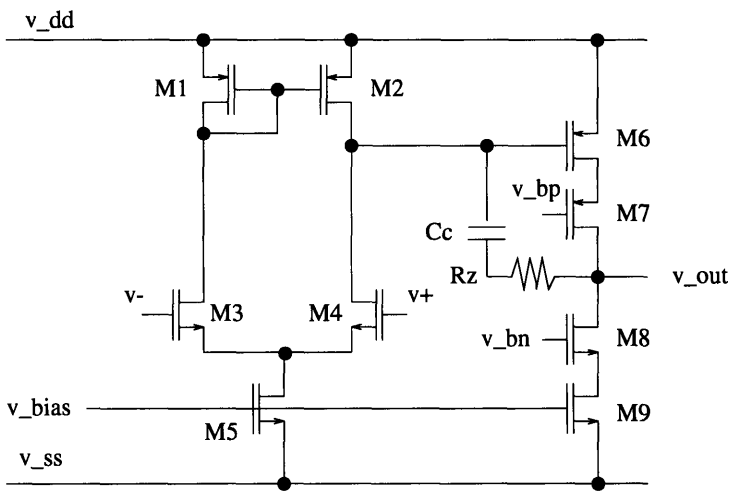


Figure 2-7: Optimal high-speed op amp topology

Optimal Design Parameters		
Design Parameter	Allowable Range	Optimal Value
M_1, M_2	0.5 $\leadsto$ 100	13.2
M_3, M_4	0.5 $\leadsto$ 100	13.6
M_5	0.5 $\leadsto$ 100	33.0
M_6, M_7	0.5 $\leadsto$ 100	52.9
M_8, M_9	0.5 $\leadsto$ 100	66.1
C_C (pF)	0.1 $\leadsto$ 20	5.05
R_Z $(k\Omega)$	0 $\leadsto$ 100	4.88
V_{BIAS} (V)	-5 $\leadsto$ 5	-2.65
V_{BP} (V)	-5 $\leadsto$ 5	-1.35
V_{BN} (V)	-5 $\leadsto$ 5	-0.19

Table 2.15: Optimal design parameters for a high speed op amp

Optimal Design Performance Summary			
Performance	Specification	Design	SPICE
Active area (μm^2)	< 50000	48729	–
DC power (mW)	< 20	15.3	15.5
Input offset (mV)	$= 0$	-1.6e-2	0.25
Input CMR (V)	-3 $\leadsto$ 3	-1.2 $\leadsto$ 2.8	-4.0 $\leadsto$ 3.5
Output swing (V)	-3 $\leadsto$ 3	-2.3 $\leadsto$ 0.6	-4.8 $\leadsto$ 3.8
Differential gain @ DC (dB)	> 80	95	101
CMRR @ DC (dB)	> 80	101	77
Unity gain bandwidth (MHz)	> 5	10.8	8.5
Phase margin (degrees)	> 60	65	43
PSRR+ @ DC (dB)	> 80	72	69
PSRR− @ DC (dB)	> 80	72	56
Slew rate (V/μs)	MAXIMIZE	101 (51)	41
Area of C_C (%)	–	24	–
Area of R_Z (%)	–	20	–
First stage bias current $I_{(CS)}$ (μA)	–	509	518
Second stage bias current $I_{(OS)}$ (μA)	–	1020	1040
Output resistance R_{OUT} ($K\Omega$)	–	1384	2521 (?)
Dominant pole $pole_I$ (Hz)	–	187	–
Output pole $pole_{II}$ (MHz)	–	7.39	–
Compensation pole $pole_{III}$ (MHz)	–	26.5	–
Zero (MHz)	–	8.28	–

Table 2.16: Optimal high speed op amp performance summary

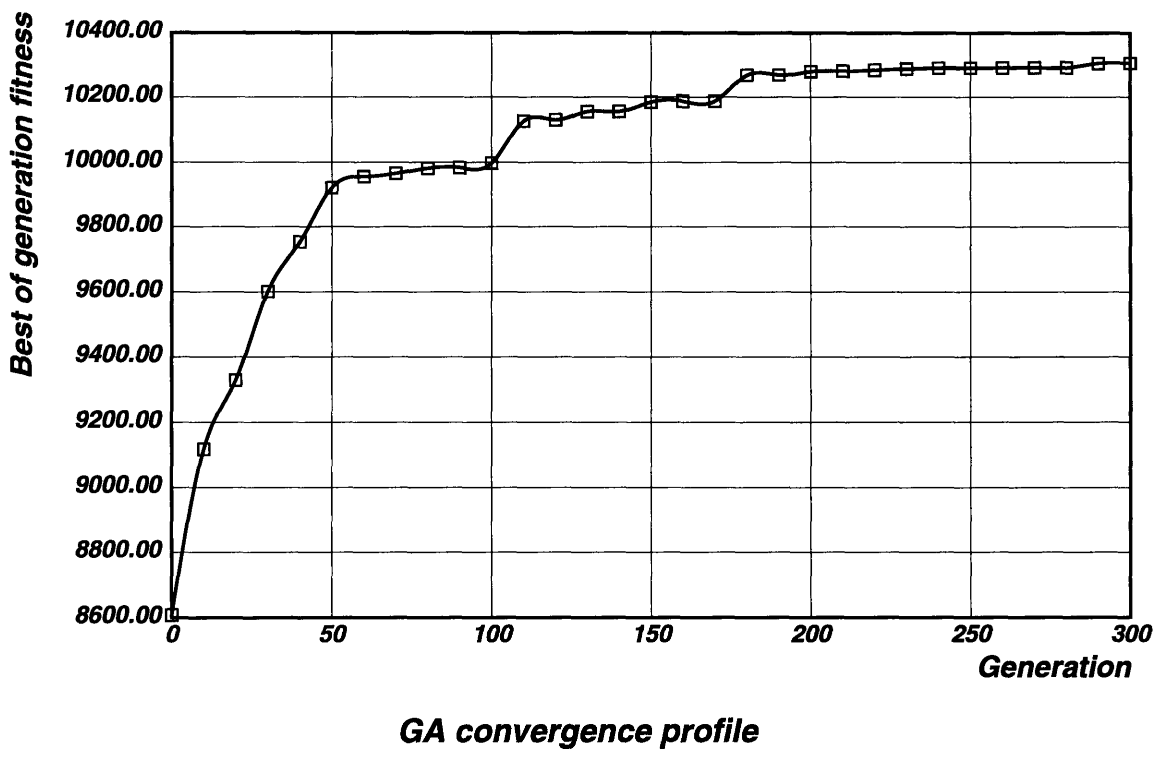


Figure 2-8: Genetic algorithm convergence profile for the high speed op amp design run

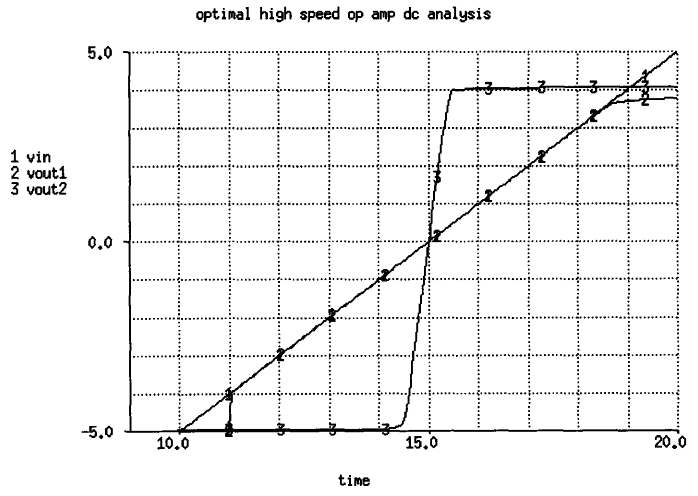


Figure 2-9: DC characteristics of the optimal high speed op amp

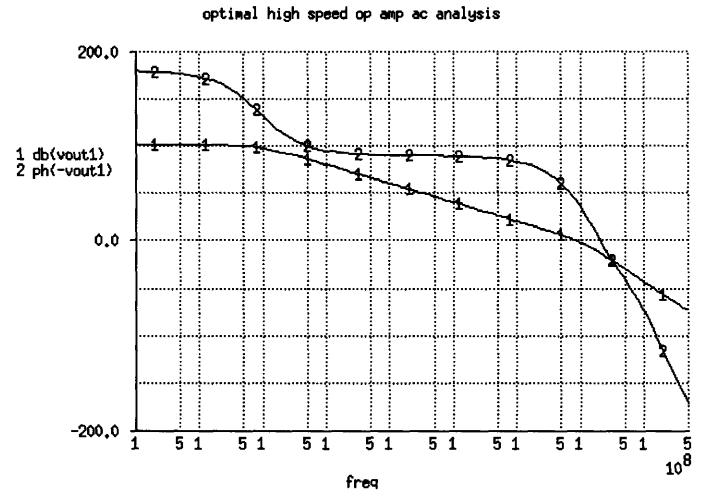


Figure 2-10: Frequency domain behavior of the optimal high speed op amp

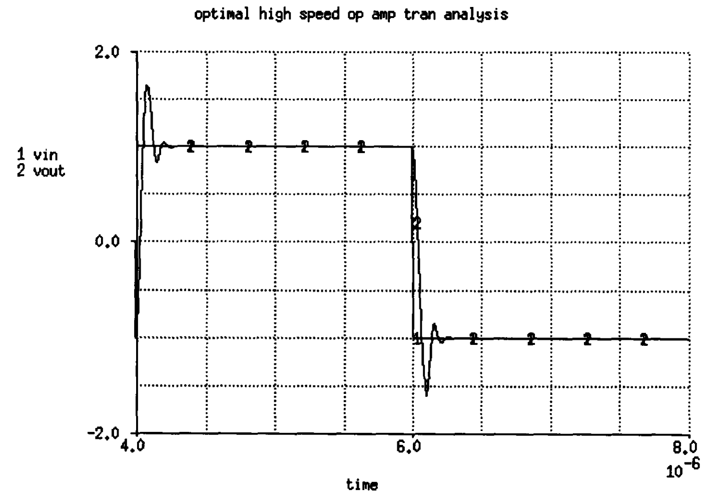


Figure 2-11: Switching characteristics of the optimal high speed op amp

Chapter 3

Modular Composition Scheme

One of the core ideas that underlies the op amp design methodology in my system is the modular composition of basic subcircuit blocks (modules).

3.1 Overview

Selecting an op amp topology for a given op amp design problem is not an easy task. Many design tools, such as OPASYN [15], OASYS [13], and CIROP [23] attempt to imitate the human designers. In particular, OPASYN heuristically prunes a decision tree to select one of the four fixed op amp topologies in its library. In contrast, OASYS, CIROP, and my system construct op amp topologies by composing subcircuit modules.

The subcircuit module composition approach helps in formalizing and generalizing the design methodology. The language that is used to manipulate the design knowledge (in my case, the equation-solving language described in Chapter 4) is separated from the knowledge itself (the equations found in the analysis/design knowledge part of my subcircuit objects). The knowledge database can easily be reused to implement other design schemes. For example, the design knowledge in my system, besides representing the core of our approximate performance evaluator described in Chapter 4, can also be used to implement two alternative design routes summarized in Chapter 6.

From a practical standpoint, the primary advantage of automated op amp design systems based on hierarchical composition of basic blocks is their ability of composing a large number of different op amp topologies. For example, the design space in my system features 56 distinct practical device-level topologies. An addition of a new basic block to the system doubles the number of practical topologies in most cases.

Unfortunately, operational amplifiers (and all analog circuits in general) are not as easily decoupled into a set of interconnected modular basic blocks compared to digital circuits. While the flow of information between digital basic blocks is limited (to the first order) to two digital voltage levels, the modules in a typical op amp communicate through a continuum of voltages and currents. In general, the less information flows between the levels of hierarchy, the better the decomposition (i.e. the easier the design task).

While there is no universally accepted decomposition of op amps, a common way of decomposing a class of op amp topologies is a breakup into an input stage, an output stage, and a compensation module, where the input stage is composed of a current mirror, differential stage, and a current source. Each of the modules can be further decoupled into its constituent interconnected components, such as MOS transistors, capacitors, resistors, voltage biases (sources), and wires. The design process implemented in my system is based on this particular decomposition scheme, shown in Figure 3-1.

The op amps designed by my system's composition scheme are "unbuffered", or OTAs (operational transconductance amplifiers) due to their inability to drive small resistive loads or large capacitive loads. A buffer stage can be added in order to alleviate this problem. I have not included a buffer stage design in my scheme since the primary target application of my op amps is as an integral on-chip sub-system within a larger VLSI CMOS analog-digital system.

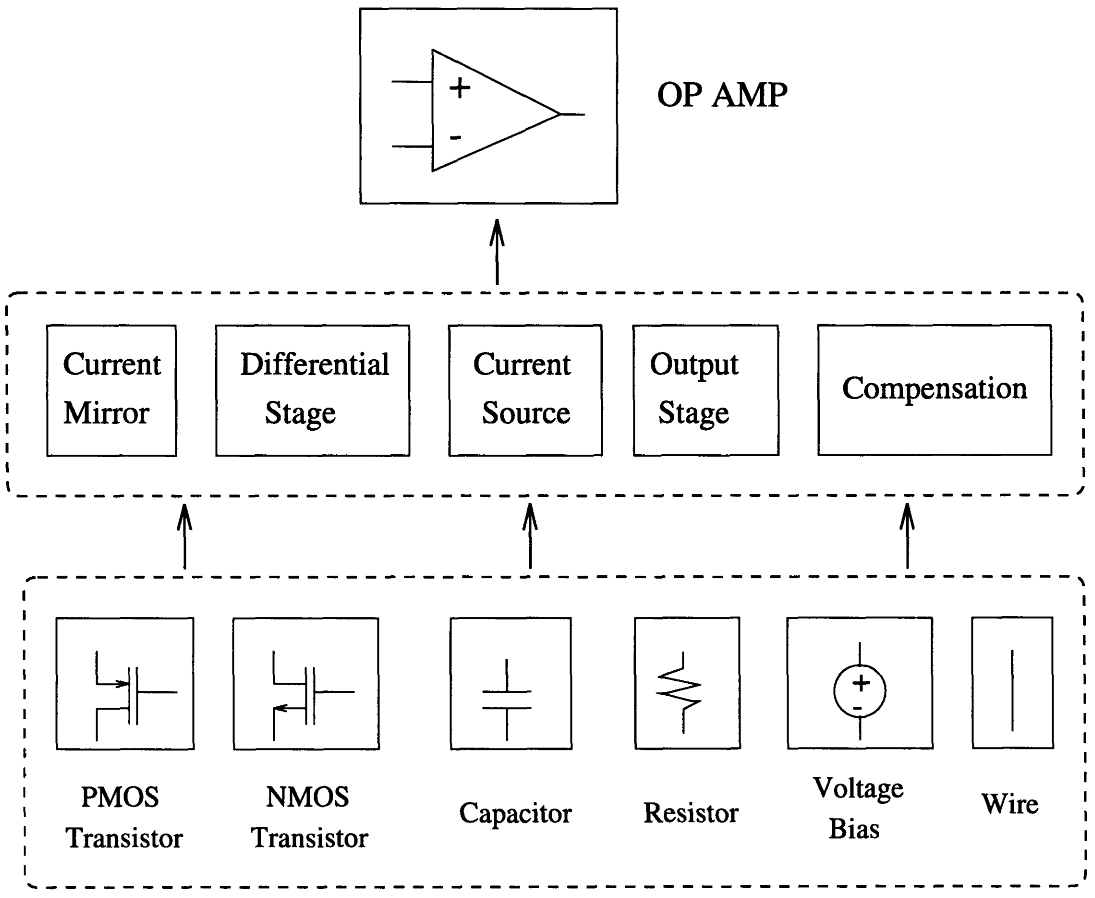


Figure 3-1: Modular composition of op amps

3.2 Op Amp Topological Template

The block-level structure of the op amps that can be designed by our program is given in Figure 3-2.

The first stage of my general op amp topology consists of a differential stage (i.e. a differential input pair), a current source (in fact, a sink), and a current mirror loading module. The first stage provides a differential amplification of the input signals v_+ and v_-. The single ended output of this stage, v_d, continues into the second, output stage (a common-source amp with a current sink). The second stage functions as a class-A inverting amplifier. The compensation of this op amp is performed

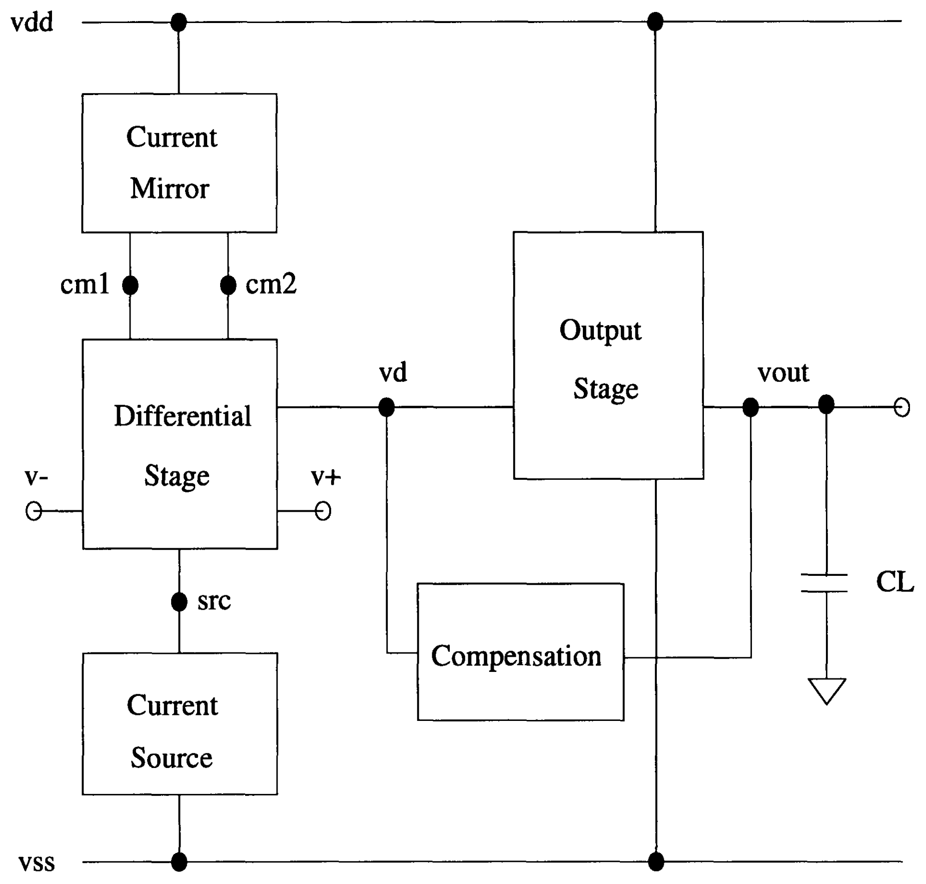


Figure 3-2: Topological template for op amps

by the compensation module (a pole-splitting Miller capacitor and an optional zero nulling resistor). The loading of the output is assumed to be purely capacitive, and is represented by the capacitor C_L.

Each of the subcircuit modules in my op amps can be designed in one or more block styles. The module styles are summarized in Figure 3-3. Diagrams of the device-level structure of each particular block style are given in the Appendix.

The only two unorthodox block styles are the *nil* versions of the output stage and the compensation module. The nil output stage block is composed of a single wire, and it enables my system to design a one stage op amp. The nil compensation module is an empty block used in one stage designs, or two stage designs with weak

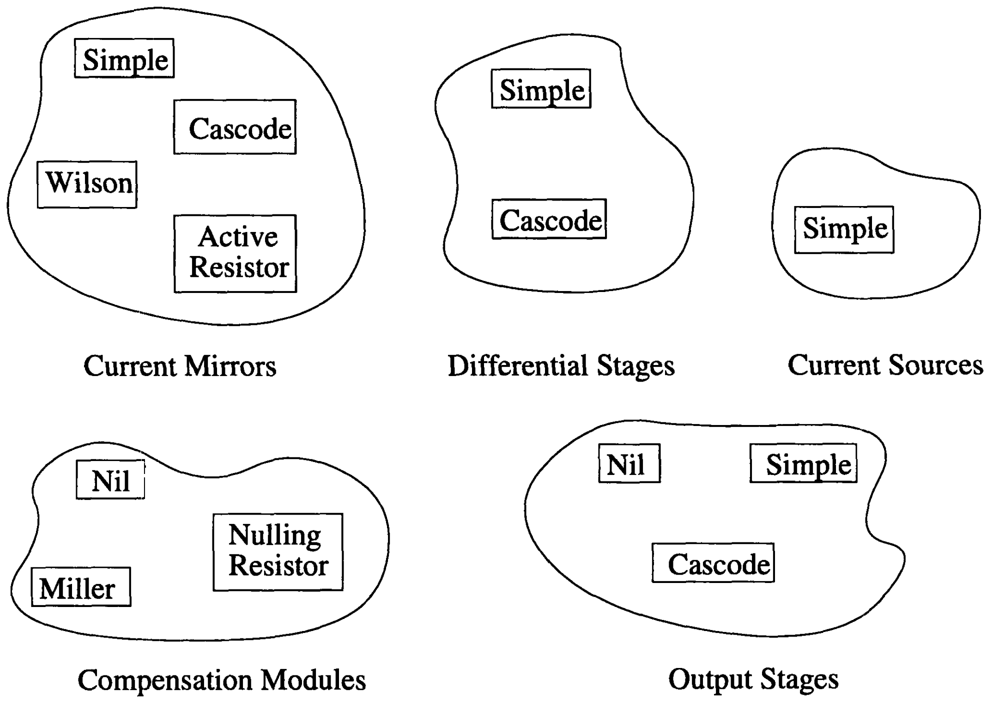


Figure 3-3: Subcircuit styles summary

compensation requirements.

While the total number of different op amp topologies that can be designed in my system is 72, they include 16 non-practical topologies which feature a non-nil compensation module with a nil output stage (which shorts out the compensation module and renders it ineffective).

3.2.1 Topology Selection Strategy

The decisions OPASYN undertakes in selecting an op amp topology from its library are based on only two performance requirements: the differential gain and the PSRR. If the gain requirement is larger than some threshold, OPASYN selects a two-stage configuration. Similarly, in case of a stricter PSRR specification, the system chooses a cascoded differential stage topology.

In other automated design systems, such as CIROP and OASYS, the block style selection is also carried by a set of heuristic rules. Usually, the simplest blocks are selected first. Next, an attempt to design and optimize an op amp satisfying the design requirements is carried out. If the design fails to satisfy the design specifications due to topological shortcomings, a set of heuristic failure rules are invoked and the topology is modified by a replacement of one or more subcircuit blocks with blocks of different style.

However, a successful implementation of good failure and redesign rules is a difficult task. The designer of these rules has to achieve a balanced trade-off between (1) the bias of the rules towards particular topologies for reasons of efficiency, and (2) the ability of the rules to effectively search through the entire topological space. In addition, no matter how well crafted these failure and re-design rules may be, this heuristic strategy could be inefficient due to many possible repeated failures of the designs.

In my system, the structure of the featured genetic optimization algorithm enables a transformation of the topological selection from being a heuristic rule-based selection process to simply being a design parameter in the design optimization phase. My novel approach, detailed in Chapter 5, guarantees a more efficient, exhaustive, and intrinsically parallel search of the topological design space.

Also, in my opinion, my system features a more effective bias and convergence of the design style towards the best topology to the design problem. The design style convergence in my topologically diverse genetic pool of op amps is biased by natural selection rules which are based on the evaluated performance of each op amp. From generation to a generation, my genetic pool can continue to maintain topological diversity. In comparison, the failure and redesign rules in other automated systems are performed in linear sequential fashion, and may thus introduce unwanted bias in the topological convergence process.

3.3 Object-Oriented Implementation Approach

My software approach in implementing the hierarchical composition of subcircuit blocks is object-oriented.

The top level of the hierarchy is represented by an op amp object. The op amp module inherits from its five constituent modules. Each of the modules are also objects that are constructed from the six possible circuit components.

To create a particular op amp topology, the genetic algorithm driver (or the user) simply assert

```
(opamp <opamp-name> <current-mirror> <diff-stage>
       <current-source> <output-stage> <compensation>)
```

which returns an op amp object.

3.3.1 Object Structure

The op amp object, its constituent subcircuit objects, and the circuit component objects have a similar structure. Each contains the following information:

1. *Type/name information*

2. *Composition/VLSI knowledge*

3. *Analysis/model knowledge*

The type/name information identifies the object and specifies its function. The composition/VLSI knowledge either explicitly details the interconnections between the constituent elements of a subcircuit module or details the physical VLSI makeup of a particular circuit component. For the subcircuit module objects the analysis/model knowledge declares the design parameters and features a set of relevant design equations. For the circuit component objects, the analysis/model knowledge contains model equations that describe that component's behavior. Figure 3-4 lists the code for one subcircuit module - the simple differential stage (SDS). Figure 3-5 shows large and small signal circuit diagrams that correspond to the SDS module. The code listings for the remaining subcircuit and component objects are given in the Appendix.

```
(define simple-diff-stage
  (lambda (m)
    (case m
      ;; Type/name information:
      ((type) 'diff-stage)
      ((name) 'simple-diff-stage)
      ;; Composition/VLSI knowledge:
      ((make) (lambda (model w l)
                (list '(* simple differential stage)
                      ((nmos-tran 'make)
                       'sds1 'cm1 'vp 'src 'vss model w l)
                      ((nmos-tran 'make)
                       'sds2 'cm2 'vn 'src 'vss model w l)
                      ((wire 'make)
                       'sds1 'cm2 'vd))))
      ;; Analysis/model knowledge:
      ((params) '(sds-w sds-l))
      ((design) '((= ds-area (* 2 sds-w sds-l))
                  (= v-cm-to-vp
                     (- (vtn 0)))
                  (= v-vp-to-src
                     (+ (sqrt (/ (* 2 ds-i) kn sds-sz)) (vtn 0)))
                  (= ds-gm (gmn sds-sz ds-i kn))
                  (= sds-gds (gdsn sds-l ds-i))
                  (= r-vd-to-src (/ 1 sds-gds))
                  (= sds-w (* sds-sz (leff sds-l ldn)))))
      (else (error "Unknown message -- simple-diff-stage.")))))
```

Figure 3-4: Code listing for the simple differential stage object

The composition/VLSI knowledge featured in the subcircuit modules and circuit components provides for the capability of producing a source deck segment for a circuit simulator (written in a SPICE syntax). The top level op amp object not only combines these deck segments, but also adds an appropriate sequence of simulator analysis commands. In op amps containing an inverting non-nil output stage, the op amp object performs a label swap of the input nodes v_+ and v_-. The automated creation of a simulator source deck interfaces the object composition phase of my system with the simulator-based performance evaluator which is used to verify the

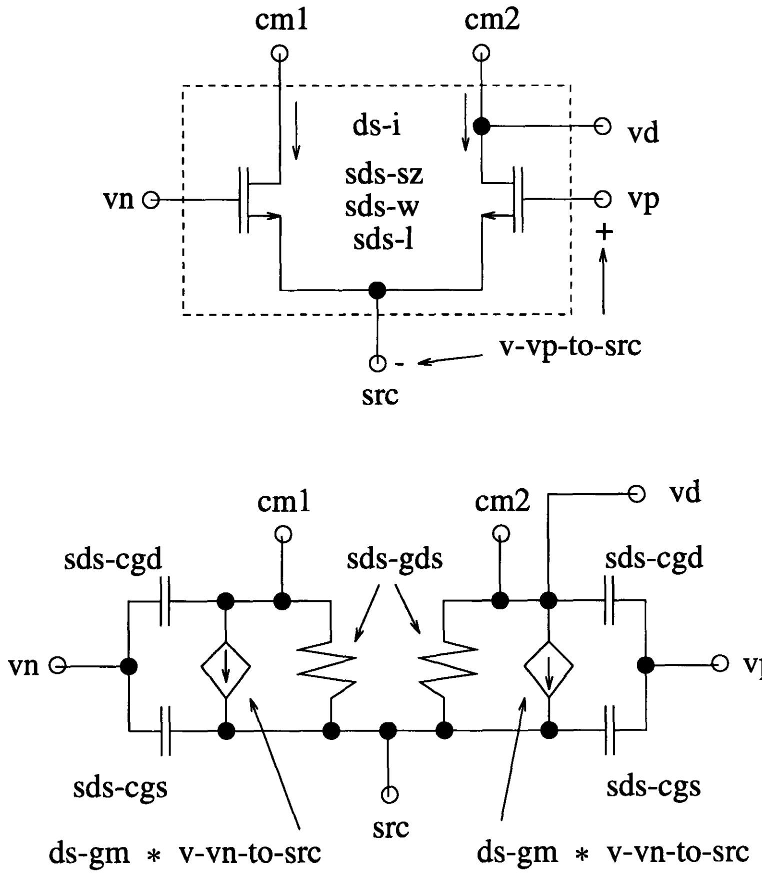


Figure 3-5: Large signal (top) and small signal (bottom) circuit diagrams of the simple differential stage module

optimized designs.

For example, the expression

```
((nmos-tran 'make)
 'sds1 'cm1 'vp 'src 'vss model w l)
```

found in the composition/VLSI knowledge list of the object shown in Figure 3-4 constructs one of the differential input stage transistors. Assuming that the `model` parameter is "5n", the channel width `w` $= 43\mu m$, and the channel length `l` $= 10\mu m$, this expression produces the following circuit simulator deck entry:

```
msds1 %cm1 %vp %src %vss nmos5n w=43u l=10u ad=648p as=648p
+ pd=116u ps=116u nrd=.35 nrs=.35
```

The analysis/model knowledge of the subcircuit modules and circuit components represented by the set of approximate design and model equations is the basis of the design equations based performance evaluator in my system.

For example, the following equation, taken from the analysis/design knowledge list of design equations of the simple differential stage (SDS) subcircuit object in Figure 3-4, calculates the active area of that subcircuit:

```
(= ds-area (* 2 sds-w sds-l))
```

where `ds-area` is the total active transistor area, and `sds-w` and `sds-l` are the channel width and length of the two matched devices in the module.

The equation

```
(= v-cm-to-vp (- (vtn 0)))
```

computes the minimum allowed DC voltage drop between `cm1` (or `cm2`) and `vp` nodes in the SDS subcircuit, so that the leftmost NMOS transistor in the subcircuit stays in saturation. The subexpression `(vtn 0)` computes the threshold voltage of an NMOS transistor assuming a zero source-to-bulk voltage.

The equation

```
(= v-vp-to-src (+ (sqrt (/ (* 2 ds-i) kn sds-sz)) (vtn 0)))
```

calculates the minimal DC voltage drop between nodes `vp` and `src` such that the leftmost NMOS transistor remains in the saturation region. The variable `ds-i` is the DC bias current flowing through each of the branches of the SDS subcircuit, `kn` is the transconductance parameter for NMOS transistors, and `sds-sz` is the size of each of the transistors in the SDS module.

The equations

```
(= ds-gm (gmn sds-sz ds-i kn))
(= sds-gds (gdsn sds-l ds-i))
(= r-vd-to-src (/ 1 sds-gds))
```

compute the small signal stage transconductance `ds-gm`, the small signal device conductance `sds-gds`, and small signal resistance `r-vd-to-src` of the rightmost transistor in the stage. The subexpressions `(gmn ...)` and `(gdsn ...)` are recursively expanded by the equation solver into the following model equations:

```
(= (gmn w/l id kn) (sqrt (* 2 kn id w/l)))
(= (gdsn l id) (* (lambdan l) id))
(= (lambdan l) (* lambdan-maw (/ (leff min-active-width ldn)
                                 (leff l ldn))))
(= (leff l ld) (- l (* 2 ld)))
```

where the subexpression `(lambda ...)` computes the channel-length modulation parameter, while `(leff ...)` calculates the effective channel width.

Finally, the equation

```
(= sds-w (* sds-sz (leff sds-l ldn)))
```

computes the actual channel width of the two devices in the SDS subcircuit.

The techniques I use to transform the design equations by subexpression expansions, and the methods I use to solve the design equations are discussed in Chapter 4.

Chapter 4

Op Amp Performance Evaluation

The objective of a circuit performance evaluator is to compute various performance functions from the set of design parameters as shown in Figure 4-1. Many of the performance functions for my op amps are non-linear functions of the design parameters, and can be accurately determined only by a circuit simulation. My system thus features two op amp performance evaluators: an approximate design equation based evaluator, and a simulator based evaluator.

4.1 Approximate Design Equation Performance Evaluator

The approximate design equation performance evaluator is based on the set of first order analytic design equations that are asserted from the top-level op amp object, its constituent subcircuit modules, and the circuit components featured in the design. These equations express approximate functional dependencies of the performances of the op amp on the design parameters.

The primary advantage of my approximate equations based evaluator is its much greater computing speed when compared to a simulator based scheme. In addition, analyzing the structure of the equations may aid in determining the best design

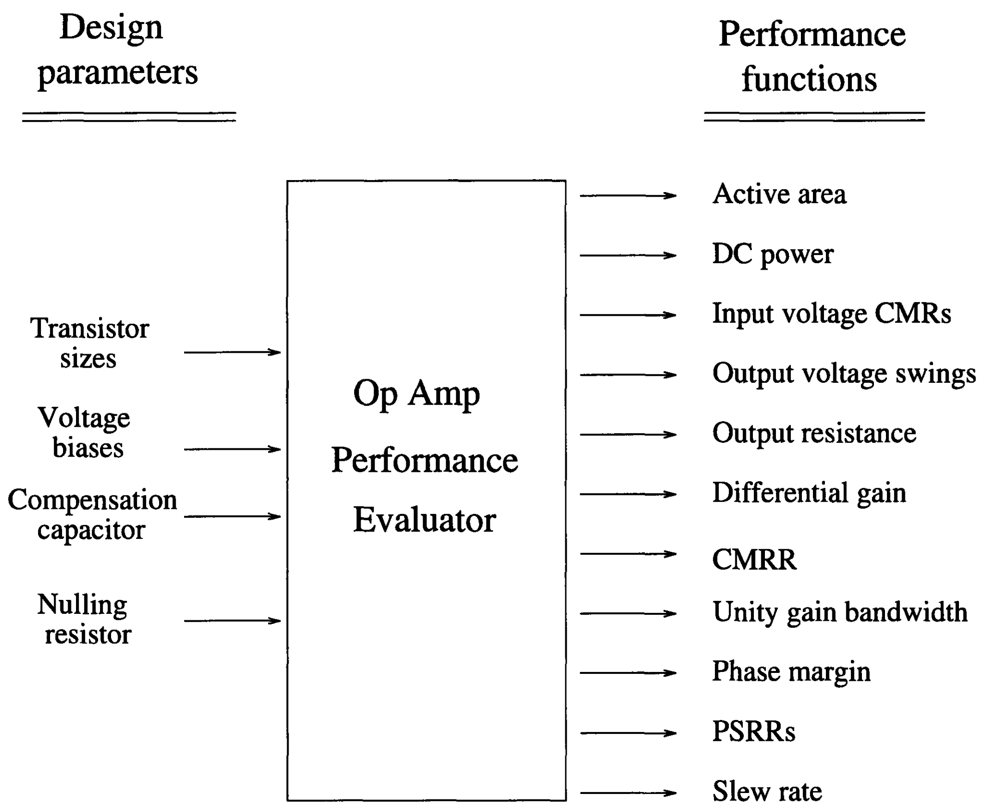


Figure 4-1: Op amp performance evaluation

search direction in the parameter space. However, when an equations based op amp evaluator is used with any optimization method, the achieved designs are in fact only near-optimal due to the approximate nature of evaluating the designs.

The process of creating the design equations (the knowledge base) for each subcircuit module in my modular composition scheme is significantly less time consuming that when all design equations for a fixed circuit topology are mixed with the language that manipulates that knowledge in a single computer program.

The design equations in my evaluator are solved by a mixture of symbolic algebraic transformations, and numerical methods. The balanced mix of symbolic transformations and numerical computations allows full freedom in using highly non-linear equations in my system, unlike exclusively symbolic equation solvers such as ORACLE in the CIROP system [23]. An ability to handle a variety of non-linear equations is particularly important in the approximate analysis based design of MOS analog circuits compared to bipolar analog circuits.

4.1.1 Design Equations

Each object in my hierarchical composition scheme contains relevant design equations which represent the analysis/model knowledge associated with that object.

Typically, the higher the object in the hierarchy, the more general and high level the design equation asserted by that object. For example, among the design equations asserted by the top-level op amp object are the equations used to determine the DC power dissipation of a two stage op amp:

```
(= dc-power (* (+ cs-i os-i) (- vdd vss))
```

where `cs-i` and `os-i` are the DC bias currents flowing through the current source and output stage. In case the op amp consists of only one stage, the **`nil-output-stage`** object asserts the equation `(= os-i 0)`.

The op amp object asserts almost all of the high level equations that compute the performance of the op amp. In some instances, however, the structure of the equation is influenced by the choice (or a combination of choices) of the constituent subcircuit modules, and equation-switching information needs to be propagated upward - from the subcircuit module level to the op amp object. Here is, for example, the equation which computes the slew rate:

```
(= slew-rate
  (if (and compensation? output-stage?)
      (/ cs-i cc)
      (if (and (not compensation?) output-stage?)
          (/ os-i c-ii)
          (/ cs-i c-i))))
```

where `compensation?` and `output-stage?` are boolean signal variables that are set to true unless a `nil` module is used as a compensation or an output stage respectively.

The variables `cc`, `c-i`, and `c-ii` are the compensation capacitance, and the loading capacitances of the first and the second stage respectively. For example, for a nil output stage, `c-i` is equal to the output load capacitance `cl` and `c-ii` is non-existent; for a non-nil output stage, `c-i` is equal to the gate-to-source capacitance of the common-source device and `c-ii` is equal to to the output load capacitance `cl`.

An alternative to the equation-switching strategy is to leave the assertion of the equations to the subcircuit modules. This is possible as long as it is one single subcircuit choice, and not a combination of choices that forces a change in the structure of the equation. For example, the DC PSRR (power supply rejection ratio) equations differ with respect to the choice of the utilized output stage. This approach does not require use of boolean signals, but is less elegant since it causes the assertion of a high level equation to be done from a less visible lower level of hierarchy.

At the middle level of hierarchy, the subcircuit objects assert equations that compute various properties of the modules, such as voltage drops, currents, transconductances, and output resistances. For example, the **`simple-output-stage`** object asserts an equation that specifies the DC bias stage current `os-i`:

```
(= os-i (idn-sat sos-sz2 (- vbias vss) 0 kn))
```

where **`idn-sat`** computes the DC saturation MOS transistor current given the size `sos-sz2` of the MOS device, the gate to source voltage `(- vbias vss)`, a zero source to bulk voltage `0` and the transconductance parameter `kn`.

Most of the equations associated with the frequency domain behavior of the op amps are asserted by the compensation modules. In particular, these equations include a complex domain transfer function equation, along with equations for the poles and zeros, and the phase margin. This is understandable since it is the compensation module choice that drastically changes the algebraic structure of the transfer function.

The circuit components, located at the bottom of the hierarchy, assert basic large and small signal device model equations such as, for instance, the DC saturation current equation and the threshold voltage (body effect) equation asserted by the NMOS transistor object:

```
(= (idn-sat sz vgs vsb kn)
   (* 1/2 kn sz (expt (- vgs (vtn vsb)) 2)))

(= (vtn vsb)
   (+ vton (* gamman (- (sqrt (+ phin vsb))
                        (sqrt phin)))))
```

The large and small signal model equations used in my program are summarized in the Appendix.

The equation hierarchy, from high to low level, enables the equation solver to ignore the details irrelevant to a variable's solution.

4.1.2 Equation Solving Language

The equation solving language developed for my system is flexible and is easily used to implement both performance evaluator based and human engineer mimicing design strategies in my system.

The built in language is capable of automated recursive symbolic and numerical substitutions (with an optional overriding mechanism) and structural transformations of an equation. The transformation process stops when all the variables in the transformed equation (except the variable we are solving for) have numerical values. After a residual function and its derivative are constructed from the transformed equation, the solver reverts to numerical methods to determine the value of the unknown variable.

The equation solving language solves for a single variable from one equation. However, the symbolic substitution mechanism provides for solver's ability to solve a class of n simultaneous equations in n unknowns: systems in which $n-1$ unknown variables are the LHS of $n-1$ equations. A full capability to handle general cases of simultaneous equations was not necessitated by the op amp design equations.

Equation Solving Example

The following example illustrates the abilities of the equation solving language.

The equation solver initially allocates hash table space for all equation variables. Each variable is associated with four slots: a slot for the the numerical value of the variable, a slot for a residual function for that variable (used in the numerical method scheme described later in this Chapter), a slot for the derivative of the residual function, and a slot for the design specification for the variable (if any).

In addition, the initialization process uploads numerical values for the design constants and design defaults (summarized in the Appendix) from a CMOS technology file.

Imagine that we are now at a particular point in the design process when the op amp topology is known. The set of the design equations for that op amp is obtained in a straightforward fashion - by appending the design equations from each of the constituent subcircuit modules and the circuit components. For example, let's assume that an op amp composed of a simple current mirror, a simple differential stage, a simple current source, a simple output stage, and a miller compensation module has been selected (determined either by the topology design parameter during a genetic optimization algorithm run, or by the explicit choice of the user):

```
(define opamp1
  (opamp 'opamp1 simple-current-mirror
         simple-diff-stage simple-current-source
         simple-output-stage miller-compensation)
```

This op amp topology is described by a total of 79 design equations that can be obtained by issuing a design request to the newly composed op amp object by typing `(opamp1 'design)`. Suppose we now want to compute the DC power dissipation of this op amp for a particular choice of the design parameters (transistor sizes, bias voltages, and compensation capacitor and resistor values). After writing the values of the design parameters in the variable hash table slots, we issue the request `solve: 'dc-power)`.

The solver searches and finds the high level DC power equation, and then proceeds to transform the DC power equation with a sequence of recursive symbolic substitutions until all variables in the transformed equation (except dc-power) are associated with a numerical value (i.e. are either design constants, design defaults, or have previously been solved for). Under the assumption that only the design constants, defaults, and parameters are the only variables associated with numerical values, the structural transformation process successfully completes with the equation:

```
(=
 dc-power
 (*
  (+
   (* 1/2 kn scs-sz
      (expt (- (- vbias vss)
               (+ vton (* gamman (- (sqrt (+ phin 0))
                                    (sqrt phin))))) 2))
   (*
    1/2
    kp
    sos-sz1
    (expt
     (-
      (-
       (-
        (sqrt
         (/
          (* 2
             (* 1/2
                (* 1/2 kn scs-sz
                   (expt (- (- vbias vss)
                            (+ vton (* gamman (- (sqrt (+ phin 0))
                                                 (sqrt phin))))) 2))))
          kp scm-sz))
        (- vtop (* gammap (- (sqrt (- phip 0)) (sqrt phip))))))
       (- vtop (* gammap (- (sqrt (- phip 0)) (sqrt phip)))))
      2)))
  (- vdd vss)))
```

in which all variables are known design parameters, or process constants.

In the next step, the solver constructs a residual function of the form

```
(lambda (<var>)
  (- <RHS> <LHS>))
```

where <LHS> and <RHS> are the left and right hand sides of the transformed equation where <var> is the variable we are solving for.

In the particular case of solving for the `dc-power`, the solver obtains its numerical value by evaluating the residual function at zero.

In case the unknown variable is not the LHS of the transformed equation, the solver proceeds to construct a function that computes the derivative of the residual function by using a numerical central difference approximation formula. The value of the unknown variable is then computed in two steps: in step (1) the zero-crossing of the residual function is bracketed by applying the *bisection bracketing method*; and in step (2) the *Newton's method* is used to determine the location of the zero-crossing. For example, to solve for the current source MOS transistor size `scs-sz`, we type:

```
(solve: 'scs-sz in: 'dc-power)
```

It is also possible to override the substitution mechanism causing it to perform symbolic substitutions for specific variables regardless of whether those variables have numerical values. This feature is valuable in the implementation of the design strategy that imitates human circuit designers, also featured in my system and described in Chapter 6. For example, to solve for the current source size `scs-sz`, symbolically substituting for the output stage DC current `os-i`, we type:

```
(solve: 'scs-sz in: 'dc-power sub-vars: 'os-i)
```

The experience has shown robust performance of the combined bisection bracketing and Newton's method, and thus eliminated the need for constructing a more accurate symbolic derivative of the residual function.

4.2 Simulator Based Performance Evaluator

The circuit simulator based performance evaluator in my system is used to verify the optimal op amp designs. The primary advantage of this evaluator is its greater accuracy in predicting the performance of the design. The disadvantages include a much higher computing time when compared with an approximate equation based evaluator, and a possibly high rate of convergence failures during a typical optimization run.

My simulator based op amp evaluator consists of a two-way interface to an industry-standard general-purpose circuit simulator (SPICE), and a raw simulator data processing software.

As discussed in Chapter 3, the composition/VLSI knowledge in the objects provides the capability of producing a complete source SPICE deck, complete with all required circuit measurement setups and analysis commands. After the circuit simulation is completed, the binary raw data reader program loads the waveform data into my system for processing. The processing software features a number of waveform analysis algorithms which are used in computing the op amp performance functions.

4.2.1 Circuit Simulation Experiments and Analysis of Raw Data

Each circuit simulation run consists of three analyses: DC (bias point and DC transfer curve), AC (frequency domain), and TRAN (transient or time domain). The MOS transistor model employed by the simulator was UCB Level 1.

It is important to note that my system does not perform the DC input voltage sweep by repeated DC bias point computations. Instead, I try to resemble the reality by employing a slow transient (time domain) sweep of the input voltage for a period of 10 seconds. The power rails and bias voltages are ramped up to their constant values during the preceding 10 seconds. An identical ramping approach was adopted in the initial phase of the AC analyses to get the bias points and small signal circuit parameters.

This methodology, originally suggested by Gerry Sussman, decreases the possibility for a DC point non-convergence which is one of the primary obstacles to a successful and accurate simulation of a circuit composed of many non-linear elements (such as transistors). While the rate of convergence failures for my op amps noticeably decreased, it did not vanish. Also, a major drawback of the input ramping approach is that it drastically increases the simulation time.

The convergence of the simulator can further be improved by changing the run-time parameters of the simulation run, such as changing the iteration control method and increasing the relative error tolerance. Also, a gmin-stepping approach is sometimes helpful.

I have also developed a node voltages caching scheme to help the simulator's convergence. In this scheme, the node voltages of the op amps in the upcoming simulation experiments are set to the converged node voltages of previously simulated op amps which are a minimal Euclidean distance away in the multidimensional space of design parameters.

Nevertheless, the simulator convergence problems remain, and are particularly exacerbated if we try to use the simulator based performance evaluator with a non-local optimization strategy, such as a genetic algorithm. Such optimization schemes often generate unorthodox points in the space of design parameters which more easily cause a simulator convergence failure in my observation. This is why I primarily use the simulator based op amp evaluator for verification of the optimal designs, obtained by a approximate equations performance evaluator based optimizations.

DC Analysis

The circuit setup for the DC analysis is shown in Figures 4-2–4-4.

The unity-gain configuration of the circuit of Figure 4-2 is used to obtain a measure of the input common-mode range. The input signal V_{IN} is a -5 to 5V 1V/sec ramp. The voltage range over which the output V_{OUT} follows the input with a 5% slope tolerance is taken as a good measure of the input CMR.

The algorithm for computing the input voltage CMR first computes the derivative of the output voltage waveform. Next, it proceeds to label with "1" those points in the output voltage waveform at which the waveform slope is within the slope tolerance, and with "0" the rest. In effect, the waveform is digitized into a sequence of 1's and 0's. Finally, the program locates the longest uninterrupted segment of labeled points in the digitized waveform and returns the edge points of this segment as a fair estimate of the input CMR.

Similarly, the non-inverting circuit configuration of Figure 4-3 of gain ten is used to determine the output swing. The output swing algorithm is identical to the input CMR algorithm.

The zero-input unity-gain circuit configuration of Figure 4-4 is used to determine the input offset (which appears at the output of the op amp), as well as the power dissipation computed as $Power = V_{DD}I_{DD} + V_{SS}I_{SS}$.

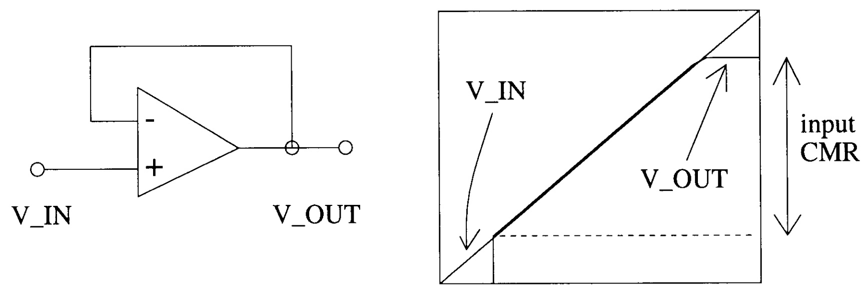


Figure 4-2: Measurement of the input CMR

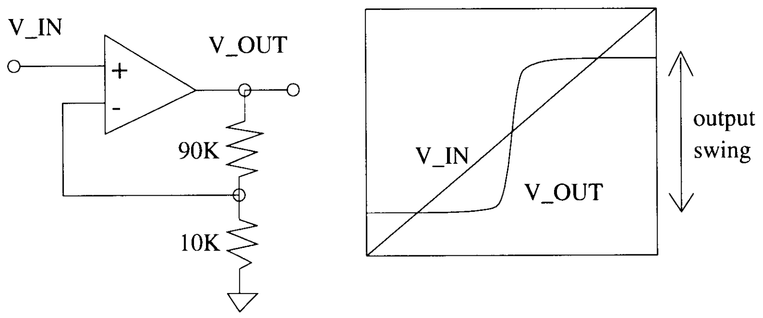


Figure 4-3: Measurement of the output swing

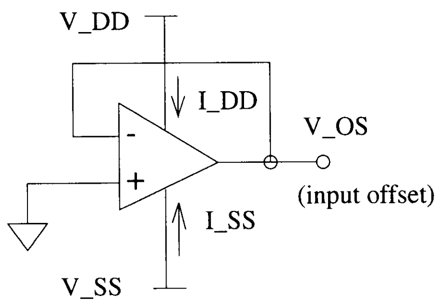


Figure 4-4: Measurement of the input offset and power dissipation

AC Analysis

The circuits used in the AC frequency domain measurements are given in Figures 4-5–4-7. It is important to note that these measurements use a linearized version of the op amps.

Figure 4-5 shows an open-loop op amp setup for measurement of the differential gain, the unity-gain bandwidth frequency, and the phase margin. The input to the linearized op amp is a 1V sinusoid which sweeps through a range of frequencies. In reality, this input voltage amplitude would cause a saturation of the op amp. However, no saturation is present in my AC measurements since the simulator uses a linearized version of the op amp. The 1V choice is simply a convenient way to make the output voltage and the differential gain numerically equal. Note the DC V_{OS} input offset voltage source which is important in order to properly bias the op amp.

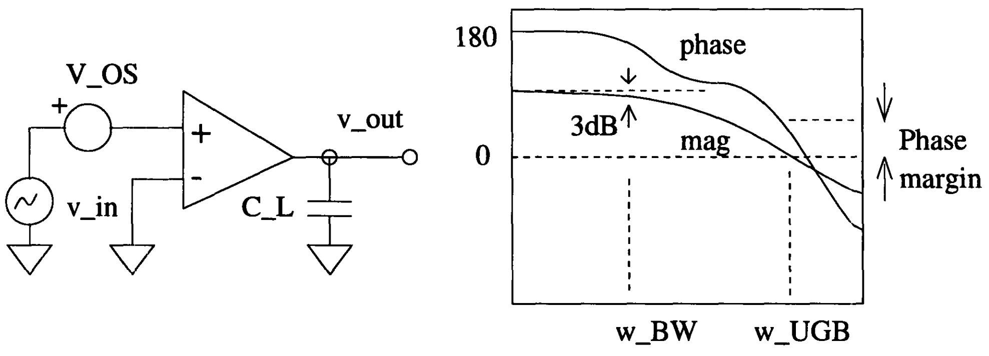


Figure 4-5: Measurement of the differential gain, UGB frequency, and phase margin

The common-mode gain, and thus the CMRR are determined from the circuit setup shown in Figure 4-6.

The PSRR values were determined from gain measurements of the circuit setups shown in Figure 4-7. For *PSRR+* a 1V sinusoid is added to the positive rail supply. *PSRR−* is determined when a 1V sinusoid was added to the negative supply.

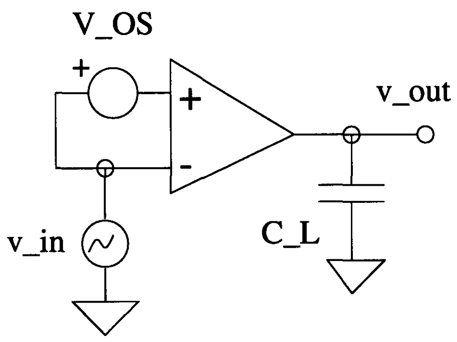


Figure 4-6: Measurement of the common-mode gain and CMRR

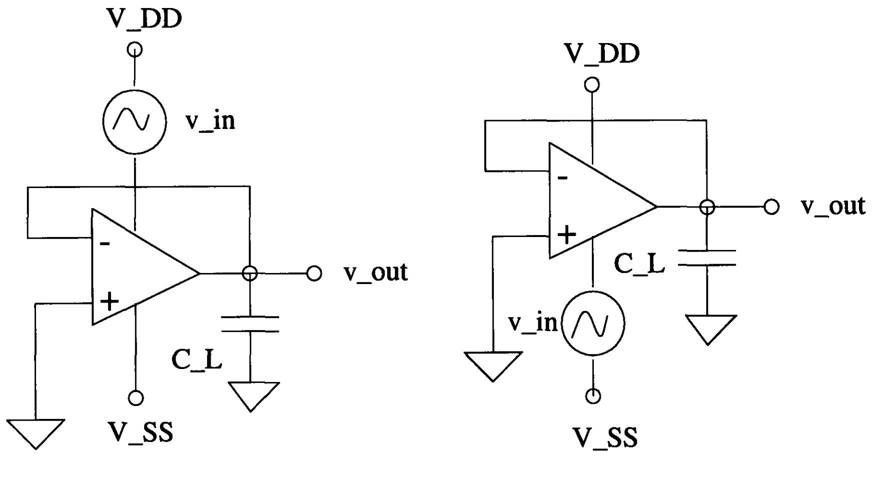


Figure 4-7: Measurement of the PSRR+ (left) and PSRR- (right)

TRAN Analysis

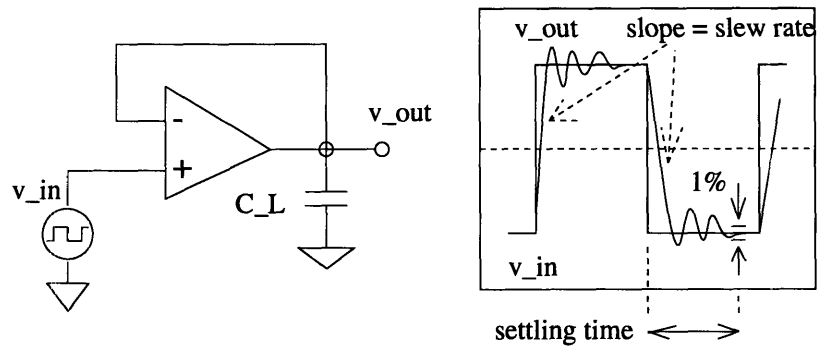


Figure 4-8: Measurement of the slew rate and settling time

The slew rate and the settling time are measured from the time domain setup shown in Figure 4-8. The input signal is a 250KHz 1V square wave. The slew rate is simply computed as the slope of the output at the zero volt crossings.

The settling time algorithm is based on digitizing the output voltage waveform. The algorithm marks all points in the output voltage waveform that are within 1% of waveform's final value in each semi-cycle. The settling time is then calculated by subtracting the time length of the longest segment of sequential marked points from the half-period.

Each pair of values from the rising and falling input edge is averaged to obtain an average value for both performances.

Chapter 5

Main Design Strategy

The main design strategy featured in my system is built around a genetic optimization algorithm which evolves a population of op amp chromosomes using simplified principles of genetics and natural selection.

5.1 Why a Genetic Algorithm?

As I said in Chapter 1 the tools for automated analog circuit synthesis differ with respect to the incorporated design strategy. A number of design systems mimic the human designer in designing an op amp. An important drawback of this approach is that it does not yield optimal designs. In addition, as discussed in Chapter 3, even when the systems assemble the op amps from subcircuit modules, their heuristic selection and failure/redesign rules do not guarantee an exhaustive and effective search through the topological op amp space.

In order to achieve optimal design (or more precisely near-optimal if the design system uses an approximate equation based performance evaluator), many automated op amp design tools feature optimization algorithms.

Which of the op amp performance functions will assume the role of an objective function to be minimized or maximized will vary depending on the intended application of the op amp. The remaining performance requirements can be set up as inequality and equality constraints. One can also combine several performance functions in a weighted norm to perform multiple criterion optimization. The constraints can also be converted into penalty functions and incorporated in the objective function to achieve an unconstrained formulation of the problem.

The algorithmic optimizer choices differ from system to system: feasible directions methods, steepest descent, constrained non-linear (NLP) methods, multiple objective schemes and many others have been used.

During the course of my research in optimization of MOS circuits we tried a number of optimization algorithms: single objective minimization schemes such as the Nelder-Mead (also known as polytope or simplex) method, Powell's non-gradient method, quasi-newton methods (with BFGS and other update schemes) as well as the weighted minmax (L-∞) multiple objective scheme [18].

In order to guarantee an exhaustive but efficient search of the topological space, we need to extend the design optimization problem by incorporating a design parameter which describes the op amp topology. The topology design parameter would take on as many distinct values as there are topological alternatives.

The introduction of the topology design parameter induces dense discontinuities in the search space. The integer-valued topology design parameter has no continuous analogue since there is no sensible interpretation of a non-integer value. As this parameter changes from one value to another, the topological perturbation causes extreme changes in the objective function. On the other hand, all aforementioned optimization methods depend on the restrictive requirements of continuity of the objective function.

While some of the zero-order (non-gradient) methods such as Nelder-Mead allow non-differentiability and noise in the objective functions, they will fail in search spaces fraught with dense discontinuities and multimodalities. So it comes as no surprise that in all op amp design automation efforts up to date the topological selection is a heuristic rule-based process rather than a particular integer value of a topology design parameter. In conclusion, all aforementioned optimization methods share a common bug: they lack robustness.

In our search for a robust optimization algorithm capable of accommodating a topology design parameter, we may consider a combinatorial optimizer, such as "branch and bound" methods. In these methods the objective function is optimized for all possible combination of values of the integer design parameter(s). But even for a moderate number of parameter combinations, this scheme becomes too expensive.

The simulated annealing algorithm is intended for pure discrete problems. It has reportedly met limited success when applied to continuous or mixed discrete and continuous problems due to its highly randomized nature and other factors [21].

Genetic algorithms (GAs) differ in fundamental ways from other optimization algorithms. They achieve greater robustness due to four differences: coding of the design parameter set, parallel search from a population of design points, blindness to auxiliary information, and probabilistic iteration transition rules [10]. GAs are well suited for large multidimensional search spaces.

The parameter set coding in GAs allows a straightforward integration of a topology design parameter and allows GAs to deal successfully with the dense discontinuities and of the search space.

Moreover, the population based search enables genetic algorithms to exhibit attractive global search properties and to search multimodal spaces. In comparison, most other optimization algorithms are intrinsically local in scope and limited to unimodal spaces. An advantageous consequence of the population based GA search is that there is no need for a good initial or feasible point to start the search.

One of the drawbacks of the genetic algorithms is that they are in essence an unconstrained method. Another drawback of the GAs is that they usually require substantially more objective function evaluations that most other optimization schemes.

I originally used the GAs in a optimal design study of a fixed topology CMOS op amp [19]. Two optimization algorithms, the general reduced gradient (GRG) method (an NLP constrained optimization scheme), and a GA were alternatively employed in four different optimization experiments. The results demonstrated that the GA had outperformed the GRG method and had confirmed GA's robustness.

5.2 A Simple Genetic Algorithm

The mechanics of a simple genetic algorithm are simple, and basically involve copying bit strings and swapping partial bit strings.

The set of design parameters for the optimization problem is coded as a finite length string (a chromosome) over a finite alphabet. The lower the cardinality of the alphabet, the better the resolution of the representation of the search space. This is why GAs use a binary based coding in most cases.

After a random generation of the initial population of chromosomes, each chromosome (individual) is associated a fitness value, equivalent to a unconstrained function value that we aim to maximize. The evolution mechanism, featuring crossover (gene exchange) and random mutation (Figure 5-1), favors individuals with a higher fitness, and thus drives the population to a higher average fitness with each generation (iteration).

GAs can optionally implement an "elite scheme" which means that the best (most fit) chromosome of each generation is automatically carried over to the next generation. Elitism insures against losing the chromosome with the highest fitness in each generation, and thus results in a steady improvement or, at worst, stagnation of the fitness function value over the GA generations.

De Jong's milestone study from 1975, summarized in [10], showed that elitism improves the local search at the expense of the global perspective.

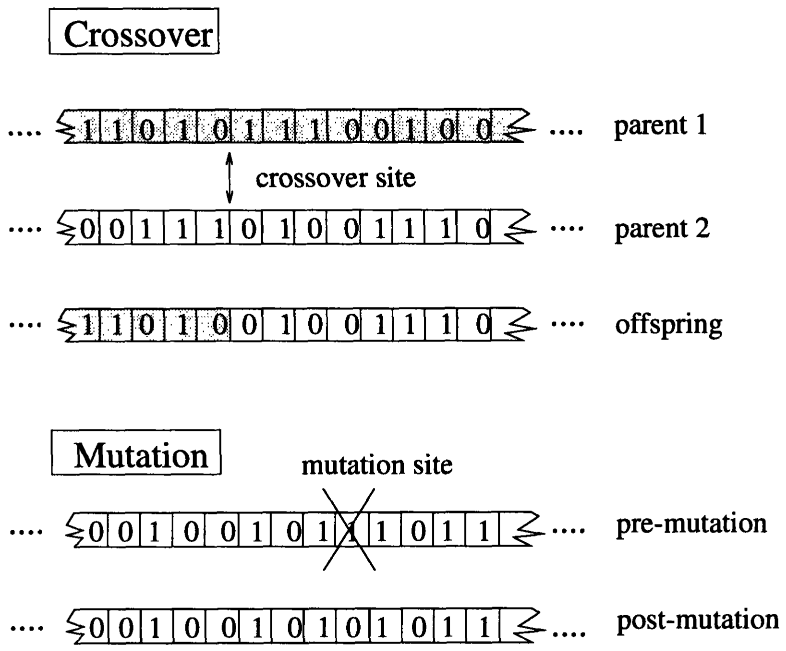


Figure 5-1: Crossover and mutation

5.3 Coding of Design Parameters

The GA design strategy in my program features a concatenated, multiparameter, mapped fixed-point coding.

The op amp chromosome structure used in my GA based design strategy is shown in Figure 5-2. My op amp chromosome contains 14 concatenated phenes (gene sequences), grouped in six functional segments: one segment for each of the five constituent subcircuit modules and one for the topology. The segments vary in length as different modules are described by a different number of design parameters.

All phenes in the op amp chromosome are 12 genes (bits) long, except the topology phene which contains only 7 genes, yielding a 163 gene op amp chromosome. Each phene encodes the value of one or more design parameters as shown in Table 5.1. The design parameter phene sharing is facilitated by realizing that two or more subcircuit modules of the same type but of different styles cannot appear in one op amp. The

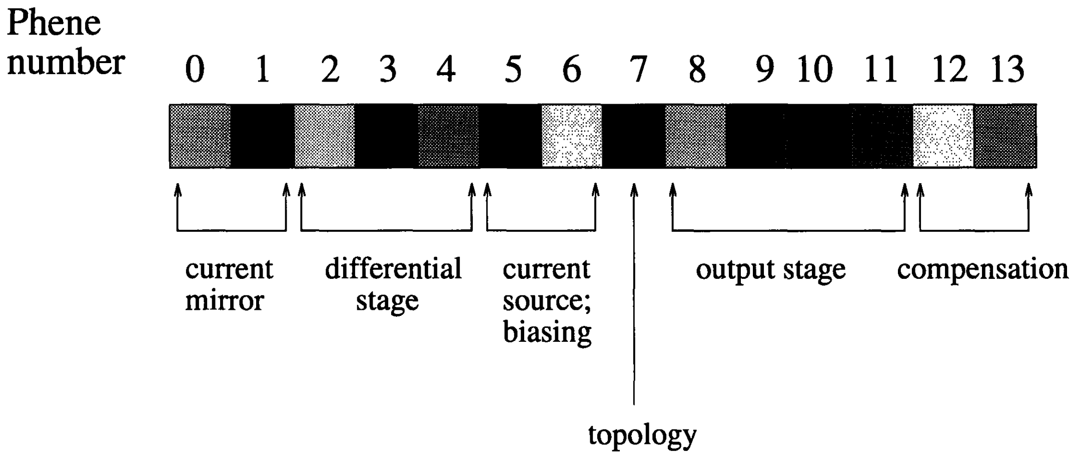


Figure 5-2: Op amp chromosome structure

phene sharing scheme is a welcome way of reducing the total chromosome length.

The range of unsigned integer values for each phene $[0, 2^{12} - 1]$ is linearly mapped onto the range of allowable values for each design parameter. The 12 genes (bits) provide more than sufficient precision for each design parameter, as it can be seen from Table 5.2.

The coding of the topology in the topology phene differs, and is shown in Figure 5-3. The particular bit (allele) values shown in the Figure encode an op amp topology featuring a wilson current mirror, a simple diff stage, a simple current source, a cascode output stage, and a nulling resistor compensation. Note that the absence of current source module style in the topology phene structure is facilitated by the single current source style choice.

A change in the gene values (one of the 7 alleles) in the topology phene modifies the topology of the op amp and can potentially introduce sharp fitness function discontinuities and nonlinearities. Moreover, the genes in the topology phene mask and modify the expression of other genes in the chromosome. This gene interaction is called epistasis and has direct repercussions on the efficiency of the GA. If the epistasis is too high, the GA efficiency deteriorates down to the level of a random

Design Parameters Coding			
Phene	Genes	Value	Design Parameter(s)
0	12	0.5 $\rightsquigarrow$ 100	`scm-sz arl-sz ccm-sz wcm-sz`
1	12	0.5 $\rightsquigarrow$ 100	`ccm-csz wcm-csz`
2	12	0.5 $\rightsquigarrow$ 100	`sds-sz cds-sz`
3	12	0.5 $\rightsquigarrow$ 100	`cds-csz`
4	12	-5 $\rightsquigarrow$ 5	`cds-vbias`
5	12	0.5 $\rightsquigarrow$ 100	`scs-sz`
6	12	-5 $\rightsquigarrow$ 5	`scs-vbias sos-vbias cos-vbias`
7	7	0 $\rightsquigarrow$ 127	*topology*
8	12	0.5 $\rightsquigarrow$ 100	`sos-sz1 cos-sz1`
9	12	0.5 $\rightsquigarrow$ 100	`sos-sz2 cos-sz2`
10	12	-5 $\rightsquigarrow$ 5	`cos-vbiasn`
11	12	-5 $\rightsquigarrow$ 5	`cos-vbiasp`
12	12	0.1e-12 $\rightsquigarrow$ 20e-12	`cc`
13	12	0 $\rightsquigarrow$ 100e3	`rz`

Table 5.1: Coding of op amp design parameters

search. I believe my phene sharing scheme reduces the epistasis, since it reduces the number of genes that are masked or stopped by the topology genes.

5.4 GA Design Strategy Setup and Runtime Parameters Choice

The genetic algorithm implemented for my experiments is an elitist scheme GA with a simple crossover reproduction and a mutation operators.

Design parameter coding precision	
Parameter(s)	Coding precision
Transistor sizes	0.0243
Bias voltages	2.44 mV
Compensation capacitance	4.86 fF
Nulling resistance	24.4 Ω

Table 5.2: Precision of the design parameter coding

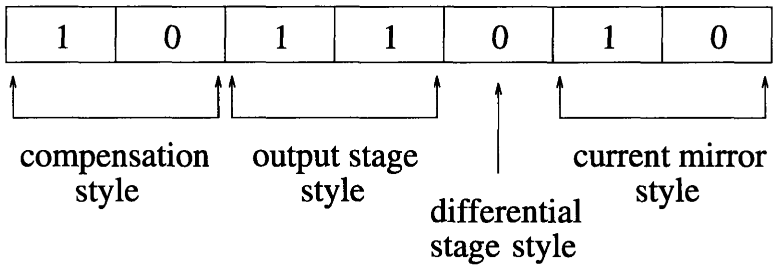


Figure 5-3: Topology phene structure

The implementation of the crossover and mutation operators in our program is straightforward. The rates of crossover and mutation are controlled by the two run-time probability parameters p_c and p_m. The probability that a given op amp chromosome will survive and be mated with another chromosome is proportional to the fitness value of that chromosome. The crossover and mutation sites are chosen at random.

A summary of the default GA design strategy setup and runtime parameters in my system is given in Table 5.3.

There is a sufficient body of work in GA theory that shows that tweaking the crossover and mutation probabilities should not produce large perturbations in the robustness of the GA under an assumption that the GA coding representation is well

Default GA setup and run-time parameters	
Parameter	Choice or Value
Chromosome length (genes)	163
Phenes/Chromosome	14
Genes/Phene	12 or 7
Population Size	30
Crossover Probability	90%
Mutation Probability	1%
Generations	300

Table 5.3: GA default design strategy setup and run-time parameters

chosen. The probabilities, however, can be used to control the convergence speed of the algorithm. In general, a higher crossover rate and a lower mutation rate speed up the GA convergence, but can deteriorate global search properties.

The crossover and mutation probabilities were adjusted through repeated test optimization runs. My objective was to achieve a rapid, but controlled convergence to the optimal solution, as well as to preserve the global properties of the GA search. As a result, the crossover probability is moderately high at 90%, while the mutation rate is at a medium to low 1% (compared to typical default GA crossover and mutation rates of 80% and 3% respectively for a general family of optimization problems). I observed no large changes in the robustness of the of the GA search which confirmed that the coding scheme was well chosen.

The elitism built in my GA procedure helps in localizing the search and in enhancing the convergence. I try to compensate for the possible deterioration of global search properties by simply repeating each design experiment many times with fresh random initial populations.

Since the number of fitness function evaluations in each generation is equal to the size of the population, we are interested in having as small a population as possible. On the other hand, a necessary condition for a globality in the GA search is that every point in the search space needs to be reachable (i.e. all alleles present) by crossover only. The work of Reeves [22] shows that for a linearly mapped binary coding, and 163 gene chromosomes, a minimal population of 15 is sufficient to achieve a 99% confidence of all alleles present. My population size of 30 has a negligible allele loss rate of 3e-5%.

Reeves also investigated the possible benefits of using a structured rather than a standard random initial population. He discovered that initial populations generated by using linear error-detecting codes are 10-20% more effective in covering the search space than random populations. While I use a random initial population in my system, the small population size enable the aforementioned repeated design experiment runs without overloading the computer resources. The repeated optimizations enhance the coverage of the search space.

Finally, my chosen population size was experimentally justified when the op amp design experiments showed no observable difference in the quality of the results and the convergence rate for my population size of 30 and for much larger population sizes (up to 1000).

The number of generations, set at 300, allows plenty of evolution time for the op amp populations to evolve and converge to the optimal solutions.

5.5 Fitness Function Structure

One way to formulate a particular op amp design problem is as a constrained optimization. One of the performance functions assumes the role of the objective function to be minimized or maximized, while the others are set up as inequality and equality constraints. An example of such a formulation was given in Chapter 2.

The genetic algorithm based search strategy is, however, an unconstrained method.

The most widely used method to treat constraints is to incorporate them in the objective (fitness) function. Typically, the fitness function becomes a weighted sum of the original objective function minus a penalty term for every constraint violation.

The problem with this scheme is that the choice of the structure of the penalty functions and the relative weights are different for each design problem. Typically, in order for the GA to monitor each constraint in a balanced manner, the weights, and also the penalty functions structure have to be interactively adjusted through repeated optimization runs.

The fitness function structure in my system follows the described constraint scheme. Let p be the vector of design parameters, $f_o(p)$ the performance function to be minimized or maximized and $g(p)$ an individual penalty term for each of the remaining performance functions. The overall structure of the fitness function is:

$$fit(p) = C \pm F(f_o(p)) + G(\sum g(p))$$

where C is a large positive constant, and the plus sign in front of the objective function term corresponds to maximization and the minus sign to minimization of the objective function.

The function F is typically the scaled identity function $F(x) = Sx$. However, in cases where the unrealistic growth of the objective function (which can happen due to the approximations in the op amp performance evaluations) cannot be controlled by the scaling factor S, I limit the awarding of the fitness function by using $F(x) = \min(f_{o(GOAL)},\ x)$ where $f_{o(GOAL)}$ is a realistic goal value for the optimized performance function.

The penalty functions $g(p)$ can take one of the following four forms:

1. $g(p) = S\ u(f(p) - f_{(SPEC)})\ |f(p) - f_{(SPEC)}|$
2. $g(p) = S\ u(f(p) - f_{(SPEC)})\ (\exp|f_{(SPEC)}/f(p)| - \exp(1))$
3. $g(p) = S\ u(|f(p) - f_{(SPEC)}| - \epsilon)\ |f(p) - f_{(SPEC)}|$

4. $g(p) = S\ u(|f(p) - f_{(SPEC)}| - \epsilon)\ (1 + \log(1 + |f(p) - f_{(SPEC)}|))$

where S is a scaling coefficient, $f(p)$ represents the performance function, $f_{(SPEC)}$ its constraint limit, and $u()$ is the unit step function. The first form applies to "less-than" constraints. The penalty terms for most "greater-than" constraints are essentially of the same form, except for the fact that the argument to the unit step function changes sign. It can be seen that if the constraint is binding or inactive, the penalty terms will be zero.

Otherwise, in the first, default form, the penalty is linearly proportional to the constraint violation (i.e. the distance of the performance function value from its constraint limit). The second, exponential dependence form is well suited for "greater-than" constraints, where the performance function in the violated constraint approaches zero in an exponential manner. The third, linearly varying penalty form is used for well-behaved equality constraints. The fourth, logarithmic dependence form, can be used in situations where the equality constraint violation exhibits exponential growth as the performance function in the constraint tends towards plus or minus infinity.

The function G is the scaled power law function $G(x) = Sx^n$. The default value of the exponent n is 1, which degenerates G into a scaled identity function. In some experiments the GA happily obtains a fitness function increase by balancing increases in the objective function term with a larger penalty term (i.e. obtaining a more infeasible design point). In those cases the exponent n can be set to a value between 1 and 2, or even larger, in order to focus the GA search to the feasible regions of the search space.

The scaling factors (weights) S are interactively adjusted through repeated optimization runs, and vary on a case to case basis. Their values need to be (1) large enough to eliminate or to allow only small constraint violations, and (2) small enough at the same time to allow dominance of the optimized performance function in the fitness function.

Chapter 6

Alternative Design Strategies

My system also supports a design strategy that mimics human designers in synthesizing an op amp with a pre-selected modularly composed topology.

6.1 Design Strategy Inspired by Observation of Human Designers (First-Cut)

The formalization of the design methodology by modularizing the op amp design and the decomposition of the design scheme into a knowledge base (equations) and knowledge manipulation (equation-solving) allowed us to incorporate an alternative design strategy in my system. This strategy imitates human analog circuit designers. The op amp topology in this strategy is fixed. An addition of heuristic topology selection and subcircuit failure and redesign rules is all that my system lacks in order to behave analogously to other automated design tools built around human mimicking design strategies such as CIROP [23], OPASYN [15], and OASYS [13].

The objective of the first-cut design strategy is to compute values for the design parameters for the op amp such that all performance function specifications are satisfied. The design procedure starts with an initialization the design equations solver. Next, the system writes the performance function specification values for the performance functions in their hash table slots. A sequence of equation-solving requests can then be issued to compute the design parameter values.

The most serious drawback of the obtained design that it is not optimal. To remedy this shortcoming, we can invoke an optimization algorithm with our first-cut design as an initial starting point. Since the topology is fixed, and we have a reasonably good feasible starting point, we can use local-based methods to optimize the design.

Figure 6-1 shows an example of a simple program that can be used to obtain such a first-cut design a two-stage Miller compensated op amp. The program is simply a sequence of commands of the equation solving language. The chosen op amp topology, shown in Figure 6.1, features simple styles of the current mirror, the differential stage, the current source, and the output stage.

The first-cut design procedure described with the program in Figure 6-1 explicitly specifies the order in which particular equations need to be solved. This is equivalent to the order in which equations are asserted in some other automated design systems such as the CIROP and OASYS systems.

The specification of the variable we are solving for in each transformed equation is analogous to the "variable priority" feature in the CIROP system.

A first-cut design procedure frequently has to feature the simplest approximate equations since at each point in the procedure one can solve only those equations which contain a single unknown variable (unknown in a numerical sense). For example, the unity gain bandwidth and phase margin requirements in the design procedure from Figure 6-1 are considered at an early stage in the process through their simple equations, rather than through the transfer function, which cannot yet be computed. In contrast, an optimization-based strategy can compute these two performances by first equating the magnitude of the transfer function with unity, numerically solving (Newton's method) for the unity gain bandwidth frequency, and then evaluating the transfer function to get the phase margin.

```
... <initialize> ...

;; miller compensation design
(solve: 'cc in: 'phase-margin)
(copy 'cc 'mcp-cc)

;; computation of first-stage DC bias currents
(solve: 'cs-i in: 'slew-rate)
(solve: 'cm-i)
(solve: 'ds-i)

;; simple differential stage design
(solve: 'sds-sz in: 'ugb)
(write-max 'sds-sz 'ds-sz-min)

;; simple current mirror design
(solve 'scm-sz in: 'input-cmr+)
(write-max 'scm-sz 'cm-sz-min)

;; simple current source design
(solve: 'scs-sz in: 'input-cmr-)
(write-max 'scs-sz 'cs-sz-min)
(solve: 'vbias)
(copy 'vbias 'scs-vbias)

;; simple output stage design
(solve: 'gm-ii in: 'gm-ii/gm-i)
(solve: 'sos-sz1 in: 'gm-ii sub-vars: 'os-i)
(solve: 'os-i)
(copy 'os-i 'os-i*)
(solve: 'sos-sz2 in: 'os-i*)

... <recompute performances more accurately> ...
```

Figure 6-1: A simple two-stage op amp first-cut design procedure

One may think that an addition of a few simple failure and redesign rules to the first-cut strategy may remove this shortcoming. For example, one can recompute a number of performances more accurately at the end of the first-cut procedure, and in case of a failed design performance branch to the beginning of the procedure with stricter requirements to redesign the op amp. This approach doesn't work well in practice since by imposing one stricter performance requirement on one, we may cause another performance requirement failure. In other words, we may get stuck in the infeasible region, and fail to complete the design task unless the failure and redesign rules are not sophisticated enough.

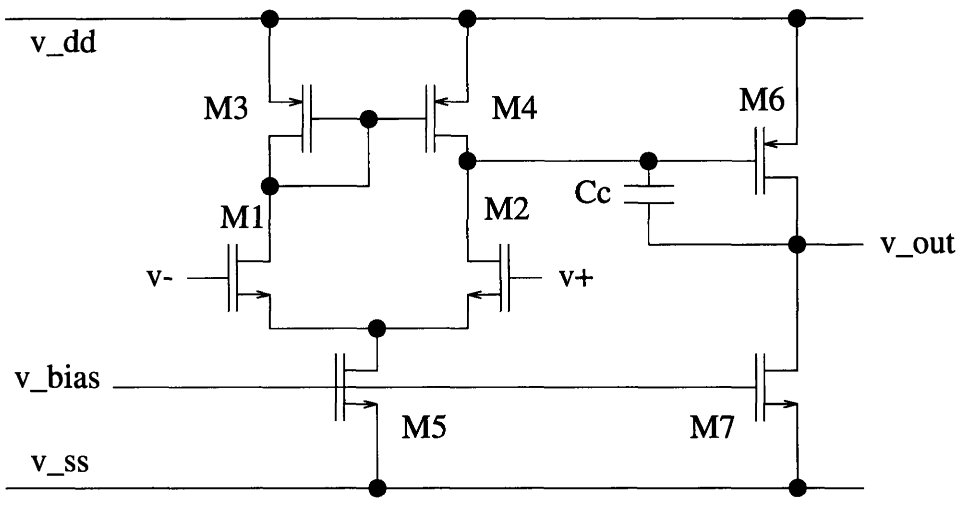


Figure 6-2: Two stage Miller compensated op amp

The necessity for smart failure and redesign rules, together with the fact that no first-cut strategy arrives at an optimal design are the primary reasons why I strongly favor optimization-based strategy in this thesis.

6.1.1 First-Cut Design Example

To demonstrate the shortcomings of the first-cut design strategy I first used the procedure in Figure 6-1 to design the simple two stage op amp shown in Figure 6.1. Second, I used my main genetic algorithm based design strategy with the topology alleles fixed to design the same op amp topology with a minimization priority given to the active area of the amplifier.

The set of design specifications for both design experiments are given in Table 6.1 and Table 6.2. The nature of the first-cut strategy is such that it considers only a subset of the specifications for the GA based strategy as indicated in the table. In order to consider the rest of the specifications, we need to incorporate failure and redesign rules in the first-cut strategy.

Table B.2 in the Appendix summarizes the process constants.

Design Parameters	
Design Parameter(s)	Allowable Range
Transistor sizes M	$0.5 \leadsto 100$
Compensation capacitor C_C (pF)	$0.1 \leadsto 20$
Bias voltages V_{BIAS} (V)	$V_{SS} \leadsto V_{DD}$

Table 6.1: Simple two stage op amp design parameter specifications

The design parameters produced by both design strategies are given in Table 6.3. The achieved performance of the designed op amps are shown in Table 6.4 and Table 6.5. The tables include SPICE verification data.

As it can be seen from Table 6.4, the overall performance of the first-cut designed op amp is quite good, despite the fact that the first-cut strategy takes into account only a few performance specifications, and despite the complete absence of failure and redesign rules in the procedure. In fact, the performance of this op amp satisfies all performance specifications in Table 6.2, with minor performance constraint violations present only in the unity gain bandwidth, the phase margin, PSRR-, and the slew rate.

As expected from an optimization-based strategy, the overall performance of the GA designed two stage op amp summarized in Table 6.5 is better than the performance of the first-cut designed op amp. This is primarily due to the 25% reduction in the minimized active area. All other performances of the GA designed op amp are equal or slightly better than the performances of the first-cut designed op amp except for a minor 9% increase in the power dissipation, and a negligible sub-percent unity gain bandwidth increase in the GA solution. The minor unity gain bandwidth violation in the GA solution is mostly due to the fact that this performance was not computed from the full transfer function, and the minor violation in the slew rate is due to the specific constraint handling scheme used in the GA.

Design Performance Specifications			
Performance	Specification	First-Cut	GA
CMOS Technology	SIMPLE $5\mu m$	$\surd$	$\surd$
Ambient temperature (C)	25	$\surd$	$\surd$
Positive power supply V_{DD} (V)	5	$\surd$	$\surd$
Negative power supply V_{SS} (V)	-5	$\surd$	$\surd$
MOS channel lengths L (μm)	10	$\surd$	$\surd$
Loading capacitance C_L (pF)	20	$\surd$	$\surd$
Stage transconductance ratio g_{mII}/g_{mI}	10	$\surd$	
Active area (μm^2)	MINIMIZE		$\surd$
DC power (mW)	< 20		$\surd$
Input offset (mV)	$= 0$	$\surd$	$\surd$
Input CMR (V)	$-3 \leadsto 3$	$\surd$	$\surd$
Output swing (V)	$-3 \leadsto 3$		$\surd$
Differential gain @ DC (dB)	> 80		$\surd$
CMRR @ DC (dB)	> 80		$\surd$
Unity gain bandwidth (MHz)	> 1	$\surd$	$\surd$
Phase margin (degrees)	> 60	$\surd$	$\surd$
PSRR+ @ DC (dB)	> 80		$\surd$
PSRR– @ DC (dB)	> 80		$\surd$
Slew rate $(\mathrm{V}/\mu s)$	MAXIMIZE	$\surd$	$\surd$

Table 6.2: Simple two stage op amp performance specifications

Computed Design Parameters			
Design Parameter	Allowable Range	First-Cut Value	GA Value
M_1, M_2	0.5 $\rightsquigarrow$ 100	5.15	2.95
M_3, M_4	0.5 $\rightsquigarrow$ 100	1.1	.84
M_5	0.5 $\rightsquigarrow$ 100	2.24	.79
M_6	0.5 $\rightsquigarrow$ 100	34.7	50.53
M_7	0.5 $\rightsquigarrow$ 100	35.4	23.87
C_C (pF)	0.1 $\rightsquigarrow$ 20	4.43	2.59
V_{BIAS} (V)	-5 $\rightsquigarrow$ 5	-3.32	-3.11

Table 6.3: First-cut and GA computed design parameters for a simple two stage op amp

Regardless of the random initialization of the GA op amp population, the genetic search arrived at a reasonable design after only 30 generations. After 150 generations, the design was practically converged, and underwent only minor parametric adjustments in the remaining 150 generations. The injection the first-cut designed op amp in the initial population of the genetic search was not necessary as illustrated by the fast and successful GA convergence.

First-Cut Design Performance Summary			
Performance	Specification	Design	SPICE
Active area (μm^2)	–	17390	–
DC power (mW)	–	1.49	1.56
Input offset (mV)	$= 0$	0	.071
Input CMR (V)	-3 $\rightsquigarrow$ 3	-3.0 $\rightsquigarrow$ 4.0	-4.0 $\rightsquigarrow$ 4.7
Output swing (V)	–	-4.3 $\rightsquigarrow$ 4.0	-4.9 $\rightsquigarrow$ 4.8
Differential gain @ DC (dB)	–	83	84
CMRR @ DC (dB)	–	89	96
Unity gain bandwidth (MHz)	> 1	1.0	.89
Phase margin (degrees)	> 60	56	56
PSRR+ @ DC (dB)	–	86	88
PSRR– @ DC (dB)	–	92	77
Slew rate (V/μs)	> 2	2.0	1.97
Area of C_C (%)	–	59	–
First stage bias current $I_{(CS)}$ (μA)	–	8.86	9.08
Second stage bias current $I_{(OS)}$ (μA)	–	140	147
Output resistance R_{OUT} ($K\Omega$)	–	239	231
Dominant pole $pole_I$ (Hz)	–	71.8	–
Output pole $pole_{II}$ (MHz)	–	1.81	–
Zero (MHz)	–	10.0	–

Table 6.4: First-cut designed simple two stage op amp performance summary

GA Design Performance Summary			
Performance	Specification	Design	SPICE
Active area (μm^2)	MINIMIZE	12950	–
DC power (mW)	< 20	1.64	1.72
Input offset (mV)	$= 0$	-.0064	.062
Input CMR (V)	-3 $\leadsto$ 3	-2.8 $\leadsto$ 4.1	-4.0 $\leadsto$ 4.8
Output swing (V)	-3 $\leadsto$ 3	-4.1 $\leadsto$ 4.1	-4.8 $\leadsto$ 4.9
Differential gain @ DC (dB)	> 80	84	85
CMRR @ DC (dB)	> 80	90	97
Unity gain bandwidth (MHz)	> 1	1.0	.88
Phase margin (degrees)	> 60	60	60
PSRR+ @ DC (dB)	> 80	87	89
PSRR– @ DC (dB)	> 80	93	89
Slew rate (V/μs)	> 2	2.04	1.96
Area of C_C (%)	–	46	–
First stage bias current $I_{(CS)}$ (μA)	–	5.27	5.4
Second stage bias current $I_{(OS)}$ (μA)	–	159	167
Output resistance R_{OUT} ($K\Omega$)	–	210	201
Dominant pole $pole_I$ (Hz)	–	64.7	–
Output pole $pole_{II}$ (MHz)	–	1.89	–
Zero (MHz)	–	22.05	–

Table 6.5: GA designed simple two stage op amp performance summary

6.2 Automated Generation of First-Cut Design Procedures

Is an automated generation of first-cut design programs, such as the one shown in Figure 6-1 possible? In other words, is it possible to write a program which will be able to determine the assertion order of the design equations? My attempt in answering this question resulted in a prototype program described below.

In my prototype program most equation variables are simple transceiver network nodes with memory and a processing unit. These variable nodes are capable of communicating (transmitting and receiving numerical values) to and from any other network node. The memory slot holds the numerical value of the variable, if any. The processing unit in each variable node computes its numerical value from the inputs. There are also start-up nodes which do not contain a processing unit, and are capable of transmitting only.

The subset of nodes associated with a numerical value are termed activated nodes. The transmitting nodes are the node subset of the activated nodes that have been activated (received a numerical value) in a just completed iteration of the main loop of the program. The receiving nodes are all nodes that need to receive data from the current set of active nodes in order to be processed, i.e. to compute their numerical value from a design equation.

The design process starts with an initialization of the start-up variable nodes by writing numerical values into their node memory. These nodes include performance specification nodes such as the nodes for the phase margin and the slew rate, and design constant nodes such as the supply voltage and process parameters.

Upon completion of the initialization, the program enters its main loop, shown in Figure 6-3. In each iteration, we first find all receiving nodes. If there are none, it means that the program reached the end nodes of the node network, so the design is completed. Otherwise, the system proceeds to select and process those nodes from the receiving nodes set that can be fully processed (i.e. all of their input nodes have an associated numerical value). At the loop re-entry, the just processed nodes become the new transmitting nodes, and are added to the set of activated nodes. As the program progresses, it constructs a directed op amp design network (graph) of variable nodes, which is analogous to specifying the assertion order of the design equations.

```
(let main-loop ((transmitting-nodes start-up-nodes)
                (activated-nodes '())
                (receiving-nodes (find-receiving-nodes start-up-nodes)))
  (if (null? receiving-nodes)
      'design-completed
      (let ((new-transmitting-nodes
             (find-new-transmitting-nodes receiving-nodes)))
        (for-each process-node new-transmitting-nodes)
        (loop new-transmitting-nodes
              (node-union activated-nodes new-transmitting-nodes)
              (find-receiving-nodes new-transmitting-nodes)))))
```

Figure 6-3: Main loop of a prototype program for automated first-cut procedure generation

A partial op amp design node network constructed by the program for automated generation of first-cut strategies is depicted in Figure 6.2. The five variable nodes in the top "init" iteration group are the transmitting-only start-up nodes. The receiving nodes in the first iteration are `cc`, `c-ii`, `cs-i`, `l-default` and `cs-rout`, out of which only the `c-ii`, and `l-default` nodes can be processed. Note the possibility of parallel processing for the nodes in the first and the fourth iterations.

The processing of each node is performed by numerically solving a design equation for the particular node variable. For example, to process the node `cc`, the system solves the phase margin equation for `cc`. However, a variable node can usually be processed from many different equations (equivalent to the existence of a several different processing units within a node). This simply means that there will be many first-cut design network graphs, just as there are indeed many first-cut design paths

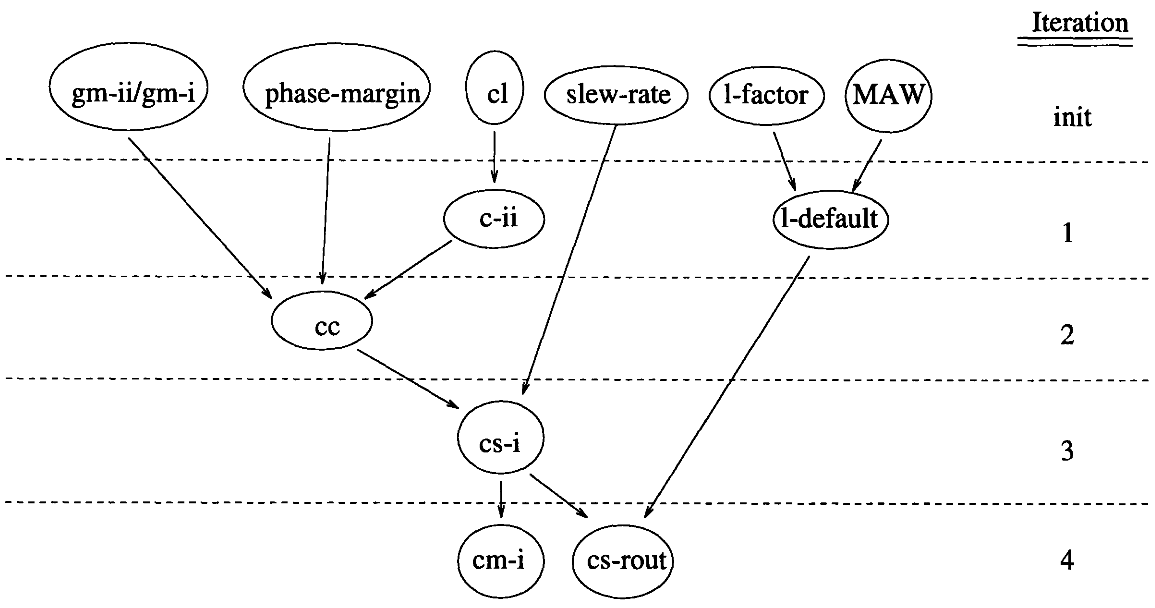


Figure 6-4: Partial design node network for a simple two-stage Miller compensated op amp.

one can take in designing an op amp. It is also possible that some of these node networks may fail to carry the design procedure to its completion.

In such a case, a full set of design specifications for one of our op amps can result in inhibitively many possible node network graphs. In order to determine which of these design graphs is the best one, one could perform a number of design experiments with different design specifications for each of the graphs and compare the results. This process seems expensive. In addition, there may not be one single best design procedure, but the design procedure's quality may be data-dependent (on the particular set of design specification values). This means that one would probably need to look for a design graph for every different set of design specifications, which would greatly increase the computation expense of the entire design procedure. Finally, given that the first-cut procedures, whether automatically generated or not, arrive at results which are not optimal, I conclude that first-cut strategies, while certainly elegant and fashionably AI, are inferior to optimization-based design strategies in terms of result quality.

Chapter 7

Design Experiments

7.1 Minimal Area Op Amp Design

The minimization of the die area of a particular VLSI chip is, perhaps, the most important optimization from the standpoint of the semiconductor industry. The die area is directly related to the manufacturing cost, since smaller chips result in a greater chip yield per wafer, lowering the cost per chip.

In this design experiment I minimize the active op amp area (as defined in the Appendix), which is proportional to the total die area. The Tables 7.1 and 7.2 summarize the complete design specifications for this experiment.

Given that the compensation capacitor and the nulling resistor typically comprise a large percentage of the active op amp area, one would expect that the dominant (best fit) op amp individuals in the initial random population of thirty modularly composed, and topologically and parametrically varied op amps and its future generations are those without a compensation module (i.e. with a nil compensation module). Indeed, within the first few generations, op amps featuring a cascode current mirror, simple styles of the differential stage and a nil output stage and compensation dominated the population. (The op amps with nil compensation and non-nil output stage are genetically inferior mainly due to their disastrous phase margin and stood

Design Parameters	
Design Parameter(s)	Allowable Range
Transistor sizes M	$0.5 \rightsquigarrow 100$
Compensation capacitor C_C (pF)	$0.1 \rightsquigarrow 20$
Nulling resistor R_Z $(k\Omega)$	$0 \rightsquigarrow 100$
Bias voltages V_{BIAS} (V)	$V_{SS} \rightsquigarrow V_{DD}$

Table 7.1: Minimal area op amp design parameter specifications

almost no chance to survive.)

The evolution of this op amp topology quickly revealed its inadequacy in satisfying the gain requirement. Between generations 10 and 20, the evolution mechanisms induced a change in the differential stage genes of some of the op amps, introducing cascode devices. The change predictably boosted the gain, and the fitness of the representatives of the new topology in the population. These op amp individuals thus started to dominate the population. Between generations 20 and 80, the genetic search seemingly focused more on the parametric search for the dominant topology individuals. By generation 80, the design parameters of the best op amps were practically determined, enjoying a slow converging improvement until the end of the genetic process at generation 300.

The genetic search terminated with the best op amp being a one stage self-compensating amplifier featuring a cascoded current mirror and a differential stage, a simple current source, and no (nil) compensation and output stage. The topology of this op amp is given in Figure 7.1, its set of optimal design parameters in Table 7.3, its achieved performance in Table 7.4, and simulation plots in Figures 7-2, 7-3, and 7-4.

Design Performance Specifications	
Performance	Specification
CMOS Technology	SIMPLE $5\mu m$
Ambient temperature (C)	25
Positive power supply V_{DD} (V)	5
Negative power supply V_{SS} (V)	-5
MOS channel lengths L (μm)	10
Loading capacitance C_L (pF)	20
Active area (μm^2)	MINIMIZE
DC power (mW)	< 20
Input offset (mV)	$= 0$
Input CMR (V)	-3 $\leadsto$ 3
Output swing (V)	-3 $\leadsto$ 3
Differential gain @ DC (dB)	> 80
CMRR @ DC (dB)	> 80
Unity gain bandwidth (MHz)	> 1
Phase margin (degrees)	> 60
PSRR+ @ DC (dB)	> 80
PSRR− @ DC (dB)	> 80
Slew rate (V/μs)	> 2

Table 7.2: Minimal area op amp performance specifications

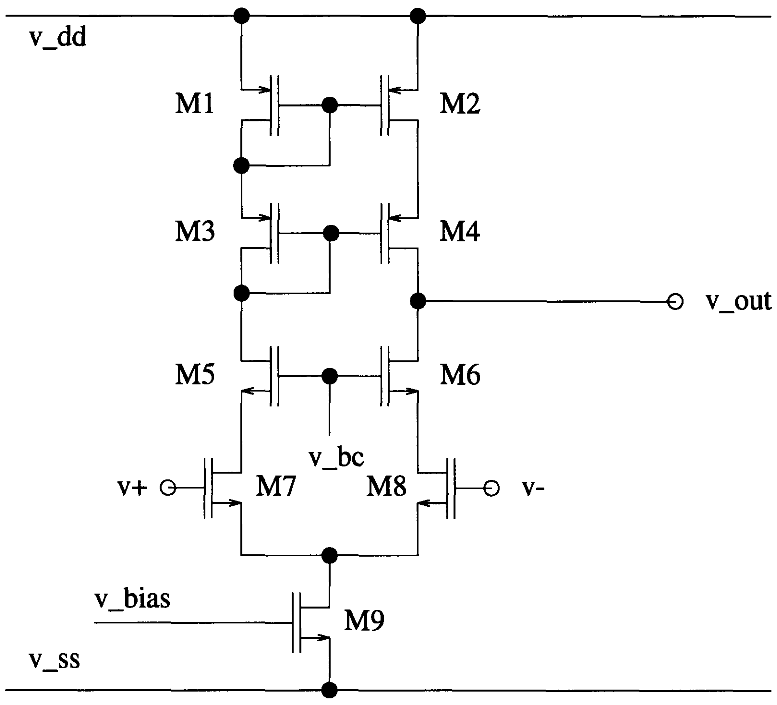


Figure 7-1: Minimal area op amp topology

Optimal Design Parameters		
Design Parameter	Allowable Range	Optimal Value
M_1, M_2	0.5 $\leadsto$ 100	6.72
M_3, M_4	0.5 $\leadsto$ 100	5.14
M_5, M_6	0.5 $\leadsto$ 100	5.16
M_7, M_8	0.5 $\leadsto$ 100	17.9
M_9	0.5 $\leadsto$ 100	1.62
V_{BIAS} (V)	-5 $\leadsto$ 5	-2.29
V_{BC} (V)	-5 $\leadsto$ 5	1.48

Table 7.3: Optimal design parameters for a minimal area op amp

Optimal Design Performance Summary			
Performance	Specification	Design	SPICE
Active area (μm^2)	MINIMIZE	6003	–
DC power (mW)	< 20	.40	.41
Input offset (mV)	$= 0$	.036	.022
Input CMR (V)	-3 $\leadsto$ 3	-1.9 $\leadsto$ 1.5	-4.0 $\leadsto$ 1.5
Output swing (V)	-3 $\leadsto$ 3	-2.3 $\leadsto$ 1.2	-2.3 $\leadsto$ 4.2
Differential gain @ DC (dB)	> 80	90	90
CMRR @ DC (dB)	> 80	53	149
Unity gain bandwidth (MHz)	> 1	.88	.88
Phase margin (degrees)	> 60	90	83
PSRR+ @ DC (dB)	> 80	90	90
PSRR− @ DC (dB)	> 80	90	99
Slew rate (V/μs)	> 2	2	2.02
First stage bias current $I_{(CS)}$ (μA)	–	40	41
Output resistance R_{OUT} ($K\Omega$)	–	288e3	262e3
Dominant pole $pole_I$ (Hz)	–	27.7	–

Table 7.4: Minimal area op amp performance summary

The expected small device size of the current source (sink) transistor M_9 in the optimal design, caused a tendency towards a small bias current, which in turn yielded small sub-milliwatt DC power dissipation number due to the proportionality between the power dissipation, the biasing stage current, and the size of the current sink.

The slew rate and the unity gain bandwidth performances showed a tight "at-spec" behavior. The slew rate, approximately equal to the ratio of the bias current and the loading capacitance C_L sets the lower bias current limit at 40 μA. The unity gain bandwidth is proportional to the transconductance of the input stage, which

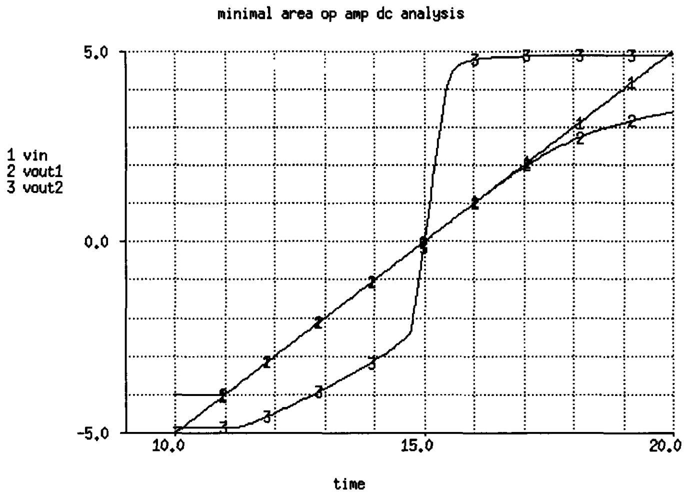


Figure 7-2: DC characteristics of the minimal area op amp

in turn varies as the square root of the size of the differential input devices and the bias current, both of which have a tendency to be as small as possible due to the minimization of the active area. Thus the constraint activity of both the slew rate and the unity gain bandwidth. The poor "at-spec" switching characteristics of this op amp are evident from the transient experiment in Figure 7-4.

The reduction of the positive end of the input CMR, and the negative end of the output swing, particularly evident from the DC plot in Figure 7-2, can be attributed to the additional voltage drop required by the cascode devices in the cascoded differential stage.

A large underestimation discrepancy between the voltage swings as computed by the equations based performance evaluator, and the circuit simulator (SPICE) based evaluator can be easily noticed in the optimal op amp performance summary. In fact, this underestimation was noticed in other design experiments as well. My approximate equations compute the voltage swings by a consideration of the saturation limits for the appropriate devices. In reality, the op amp may continue to operate reasonably well even if some of the devices are in the non-saturation (linear, ohmic)

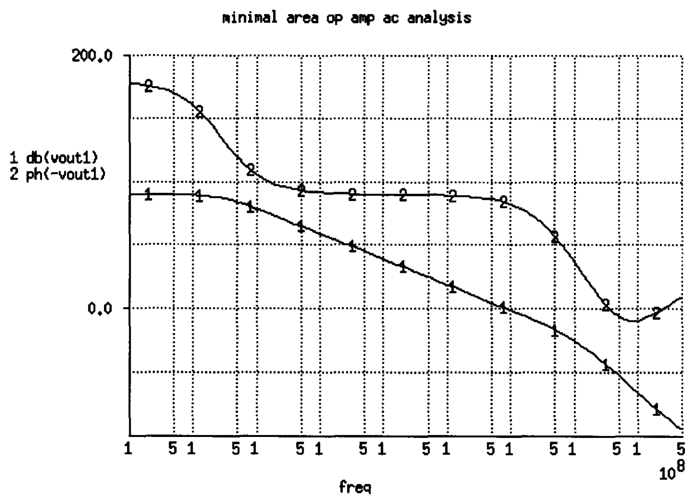


Figure 7-3: Frequency domain behavior of the minimal area op amp

region. As explained in Chapter 4, the circuit simulator based evaluator computes the swing limits not by considering the transistor saturation limits, but by looking at the output voltage waveforms, and thus the discrepancy.

For example, the discrepancy between the equations and simulator computed negative ends of the input CMR can be explained as follows. As the common mode input voltage goes down, the voltage at the `src` node, shared by the sources of the differential input devices M_7 and M_8, and by the drain of the current sink M_9, has to follow. However, the drain-source voltage of the current sink eventually falls below the limit required to keep the current sink in saturation, so this device goes into the non-saturation region, and its sinking current starts diminishing.

This bias current decrease is responsible for effectively "prolonging" the saturation life of the rest of the devices in the op amp, and considerably slows down the expected decrease in the op amp gain. In other words, the gain continues to be sufficiently high for the output voltage waveform to continue to follow the input voltage of the unity-gain negative-feedback configuration until the common mode input voltage reaches -4V, which is a sink's threshold voltage level above V_{SS}, causing the current sink to

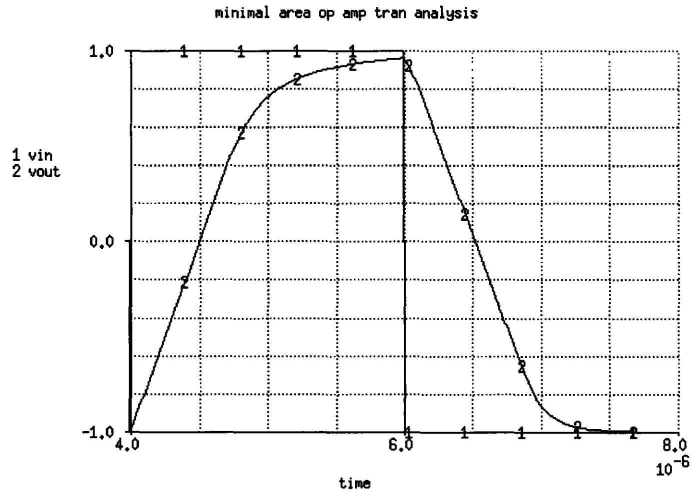


Figure 7-4: Switching characteristics of the minimal area op amp

go into cut-off.

The large discrepancy in the values for the CMRR can be contributed to the incorrectly estimated common-mode gain with my design equations when cascode devices are present in the differential stage. An additional factor of inaccuracy in the approximate computation of the CMRR is the assumption of complete symmetry in the circuit i.e. of a differential output rather than the single-ended output present.

The designed op amp attains a respectable gain and a very large (hundreds of mega-ohms) output resistance due to the cascodes in the differential stage and the current mirror. This means that this op amp approaches an ideal OTA and would be capable of driving only small to medium capacitive, or very large resistive loads. Given that the output load is a single medium sized capacitor of 20 pF, the very large output resistance does not present a problem.

7.2 An Optimal Micropower Op Amp

The power dissipation is another performance which is frequently minimized, especially when the intended application of the chip is in a battery-powered products, such as PC laptops, cellular transceivers, and heart pacemaker implants.

The specifications for my design experiments in designing a micropower op amp were almost identical to the specifications for the minimal area experiments, except for a MINIMIZE requirement for the DC power, and a "< 50000" constraint spec for the active area. The result were a little surprising at first, since they showed convergence to op amps identical to the minimal area op amp both in terms of their topology and design parameter values.

The identical design results can be explained as follows. First, since the output stage current is typically required to be several times greater than the first stage current, op amps containing only a single stage would be vastly superior in terms of power dissipation as compared to two stage topologies. Second, the slew rate requirement sets a lower first stage bias current limit of 40 μA, since the slew rate is the ratio of this current and the capacitive loading C_L. This, in turn, sets a lower DC power limit of .4 mW, which was attained by the minimal area op amp.

7.3 An Optimal High Bandwidth Op Amp

When op amps are used in high frequency applications, such as video products, the maximization of the bandwidth is a common design goal.

Therefore, in this design experiment, I maximized the unity gain with the rest of the performances constrained as shown in Table 7.5. The UGB maximization was formulated as a goal setting search with a goal value of 15MHz. The design parameter limits were identical to those shown for the minimal area experiment in Table 7.1.

The fitness function formulation contained an additional penalty term penalizing all designs in which the output pole was located below the UGB.

Design Performance Specifications	
Performance	Specification
CMOS Technology	SIMPLE $5\mu m$
Ambient temperature (C)	25
Positive power supply V_{DD} (V)	5
Negative power supply V_{SS} (V)	-5
MOS channel lengths L (μm)	10
Loading capacitance C_L (pF)	20
Active area (μm^2)	< 50000
DC power (mW)	< 20
Input offset (mV)	$= 0$
Input CMR (V)	$-3 \leadsto 3$
Output swing (V)	$-3 \leadsto 3$
Differential gain @ DC (dB)	> 80
CMRR @ DC (dB)	> 80
Unity gain bandwidth (MHz)	MAXIMIZE
Phase margin (degrees)	> 60
PSRR+ @ DC (dB)	> 80
PSRR− @ DC (dB)	> 80
Slew rate $(V/\mu s)$	> 2

Table 7.5: Maximal UGB op amp performance specifications

In the initial random population of op amps, the best overall op amp had a topology consisting of a wilson current mirror, a cascode differential stage, a simple current source, a simple output stage, and no compensation. Within the first 10 generations, the poor phase margin and input CMR performance of this topology yielded an introduction of a nulling resistor compensation, and a removal of the cascodes from the differential stage. At the same time, due to an insufficiency in the gain, cascodes were introduced in the output stage.

Between generations 30 and 40, the cascodes were dropped from the output stage yielding an improved input CMR, and lower active area. The gain diminished about 30 dB, but ended up just above its specification level. All this gave the new topology the edge in dominating the evolution for a while.

Sometime between generations 70 and 80, the currently dominating topology was modified for the last time, by a replacement of the wilson current mirror with a simple current mirror. Together with a set of changes in some of the design parameters, the new dominating op amp topology gave a smaller DC power and active area, better input CMR, all which compensated for a 10 dB drop in the gain, which ended just below its spec threshold. For the remainder of the genetic search, this final op amp topology composed of a simple styles of the current mirror, the differential stage, the current source, and the output stage, and a nulling resistor compensation shown in Figure 7.3, converged to the set of optimal design parameters tabulated in Table 7.6.

The performance of the designed high bandwidth op amp is summarized in Table 7.7. The design achieves a gain bandwidth of almost 9 MHz. The DC power dissipation in the designed op amp was high due to the fact that the genetic search attempted to maximize the UGB by an increase of the transconductance of the first stage, which yielded a large first stage bias current. The phase margin, which increases with larger transconductance ratios of the second to the first stage, helped push the output stage current in the milli-amp range.

The total area proves to be an essential resource in maximizing the UGB, with

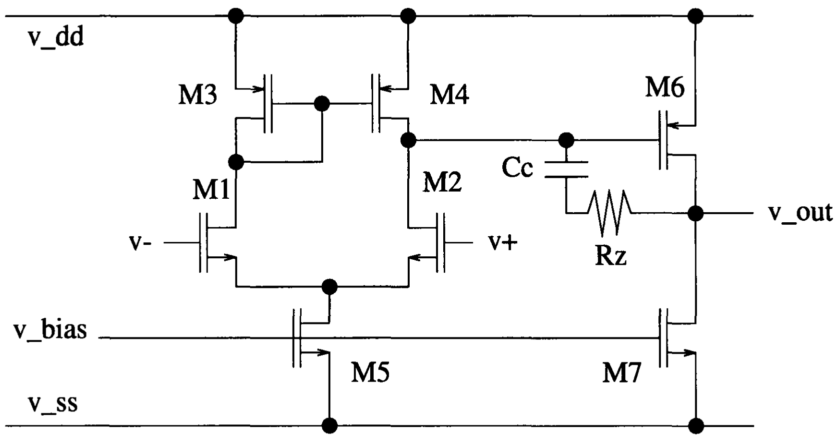


Figure 7-5: Maximal UGB op amp topology

the GA strategy converging to a design with a small constraint violation in the active area. To explain this, we realize that the UGB is to the first order proportional to the transconductance of the first stage, which in turn is proportional to the square roots of the first stage current and the size of the differential input stage transistors M_1 and M_2. In order to achieve a large first stage transconductance, the genetic search decided to push the size of the differential input stage transistors to their range limit.

The large bias currents caused a reduction and led to a small constraint violation in the gain of the op amp. This is due to the first order dependence of the gain on the inverse square root of the bias currents. The large slew rate, well above its constraint limit is basically due to the rapid charging of the compensation capacitance by the large bias current of the first stage.

The combination of a smaller than required stage two to stage one transconductance ratio, a output pole location below the UGB, and a relatively low frequency location of the zero, together with the error in the approximate computation of the phase margin from the transfer function equation, caused a significant discrepancy in the values of the phase margin as computed by the equations based and the simulator based performance evaluators.

Optimal Design Parameters		
Design Parameter	Allowable Range	Optimal Value
M_1, M_2	0.5 $\rightsquigarrow$ 100	99.3
M_3, M_4	0.5 $\rightsquigarrow$ 100	6.99
M_5	0.5 $\rightsquigarrow$ 100	10.7
M_6	0.5 $\rightsquigarrow$ 100	80.9
M_7	0.5 $\rightsquigarrow$ 100	61.9
C_C (pF)	0.1 $\rightsquigarrow$ 20	6.68
R_Z $(k\Omega)$	0 $\rightsquigarrow$ 100	3.12
V_{BIAS} (V)	-5 $\rightsquigarrow$ 5	-2.43

Table 7.6: Optimal design parameters for a maximal UGB op amp

Optimal Design Performance Summary			
Performance	Specification	Design	SPICE
Active area (μm^2)	< 50000	52487	–
DC power (mW)	< 20	15.3	16.0
Input offset (mV)	$= 0$	.03	.09
Input CMR (V)	-3 $\rightsquigarrow$ 3	-2.1 $\rightsquigarrow$ 3.0	-4.0 $\rightsquigarrow$ 4.2
Output swing (V)	-3 $\rightsquigarrow$ 3	-3.4 $\rightsquigarrow$ 3.0	-4.8 $\rightsquigarrow$ 4.4
Differential gain @ DC (dB)	> 80	76	77
CMRR @ DC (dB)	> 80	82	89
Unity gain bandwidth (MHz)	MAXIMIZE	14.7	8.9
Phase margin (degrees)	> 60	63	38
PSRR+ @ DC (dB)	> 80	79	81
PSRR– @ DC (dB)	> 80	85	90
Slew rate (V/μs)	> 2	34	35
Area of C_C (%)	–	29	–
Area of R_Z (%)	–	12	–
First stage bias current $I_{(CS)}$ (μA)	–	226	–
Second stage bias current $I_{(OS)}$ (μA)	–	1304	–
Output resistance R_{OUT} ($K\Omega$)	–	25.6	26.3
Dominant pole $pole_I$ (Hz)	–	2435	–
Output pole $pole_{II}$ (MHz)	–	10.3	–
Compensation pole $pole_{III}$ (MHz)	–	27.1	–
Zero (MHz)	–	10.1	–

Table 7.7: Maximal UGB op amp performance summary

In addition to the error in the approximate computation of the phase margin, there is also a discrepancy in the UGB values, which also comes from the fact that the system function considered in my model is only a rough approximation.

The simulation plots of the optimal high bandwidth op amp are given in Figure 7-6, Figure 7-7, and Figure 7-8. The solidly wide voltage swings achieved by the elimination of all cascode devices during the genetic search are evident from the DC characteristics in Figure 7-6. Figure 7-7 demonstrates the maximized 9MHz unity gain bandwidth. The excellent switching response of the designed opamp can be seen from the transient plot in Figure 7-8.

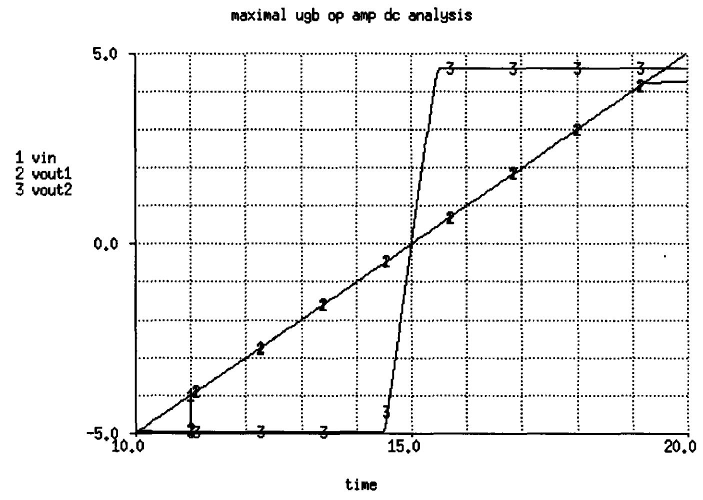


Figure 7-6: DC characteristics of the maximal UGB op amp

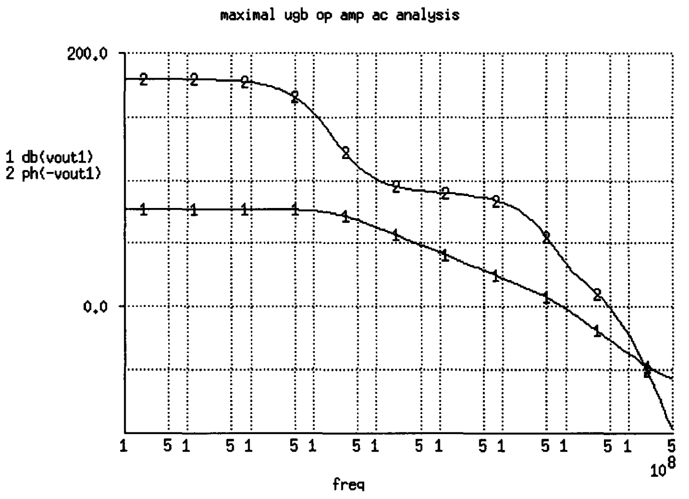


Figure 7-7: Frequency domain behavior of the maximal UGB op amp

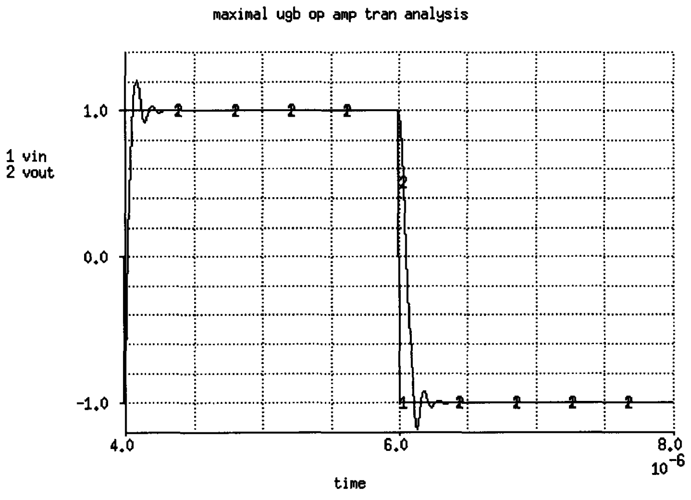


Figure 7-8: Switching characteristics of the maximal UGB op amp

7.4 In Search of A High Gain Op Amp

When I set up my system to search for an maximal gain op amp, I expected to see evolving op amp populations dominated with cascoded topologies. The results confirmed my expectations. The genetic searches readily converged to a topology featuring a cascode current mirror, a cascode differential stage, a simple current source, a cascode output stage, and a nulling resistor compensation. The GA strategy cascoded every module it could in order to maximize the gain.

While the optimal high gain op amp was evaluated to feature a vast 180 dB gain, the more accurate simulator based evaluator computed a 144 dB gain. This gain level may, however, turn out to be impractical due to the large noise figures for MOS op amps.

The errors in the approximate computation of the phase margin were particularly evident in this design experiment.

The unity gain bandwidth and the slew rate showed the expected tendency towards smaller values due to the maximization of the gain. In addition, the genetic search initially traded the higher gain for large constraint violations of these constraints, which was fixed by treating these two constraints in an exponential manner as explained in Chapter 5.

Chapter 8

Conclusion

This thesis describes my design and implementation of a system for automated synthesis of CMOS op amps. In the concluding chapter, I will focus on the contributions and future work of this thesis.

8.1 Contributions of Thesis

My op amp design system assembles the op amps by composing subcircuit modules. Each module contains design knowledge in a set of equations. This flexible, object-oriented knowledge-based approach allows the design of a large number of op amp topologies, and an easy addition of a new subcircuit building block to the module library.

My system controls the design search with an incredibly robust genetic algorithm. To my knowledge, this is a pioneering attempt to use GAs in analog circuit design.

This thesis also introduces a successful original approach in the op amp topology selection process. Unlike all other design systems, in which the topology selection is carried by a heuristic rule-based procedure, my system integrates and intertwines the search through the topology space with the search through the space of design parameters.

My novel genetic op amp design strategy arrives at optimal designs. It's genetic properties guarantee an efficient, exhaustive, global, and unbiased autonomous search through both the the topology and the design parameter spaces.

Throughout the design search, my system evaluates the performance of the op amps by solving approximate design equations using a combination of symbolic transformations and numerical techniques. An alternative, more accurate, but more expensive, and less robust circuit simulator based performance evaluator was also implemented, and is mainly used to perform verifications of the optimal designs.

The formality of my equations-solving approach allowed me to straightforwardly implement the often used, but inferior, human designer mimicking (first-cut) strategy for op amp design as an alternative design method. I also presented an attempt at an automated generation of first-cut design procedures.

The experimental results in designing optimal op amps for varied applications, verify my system's ability to automate the design synthesis of optimal CMOS op amps in a novel way.

8.2 Future Work

Many modifications and additions can be undertaken leading to improvements in my op amp design system.

An incorporation of more sophisticated MOS device models, and more complicated and accurate subcircuit and op amp design equations should lead towards a more accurate equations based performance evaluator.

Additional constraints on the locations of the poles and zeros as well as other computed parameters could be used to increase the efficiency of the search.

Implementing a number of different shemes for converting a constrained into a non-constrained problem, as necessary for the construction of the GA fitness function, would perhaps point the way to a better GA lock on the feasible region.

A study featuring implementations of different gene orderings, and varied coding schemes could also give valuable insights into the relationship between the efficiency of the genetic op amp optimizations and the incorporated coding approach.

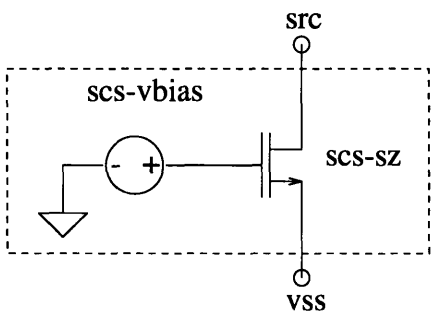
Integrating gradient information, and even gradient based methods with the GA strategy should result in a more powerful system.

New, potentially more powerful genetic operators could be added to the genetic op amp search. Methods such as sharing [10], feasibility region sampling [25], or some of our new and original, but not yet implemented ideas such as selective allele freezing, dynamic scaling of penalty terms, a structured method for manufacturing the initial op amp population, a biasing control of the GA from the symbolic equation structure, and others may lead towards significant improvements in the genetic algorithm based op amp design strategy.

Appendix A

Subcircuit Modules and Circuit Components

A.1 Circuit Diagrams of Subcircuit Modules



Simple Current Source

Figure A-1: Current source module design styles

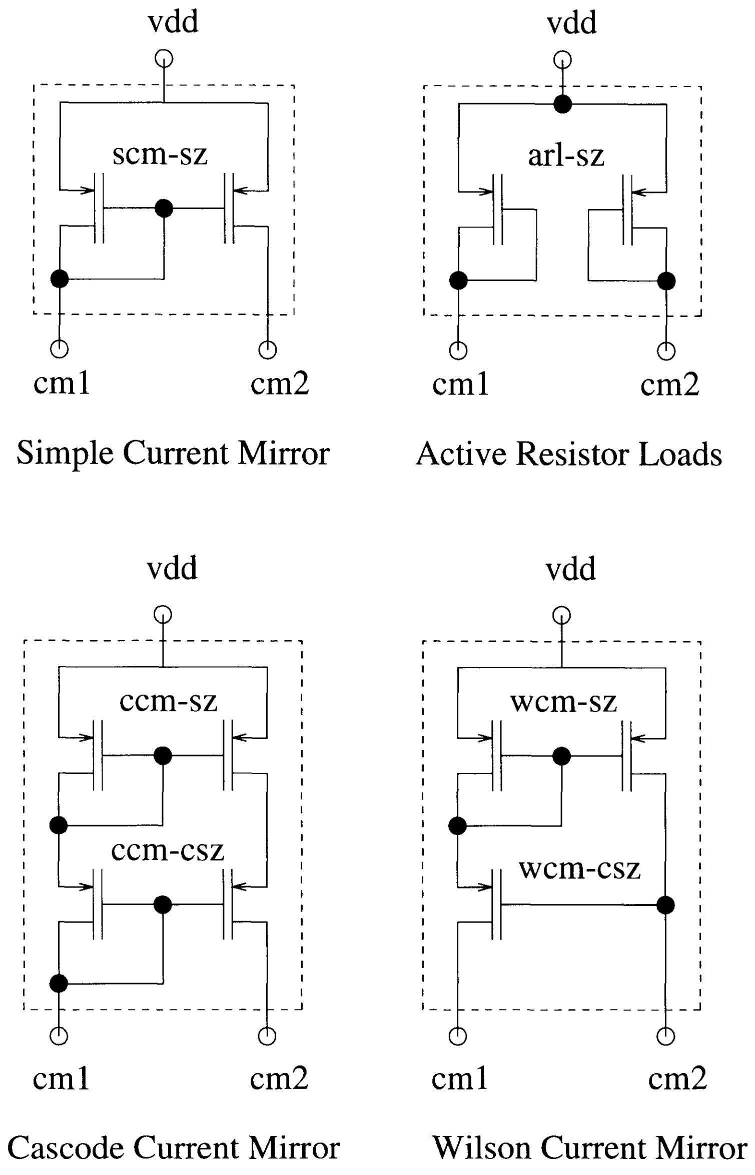
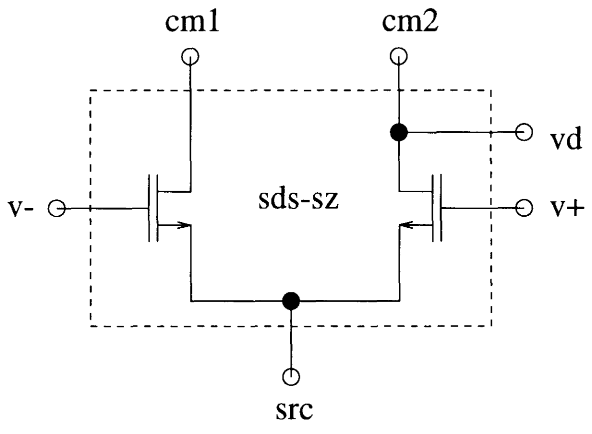
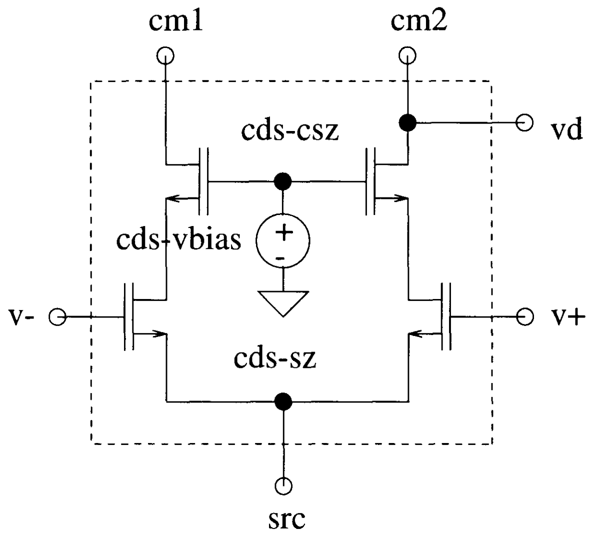


Figure A-2: Current mirror module design styles



Simple Differential Stage



Cascode Differential Stage

Figure A-3: Differential stage module design styles

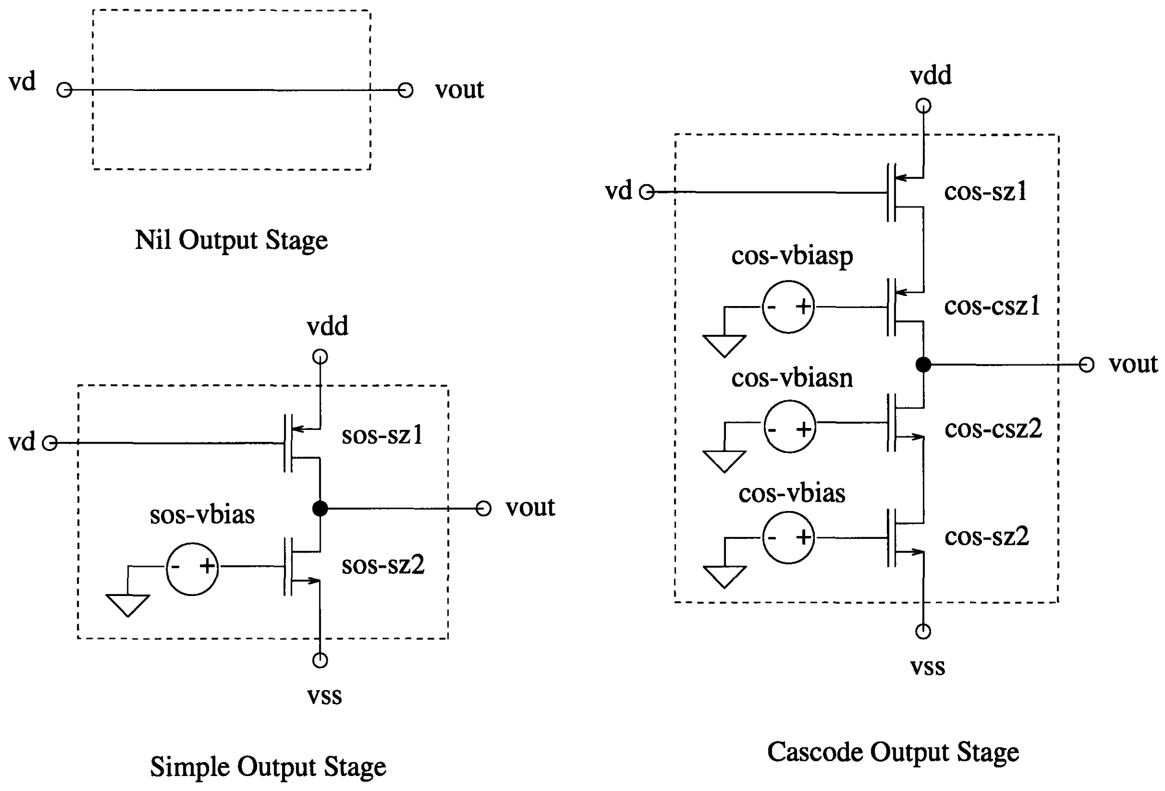


Figure A-4: Output stage module design styles

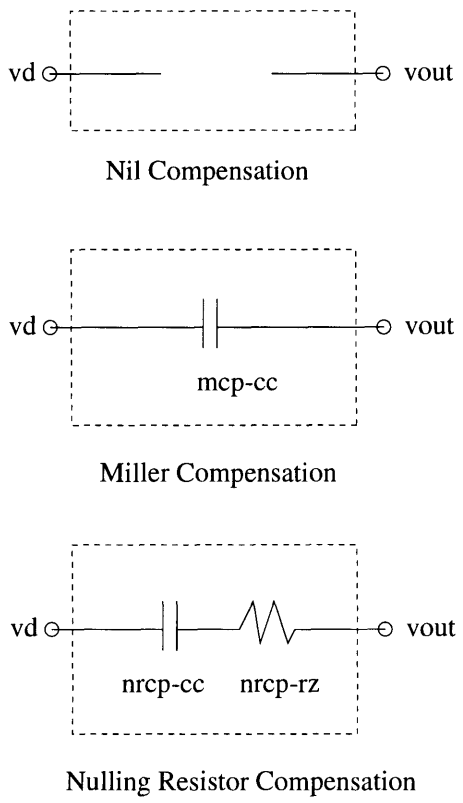


Figure A-5: Compensation module design styles

A.2 Design Equations for the Op Amp Object

```
(define opamp:design-eqs
  ;; note: vp is v+, vn is v-
  '((= area (+ cm-area ds-area cs-area os-area cp-area))
    (= slew-rate (if (and compensation? output-stage?)
                     (/ cs-i cc)
                     (if (and (not compensation?) output-stage?)
                         (/ os-i c-ii)
                         (/ cs-i c-i))))
    (= ds-i (* 1/2 cs-i))
    (= cm-i (* 1/2 cs-i))
    (= unity-gain-bandwidth (if output-stage?
                                (if compensation?
                                    (/ gm-i cc 2 pi)
                                    (* gain-i gain-ii pole-ii))
                                (/ gm-i c-i 2 pi)))
    (= input-cmr+ (- vdd v-vdd-to-cm v-cm-to-vp))
    (= input-cmr- (+ v-vp-to-src v-src-to-vss vss))
    (= dc-power (* (+ cs-i os-i) (- vdd vss)))
    (= output-cmr+ (- vdd v-vdd-to-vout))
    (= output-cmr- (+ v-vout-to-vss vss))
    (= input-offset (/ del-os-vgs1 gain-i))
    (= gain-i (* gm-i r-i))
    (= gain-ii (* gm-ii r-ii))
    (= dc-gain (decibel (* gain-i gain-ii)))
    (= cm-gain (decibel (/ (* ds-gm cm-rout)
                           (+ 1. (* 2. ds-gm cs-rout)))))
    (= cmrr (- dc-gain cm-gain))
    (= r-i (// r-vd-to-src cm-rout))
    (= r-ii os-rout)
    (= r-out r-ii)
    (= gm-i ds-gm)
    (= gm-ii os-gm)
    (= gm-ii/gm-i (/ gm-ii gm-i))
    (= l-default (* l-factor min-active-width))
    (= s (* +i 2 pi frequency))
    (= transfer-fn-db (decibel (magnitude transfer-fn)))
    (= transfer-fn-ph (* (/ 180 pi) (angle (- transfer-fn))))))
```

A.3 Subcircuit Objects

A.3.1 Current Mirrors (CM)

Simple Current Mirror (SCM)

```
(define simple-current-mirror
  (lambda (m)
    (case m
      ((type) 'current-mirror)
      ((name) 'simple-current-mirror)
      ((params) '(scm-w scm-l))
      ((make) (lambda (model w l)
                (list '(* simple current mirror)
                  ((pmos-tran 'make)
                   'scm1 'cm1 'cm1 'vdd 'vdd model w l)
                  ((pmos-tran 'make)
                   'scm2 'cm2 'cm1 'vdd 'vdd model w l))))
      ((area) '(* 2 scm-w scm-l))
      ((design) '((= cm-area (* 2 scm-w scm-l))
                  (= v-vdd-to-cm
                     (- (sqrt (/ (* 2 cm-i) kp scm-sz)) (vtp 0)))
                  (= cm-rout (/ 1 (gdsp scm-l cm-i)))
                  (= scm-w (* scm-sz (leff scm-l ldp)))))
      (else (error "Unknown message -- simple-current-mirror.")))))
```

Cascode Current Mirror (CCM)

```
(define cascode-current-mirror
  (lambda (m)
    (case m
      ((type) 'current-mirror)
      ((name) 'cascode-current-mirror)
      ((params) '(ccm-w ccm-l ccm-cw ccm-cl))
      ((make) (lambda (model w l cw cl)
                (list '(* cascoded current mirror)
                      ((pmos-tran 'make)
                       'ccm1 'ccm1 'ccm1 'vdd 'vdd model w l)
                      ((pmos-tran 'make)
                       'ccm2 'ccm2 'ccm1 'vdd 'vdd model w l)
                      ((pmos-tran 'make)
                       'ccmc1 'cm1 'cm1 'ccm1 'vdd model cw cl)
                      ((pmos-tran 'make)
                       'ccmc2 'cm2 'cm1 'ccm2 'vdd model cw cl))))
      ((area) '(+ (* 2 ccm-w ccm-l) (* 2 ccm-cw ccm-cl)))
      ((design) '((= cm-area (+ (* 2 ccm-w ccm-l)
                                (* 2 ccm-cw ccm-cl)))
                  (= v-vdd-to-cm
                     (+ (sqrt (/ (* 2 cm-i) kp ccm-sz))
                        (abs (vtp 0))
                        (sqrt (/ (* 2 cm-i) kp ccm-csz))
                        (abs (vtp 0))))
                  (= ccm-csz (* ccm-sz-factor ccm-sz))
                  (= ccm-rds (/ 1 (gdsp ccm-l cm-i)))
                  (= ccm-rdsc (/ 1 (gdsp ccm-cl cm-i)))
                  (= ccm-gmc (gmp ccm-csz cm-i kp))
                  (= ccm-etapc (etap 0))
                  (= cm-rout (+ ccm-rds ccm-rdsc
                                (* ccm-gmc ccm-rds
                                   ccm-rdsc (+ 1 ccm-etapc))))
                  (= ccm-w (* ccm-sz (leff ccm-l ldp)))
                  (= ccm-cw (* ccm-csz (leff ccm-cl ldp)))))
      (else (error "Unknown message -- cascode-current-mirror.")))))
```

Wilson Current Mirror (WCM)

```
(define wilson-current-mirror
  (lambda (m)
    (case m
      ((type) 'current-mirror)
      ((name) 'wilson-current-mirror)
      ((params) '(wcm-w wcm-l wcm-cw wcm-cl))
      ((make) (lambda (model w l cw cl)
                (list '(* wilson current mirror)
                      ((pmos-tran 'make)
                       'wcm1 'wcm1 'wcm1 'vdd 'vdd model w l)
                      ((pmos-tran 'make)
                       'wcm2 'cm2 'wcm1 'vdd 'vdd model w l)
                      ((pmos-tran 'make)
                       'wcmc1 'cm1 'cm2 'wcm1 'vdd model cw cl))))
      ((area) '(+ (* 2 wcm-w wcm-l) (* wcm-cw wcm-cl)))
      ((design) '((= cm-area (+ (* 2 wcm-w wcm-l)
                                (* wcm-cw wcm-cl)))
                  (= v-vdd-to-cm
                     (+ (sqrt (/ (* 2 cm-i) kp wcm-sz))
                        (abs (vtp 0))
                        (sqrt (/ (* 2 cm-i) kp wcm-csz))
                        (abs (vtp 0))))
                  (= wcm-csz (* wcm-sz-factor wcm-sz))
                  (= wcm-rds (/ 1 (gdsp wcm-l cm-i)))
                  (= wcm-rdsc (/ 1 (gdsp wcm-cl cm-i)))
                  (= wcm-gm (gmp wcm-sz cm-i kp))
                  (= wcm-gmc (gmp wcm-csz cm-i kp))
                  (= wcm-etapc (etap 0))
                  (= cm-rout
                     (+ wcm-rdsc
                        (* wcm-rds
                           (/ (+ 1 (* wcm-rdsc wcm-gmc
                                      (+ 1 wcm-etapc))
                                 (* wcm-gm wcm-rds wcm-gmc wcm-rdsc))
                              (+ 1 (* wcm-gm wcm-rds))))))
                  (= wcm-w (* wcm-sz (leff wcm-l ldp)))
                  (= wcm-cw (* wcm-csz (leff wcm-cl ldp)))))
      (else (error "Unknown message -- wilson-current-mirror.")))))
```

Active Resistor Loads (ARL)

```
(define active-resistor-loads
  (lambda (m)
    (case m
      ((type) 'current-mirror)
      ((name) 'active-resistor-loads)
      ((params) '(arl-w arl-l))
      ((make) (lambda (model w l)
                (list '(* active resistor loads)
                      ((pmos-tran 'make)
                       'arl1 'cm1 'cm1 'vdd 'vdd model w l)
                      ((pmos-tran 'make)
                       'arl2 'cm2 'cm2 'vdd 'vdd model w l))))
      ((area) '(* 2 arl-w arl-l))
      ((design) '((= cm-area (* 2 arl-w arl-l))
                  (= v-vdd-to-cm
                     (+ (sqrt (/ (* 2 cm-i) kp arl-sz))
                        (abs (vtp 0))))
                  (= arl-gm (gmp arl-sz cm-i kp))
                  (= cm-rout (/ 1 arl-gm))
                  (= arl-w (* arl-sz (leff arl-l ldp)))))
      (else (error "Unknown message -- active-resistor-loads.")))))
```

A.3.2 Differential Stages (DS)

Simple Differential Stage (SDS)

```
(define simple-diff-stage
  (lambda (m)
    (case m
      ((type) 'diff-stage)
      ((name) 'simple-diff-stage)
      ((params) '(sds-w sds-l))
      ((make) (lambda (model w l)
                (list '(* simple differential stage)
                      ((nmos-tran 'make)
                       'sds1 'cm1 'vp 'src 'vss model w l)
                      ((nmos-tran 'make)
                       'sds2 'cm2 'vn 'src 'vss model w l)
                      ((wire 'make)
                       'sds1 'cm2 'vd))))
      ((area) '(* 2 sds-w sds-l))
      ((design) '((= ds-area (* 2 sds-w sds-l))
                  (= v-cm-to-vp
                     (- (vtn 0)))
                  (= v-vp-to-src (+ (sqrt (/ (* 2 ds-i) kn sds-sz))
                                    (vtn 0)))
                  (= ds-gm (gmn sds-sz ds-i kn))
                  (= sds-gds (gdsn sds-l ds-i))
                  (= r-vd-to-src (/ 1 sds-gds))
                  (= sds-w (* sds-sz (leff sds-l ldn)))))
      (else (error "Unknown message -- simple-diff-stage.")))))
```

Cascode Differential Stage (CDS)

```
(define cascode-diff-stage
  (lambda (m)
    (case m
      ((type) 'diff-stage)
      ((name) 'cascode-diff-stage)
      ((params) '(cds-w cds-l cds-cw cds-cl cds-vbias))
      ((make) (lambda (model w l cw cl vbias)
                (list '(* cascoded differential stage)
                      ((nmos-tran 'make)
                       'cds1 'cds1 'vp 'src 'vss model w l)
                      ((nmos-tran 'make)
                       'cds2 'cds2 'vn 'src 'vss model w l)
                      ((nmos-tran 'make)
                       'cdsc2 'vd 'cds3 'cds2 'vss model cw cl)
                      ((nmos-tran 'make)
                       'cdsc1 'cm1 'cds3 'cds1 'vss model cw cl)
                      ((voltage-source 'make)
                       'cds1 'cds3 0 vbias)
                      ((wire 'make)
                       'cds1 'cm2 'vd))))
      ((area) '(+ (* 2 cds-w cds-l) (* 2 cds-cw cds-cl)))
      ((design) '((= ds-area (+ (* 2 cds-w cds-l)
                                (* 2 cds-cw cds-cl)))
                  (= v-cm-to-vp
                     (- (sqrt (/ (* 2 ds-i) kn cds-csz)) (vtn 0)))
                (= cds-csz (* cds-sz-factor cds-sz))
                (= v-vp-to-src (+ (sqrt (/ (* 2 ds-i) kn cds-sz))
                                  (vtn 0)))
                (= ds-gm (gmn cds-sz ds-i kn))
                (= cds-gmc (gmn cds-csz ds-i kn))
                (= cds-gds (gdsn cds-l ds-i))
                (= cds-gdsc (gdsn cds-cl ds-i))
                (= r-vd-to-src (* cds-gmc (/ 1 cds-gdsc)
                                  (/ 1 cds-gds)))
                (= cds-w (* cds-sz (leff cds-l ldn)))
                (= cds-cw (* cds-csz (leff cds-cl ldn)))
                (= cds-vbias (+ (if output-stage?
                                    (- vdd v-vdd-to-cm)
                                    0)
                                (vtn 0)
                                cds-vbias-offset))))
      (else (error "Unknown message -- cascode-diff-stage.")))))
```

A.3.3 Current Sources (CS)

Simple Current Source (SCS)

```
(define simple-current-source
  (lambda (m)
    (case m
      ((type) 'current-source)
      ((name) 'simple-current-source)
      ((params) '(scs-w scs-l scs-vbias))
      ((make) (lambda (model w l vbias)
                (list '(* simple current source)
                      ((nmos-tran 'make)
                       'scs1 'src 'scs1 'vss 'vss model w l)
                      ((voltage-source 'make)
                       'scs1 'scs1 0 vbias))))
      ((area) '(* scs-w scs-l))
      ((design) '((= cs-area (* scs-w scs-l))
                  (= v-src-to-vss (sqrt (/ (* 2 cs-i) kn scs-sz)))
                  (= vbias (+ v-src-to-vss (vtn 0) vss))
                  (= cs-i (idn-sat scs-sz scs-vgs 0 kn))
                  (= scs-vgs (- vbias vss))
                  (= scs-w (* scs-sz (leff scs-l ldn)))
                  (= cs-rout (/ 1 (gdsn scs-l cs-i)))))
      (else (error "Unknown message -- simple-current-source.")))))
```

A.3.4 Output Stages (OS)

Nil Output Stage (NOS)

```
(define nil-output-stage
  (lambda (m)
    (case m
      ((type) 'output-stage)
      ((name) 'nil-output-stage)
      ((params) '())
      ((make) (lambda (model)
                (list '(* nil output stage)
                      ((wire 'make)
                       'nos1 'vd 'vout))))
      ((area) '0)
      ((design) '((os-area 0)
                  (= output-stage? (false-val))
                  (= os-i 0)
                  (= del-os-vgs1 (- vdd v-vdd-to-cm))
                  (= v-vdd-to-vout v-vdd-to-cm)
                  (= v-vout-to-vss (+ v-cm-to-vp v-vp-to-src
                                      v-src-to-vss))
                  (= os-gm 1)
                  (= os-rout r-i)
                  (= c-i cl)
                  (= psrr+ (decibel gain-i))
                  (= psrr- (decibel gain-i))))
      (else (error "Unknown message -- nil-output-stage.")))))
```

Simple Output Stage (SOS)

```
(define simple-output-stage
  (lambda (m)
    (case m
      ((type) 'output-stage)
      ((name) 'simple-output-stage)
      ((params) '(sos-w1 sos-l1 sos-w2 sos-l2 sos-vbias))
      ((make) (lambda (model w1 l1 w2 l2 vbias)
                (list '(* simple output stage)
                      ((pmos-tran 'make)
                       'sos1 'vout 'vd 'vdd 'vdd model w1 l1)
                      ((nmos-tran 'make)
                       'sos2 'vout 'sos1 'vss 'vss model w2 l2)
                      ((voltage-source 'make)
                       'sos1 'sos1 0 vbias))))
      ((area) '(+ (* sos-w1 sos-l1) (* sos-w2 sos-l2)))
      ((design) '((= os-area (+ (* sos-w1 sos-l1) (* sos-w2 sos-l2)))
                  (= output-stage? (true-val))
                  (= os-i* (idn-sat sos-sz2 (- vbias vss) 0 kn))
                  (= os-i (idp-sat sos-sz1 (- v-vdd-to-cm) 0 kp))
                  (= del-os-vgs1 (/ (- (sqrt os-i*) (sqrt os-i))
                                    (sqrt (* 0.5 kp sos-sz1))))
                  (= v-vdd-to-vout (sqrt (/ (* 2 os-i) kp sos-sz1)))
                  (= v-vout-to-vss (sqrt (/ (* 2 os-i) kn sos-sz2)))
                  (= os-gm (gmp sos-sz1 os-i kp))
                  (= sos-gds1 (gdsp sos-l1 os-i))
                  (= sos-gds2 (gdsn sos-l2 os-i))
                  (= os-rout (/ 1 (+ sos-gds1 sos-gds2)))
                  (= sos-w1 (* sos-sz1 (leff sos-l1 ldp)))
                  (= sos-w2 (* sos-sz2 (leff sos-l2 ldn)))
                  (= c-i (cgsp sos-w1 sos-l1))
                  (= c-ii cl)
                  (= psrr+ (decibel (* gain-i (/ gm-ii sos-gds1))))
                  (= psrr- (decibel (* gain-i (/ gm-ii sos-gds2))))))
      (else (error "Unknown message -- simple-output-stage.")))))
```

Cascode Output Stage (COS)

```
(define cascode-output-stage
  (lambda (m)
    (case m
      ((type) 'output-stage)
      ((name) 'cascode-output-stage)
      ((params) '(cos-w1 cos-l1 cos-w2 cos-l2
                  cos-vbias cos-vbiasn cos-vbiasp))
      ((make) (lambda (model w1 l1 w2 l2 vbias vbiasn vbiasp)
                (list '(* cascoded output stage)
                      ((pmos-tran 'make)
                       'cos1 'cos2 'vd 'vdd 'vdd model w1 l1)
                      ((pmos-tran 'make)
                       'cosc1 'vout 'cos3 'cos2 'vdd model w1 l1)
                      ((voltage-source 'make)
                       'cos1 'cos3 0 vbiasp)
                      ((nmos-tran 'make)
                       'cosc2 'vout 'cos4 'cos1 'vss model w2 l2)
                      ((voltage-source 'make)
                       'cos2 'cos4 0 vbiasn)
                      ((nmos-tran 'make)
                       'cos2 'cos1 'cos5 'vss 'vss model w2 l2)
                      ((voltage-source 'make)
                       'cos3 'cos5 0 vbias))))
      ((area) '(+ (* 2 cos-w1 cos-l1) (* 2 cos-w2 cos-l2)))
      ((design) '((= os-area (+ (* 2 cos-w1 cos-l1)
                                (* 2 cos-w2 cos-l2)))
                  (= output-stage? (true-val))
                  (= os-i* (idn-sat cos-sz2 (- vbias vss) 0 kn))
                  (= os-i (idp-sat cos-sz1 (- v-vdd-to-cm) 0 kp))
                  (= del-os-vgs1 (/ (- (sqrt os-i*) (sqrt os-i))
                                    (sqrt (* 0.5 kp cos-sz1))))
                  (= v-vdd-to-vout
                     (* 2 (sqrt (/ (* 2 os-i) kp cos-sz1))))
                  (= v-vout-to-vss
                     (* 2 (sqrt (/ (* 2 os-i) kn cos-sz2))))
                  (= cos-vds-sat-1 (- (- v-vdd-to-cm) (vtp 0)))
                  (= cos-vds-sat-2 (- (- vbias vss) (vtn 0)))
                  (= cos-vbiasp (+ vdd cos-vds-sat-1 (- v-vdd-to-cm)
                                   (- cos-vbias-offset)))
                  (= cos-vbiasn (+ vss cos-vds-sat-2 (- vbias vss)
                                   cos-vbias-offset))
                  (= os-gm (gmp cos-sz1 os-i kp))
                  (= cos-gmc1 (gmp cos-sz1 os-i kp))
                  (= cos-gmc2 (gmn cos-sz2 os-i kn))
                  (= cos-gds1 (gdsp cos-l1 os-i))
                  (= cos-gdsc1 (gdsp cos-l1 os-i))
                  (= cos-gds2 (gdsp cos-l2 os-i))
                  (= cos-gdsc2 (gdsp cos-l2 os-i))
                  (= os-rout
                     (// (* cos-gmc1 (/ 1 cos-gdsc1) (/ 1 cos-gds1))
                         (* cos-gmc2 (/ 1 cos-gdsc2) (/ 1 cos-gds2))))
                  (= cos-w1 (* cos-sz1 (leff cos-l1 ldp)))
                  (= cos-w2 (* cos-sz2 (leff cos-l2 ldn)))
                  (= c-i (cgsp cos-w1 cos-l1))
                  (= c-ii cl)
                  (= psrr+
                     (decibel (* gain-i
                                 (/ gm-ii (// cos-gds1 cos-gdsc1)))))
```

```
              (= psrr-
                 (decibel (* gain-i
                             (/ gm-ii (// cos-gds2 cos-gdsc2)))))))
      (else (error "Unknown message -- cascode-output-stage.")))))
```

A.3.5 Compensation Schemes (CP)

Nil Compensation (NCP)

```
(define nil-compensation
  (lambda (m)
    (case m
      ((type) 'compensation)
      ((name) 'nil-compensation)
      ((params) '())
      ((make) (lambda (model)
                (list '(* nil compensation))))
      ((area) '0)
      ((design) '((= cp-area 0)
                  (= compensation? (false-val))
                  (= phase-margin
                     (if output-stage?
                         (* (/ 180 pi)
                            (angle
                             (- 1 (* +i (/ 1 gm-ii/gm-i)))))
                         (* (/ 180 pi)
                            (angle
                             (/ (* gm-i r-i)
                                (+ 1
                                   (* +i
                                      (sqrt (- (expt (* gm-i r-i) 2)
1)
                                                 ))))))))
                  (= pole-i (/ 1 r-i c-i 2 pi))
                  (= pole-ii
                     (if output-stage?
                         (/ 1 (* r-ii c-ii) 2 pi)
                         1e100))
                  (= zero 1e100)
                  (= transfer-fn
                     (if output-stage?
                         (/ (* gm-i gm-ii r-i r-ii)
                            (+ 1
                               (* s (+ (* r-i c-i) (* r-ii c-ii)))
                               (* s s r-i r-ii (+ (* c-i c-ii)))))
                         (/ (* gm-i r-i)
                            (+ 1 (* s r-i c-i)))))))
      (else (error "Unknown message -- nil-compensation.")))))
```

Miller Compensation (MCP)

```
(define miller-compensation
  (lambda (m)
    (case m
      ((type) 'compensation)
      ((name) 'miller-compensation)
      ((params) '(mcp-cc))
      ((make) (lambda (model cc)
                (list '(* miller compensation)
                      ((capacitor 'make)
                       'mcp1 'vout 'vd cc))))
      ((area) '(/ mcp-cc cox))
      ((design) '((= cp-area (/ mcp-cc cox))
                  (= compensation? (true-val))
                  (= phase-margin
                     (* (/ 180 pi)
                        (angle (/ (- 1 (* +i (/ 1 gm-ii/gm-i)))
                                  (- 1 (* +i gm-ii/gm-i (/ cc c-ii)))))))
                  (= pole-i (/ 1 (* gm-ii r-i r-ii cc) 2 pi))
                  (= pole-ii (/ (* gm-ii cc)
                                (+ (* c-i c-ii)
                                   (* c-ii cc) (* c-i cc)) 2 pi))
                  (= zero (/ gm-ii cc 2 pi))
                  (= transfer-fn
                     (/ (* gm-i gm-ii r-i r-ii (- 1 (* s (/ cc gm-ii))))
                        (+ 1
                           (* s (+ (* r-i (+ c-i cc))
                                   (* r-ii (+ c-ii cc))
                                   (* gm-ii r-i r-ii cc)))
                           (* s s r-i r-ii (+ (* c-i c-ii)
                                              (* cc (+ c-i c-ii))))))))))
      (else (error "Unknown message -- miller-compensation.")))))
```

Nulling Resistor Compensation (NRCP)

```
(define nulling-resistor-compensation
  (lambda (m)
    (case m
      ((type) 'compensation)
      ((name) 'nulling-resistor-compensation)
      ((params) '(nrcp-cc nrcp-rz))
      ((make) (lambda (model cc rz)
                (list '(* nulling resistor compensation)
                      ((capacitor 'make)
                       'nrcp1 'vout 'nrcp1 cc)
                      ((resistor 'make)
                       'nrcp1 'vd 'nrcp1 rz))))
      ((area) '(+ (/ nrcp-cc cox)
                  (* 2 (expt min-active-width 2)
                     (/ nrcp-rz r-square))))
      ((design) '((= cp-area
                     (+ (/ nrcp-cc cox)
                        (* 2 (expt min-active-width 2)
                           (/ nrcp-rz r-square))))
                  (= compensation? (true-val))
                  (= cc-min (sqrt (* (/ 1 gm-ii/gm-i) c-i c-ii)))
                  (= rz (* (/ (+ cc c-ii) cc) (/ 1 gm-ii)))
                  (= phase-margin
                     (* (/ 180 pi)
                        (angle
                          (/ (+ 1 (* +i (/ 1 gm-ii/gm-i) c-ii (/ 1 c-i)))
                             (+ 1 (* -i gm-ii/gm-i c-i (/ 1 c-ii)))))))
                  (= pole-i (/ 1 (* gm-ii r-ii r-i cc) 2 pi))
```

```
            (= pole-ii (/ gm-ii c-ii 2 pi))
            (= pole-iii (/ 1 (* rz c-i) 2 pi))
            (= zero (/ 1 (* cc (- (/ 1 gm-ii) rz)) 2 pi))
            (= transfer-fn
               (/ (* gm-i gm-ii r-i r-ii
                     (- 1 (* s (- (/ cc gm-ii) (* rz cc)))))
                  (+ 1
                     (* s (+ (* (+ c-ii cc) r-ii)
                             (* (+ c-i cc) r-i)
                             (* gm-ii r-i r-ii cc)
                             (* rz cc)))
                     (* s s (+ (* r-i r-ii (+ (* c-i c-ii)
                                              (* cc c-i)
                                              (* cc c-ii)))
                               (* rz cc (+ (* r-i c-i)
                                           (* r-ii c-ii)))))
                     (* s s s r-i r-ii rz c-i c-ii cc))))))
    (else (error "Unknown message -- nulling-resistor-compensation.")))))
```

A.4 Circuit Component Objects

A.4.1 MOS Transistors

NMOS Transistor

```
(define nmos-tran
  (lambda (m)
    (case m
      ((type) 'transistor)
      ((name) 'nmos-transistor)
      ((make) (lambda (name drain gate source bulk model w l)
                (let* ((ad (components:drain/source-area
                             model w))
                       (as ad)
                       (pd (components:drain/source-perimeter
                             model w))
                       (ps pd)
                       (nrd (components:drain/source-resistance-squares
                              model w))
                       (nrs nrd))
                  (list (glue 'm name) (glue '% drain)
                        (glue '% gate) (glue '% source)
                        (glue '% bulk) (glue 'NMOS model)
                        (glue 'w= w) (glue 'l= l)
                        (glue 'ad= ad 'p) (glue 'as= as 'p)
                        (glue 'pd= pd 'u) (glue 'ps= ps 'u)
                        (if (not (eq? nrd 'computed-from-rsdw))
                            (glue 'nrd= nrd " " 'nrs= nrs)
                            (glue " " " "))))))
      ((params) '(nmos-tran-w nmos-tran-l))
      ((area) '(* nmos-tran-w nmos-tran-l))
      ((design) '((= (vtn vsb)
                     (+ vton (* gamman (- (sqrt (+ phin vsb))
                                          (sqrt phin)))))
                  (= (idn-sat sz vgs vsb kn)
                     (* 1/2 kn sz (expt (- vgs (vtn vsb)) 2)))
                  (= (etan vsb)
                     (/ gamman (* 2 (sqrt (+ phin vsb)))))
                  (= (leff l ld)
                     (- l (* 2 ld)))
                  (= (lambdan l)
                     (* lambdan-maw (/ (leff min-active-width ldn)
                                       (leff l ldn))))
                  (= (gmn w/l id kn)
                     (sqrt (* 2 kn id w/l)))
                  (= (gmbn w/l id vsb)
                     (* (etan vsb) (gmn w/l id)))
                  (= (gdsn l id)
                     (* (lambdan l) id))
                  (= (cgsn w l)
                     (+ (* cgson w) (* 2/3 cox w (leff l ldn))))
                  (= (cgdn w)
                     (* cgdon w))))
      (else (error "Unknown message -- nmos-tran.")))))
```

PMOS Transistor

```
(define pmos-tran
  (lambda (m)
    (case m
      ((type) 'transistor)
      ((name) 'pmos-transistor)
      ((make) (lambda (name drain gate source bulk model w l)
                (let* ((ad (components:drain/source-area model w))
                       (as ad)
                       (pd (components:drain/source-perimeter model w))
                       (ps pd)
                       (nrd (components:drain/source-resistance-squares
                             model w))
                       (nrs nrd))
                  (list (glue 'm name) (glue '% drain)
                        (glue '% gate) (glue '% source)
                        (glue '% bulk) (glue 'PMOS model)
                        (glue 'w= w) (glue 'l= l)
                        (glue 'ad= ad 'p) (glue 'as= as 'p)
                        (glue 'pd= pd 'u) (glue 'ps= ps 'u)
                        (if (not (eq? nrd 'computed-from-rsdw))
                            (glue 'nrd= nrd " " 'nrs= nrs)
                            (glue " " " "))))))
      ((params) '(pmos-tran-w pmos-tran-l))
      ((area) '(* pmos-tran-w pmos-tran-l))
      ((design) '((= (vtp vsb)
                     (- vtop (* gammap (- (sqrt (- phip vsb))
                                          (sqrt phip)))))
                  (= (idp-sat sz vgs vsb kp)
                     (* 1/2 kp sz (expt (- vgs (vtp vsb)) 2)))
                  (= (etap vsb)
                     (/ gammap (* 2 (sqrt (+ phip vsb)))))
                  (= (leff l ld)
                     (- l (* 2 ld)))
                  (= (lambdap l)
                     (* lambdap-maw (/ (leff min-active-width ldp)
                                      (leff l ldp))))
                  (= (gmp w/l id kp)
                     (sqrt (* 2 kp id w/l)))
                  (= (gmbp w/l id vsb)
                     (* (etap vsb) (gmp w/l id)))
                  (= (gdsp l id)
                     (* (lambdap l) id))
                  (= (cgsp w l)
                     (+ (* cgsop w) (* 2/3 cox w (leff l ldp))))
                  (= (cgdp w)
                     (* cgdop w))))
      (else (error "Unknown message -- pmos-tran.")))))
```

A.4.2 Other Components

CMOS Capacitor

```
(define capacitor
  (lambda (m)
    (case m
      ((type) 'capacitor)
      ((name) 'cmos-capacitor)
      ((make) (lambda (name node1 node2 value)
                (list (glue 'c name)
                      (glue '% node1)
                      (glue '% node2)
                      (glue value))))
      ((area) '(/ value cox))
      (else (error "Unknown message -- capacitor.")))))
```

CMOS Resistor

```
(define resistor
  (lambda (m)
    (case m
      ((type) 'resistor)
      ((name) 'cmos-resistor)
      ((make) (lambda (name node1 node2 value)
                (list (glue 'r name)
                      (glue '% node1)
                      (glue '% node2)
                      (glue value))))
      ((area) '(* 2 (expt min-active-width 2)
                  (/ nrcp-rz r-square)))
      (else (error "Unknown message -- resistor.")))))
```

Voltage Source

```
(define voltage-source
  (lambda (m)
    (case m
      ((type) 'voltage-source)
      ((name) 'cmos-voltage-source)
      ((make) (lambda (name node1 node2 value)
                (case spice:current-analysis
                  ((dc) (list (glue 'v name)
                              (glue '% node1)
                              (glue '% node2)
                              'pwl `(0 0 ,t_dc_on ,value
                                         ,t_dc_end ,value)))
                  ((ac) (list (glue 'v name)
                              (glue '% node1)
                              (glue '% node2)
                              'dc
                              (glue value 'V)))
                  ((tran) (list (glue 'v name)
                                (glue '% node1)
                                (glue '% node2)
                                'pwl `(0 0 ,t_tran_on ,value
                                           ,t_tran_end ,value)))
                  (else (error "Unknown message -- voltage-source."))))))))
```

Wire

```
(define wire
  (lambda (m)
    (case m
      ((type) 'wire)
      ((name) 'cmos-wire)
      ((make) (lambda (name node1 node2)
                (list (glue 'v 'wire_ name)
                      (glue '% node1)
                      (glue '% node2)
                      'dc
                      (glue 0 'V))))
      (else (error "Unknown message -- wire.")))))
```

Appendix B

MOS Device Models

B.1 Device Model Constants

The large signal behavior of the transistors in the op amps in my system is modeled by a set of equations known as the Shichman-Hodges MOS device model. This device model uses a number of physical as well as fabrication-dependent technology constants which are summarized in Tables B.1 and B.2.

Symbol	Description	Value	Units
k	Boltzmann's constant	$1.3806226 \times 10^{-23}$	J/K
q	Elementary charge	$1.6021918 \times 10^{-19}$	$Coulomb$
μ_{ON}	NMOS Surface mobility	534×10^{-4}	m^2/Vs
μ_{OP}	PMOS Surface mobility	5185×10^{-4}	m^2/Vs
ϵ_{OX}	Permittivity of S_iO_2	$3.4531438 \times 10^{-11}$	F/m

Table B.1: Physical and Silicon Constants

Table B.1 provides a summary of the relevant physical and silicon constants. Table B.2 encapsulates the model constants for a typical $5\mu m$ Silicon-Gate Bulk CMOS p-Well chip manufacturing process used in my design experiments.

Symbol	Description	NMOS Value	PMOS Value	Units
L_{MAW}	Minimum active width	5		μm
t_{OX}	Oxide thickness	80		nm
K_F	Flicker noise coefficient	3×10^{-24}		–
V_{TON}, V_{TOP}	Threshold voltage ($v_{BS} = 0$)	1.0	-1.0	V
K_N, K_P	Transconductance parameter (in saturation)	17.0	8.0	$\mu A/V^2$
γ_N, γ_P	Bulk threshold parameter	1.3	0.6	$\sqrt{V}$
ϕ_N, ϕ_P	Surface potential at strong inversion	0.7	0.6	V
L_{DN}, L_{DP}	Lateral diffusion	0.8	0.8	μm
C_{GSON}, C_{GSOP}, C_{GDON}, C_{GDOP}	Gate-Source and Gate-Drain overlap capacitances	350×10^{-12}		F/m
$\lambda_N(L_{MAW})$, $\lambda_P(L_{MAW})$	Channel-length modulation parameter at L_{MAW}	0.0247	0.0494	V^{-1}
$R_\Box$	Sheet resistance of polysilicon	25		$\Omega/\Box$

Table B.2: Typical $5\mu m$ Silicon-Gate Bulk CMOS p-Well Process Constants

B.2 MOS Large-Signal Device Model

The MOS large-signal model shown in Figure B-1 is the starting point in determining the circuit level behavior of the MOS transistors, i.e. the functional relationship among the terminal voltages and currents.

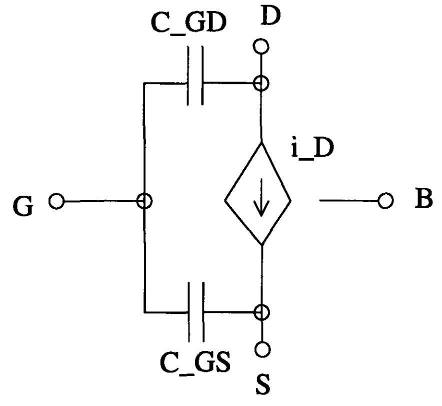


Figure B-1: Large-signal MOS transistor model

Depending on the value of the terminal voltages, MOS transistors can be in one of the three regions of operation: cut-off, non-saturation, or saturation. The desired goal of attaining high signal amplification in analog MOS amplifiers restricts most MOS transistors in these circuits to operate in the saturation region. The op amps designed by my system are no exception. We want all transistors in the op amps to be in their saturation region.

An NMOS transistor will be in the saturation region if

$$0 < (v_{GS} - V_{TN}) \leq v_{DS}$$

holds true. Similarly, for a PMOS transistor:

$$0 < (v_{SG} + V_{TP}) \leq v_{SD}$$

where V_{TN} and V_{TP} are the threshold voltages which depend on the source-bulk voltage v_{SB} through the body-effect equations:

$$V_{TN} = V_{TON} + \gamma_N \; (\sqrt{\phi_N + v_{SB}} - \sqrt{\phi_N})$$

$$V_{TP} = V_{TOP} - \gamma_P \; (\sqrt{\phi_P - v_{SB}} - \sqrt{\phi_P})$$

The boundary between the non-saturation region and the saturation region is defined by the saturation voltage v_{DS_SAT}:

$$v_{DS\_SAT} = v_{GS} - V_T$$

The current that flows through the MOS transistors in saturation is given by the following equations:

$$i_{DN} = \beta_N \; (v_{GS} - V_{TN})^2$$

$$i_{DP} = -\beta_P \; (v_{SG} + V_{TP})^2$$

for NMOS and PMOS respectively, where β_N and β_P are the transconductance parameters given by

$$\beta_N = K_N \; S$$

$$\beta_P = K_P \; S$$

The effective width and length of the channel are related to the actual geometric width and length of the transistor through

$$W_{eff} \approx W$$

$$L_{eff} = L - 2 \; L_D$$

where we have ignored the oxide encroachment of the source and drain areas as shown in Figure B-2.

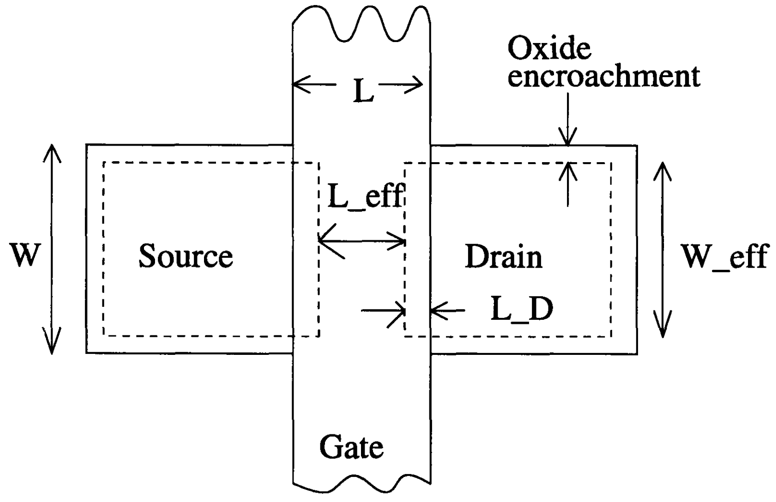


Figure B-2: Top view of a MOS transistor

Large Signal Capacitances

The following overlap capacitances play a role in modeling the MOS transistor behavior in the saturation region:

$$C_{GSN} = W_{eff}\, C_{GSON} + \frac{2}{3}\, C_{OX}\, W_{eff}\, L_{eff}$$

$$C_{GSP} = W_{eff}\, C_{GSOP} + \frac{2}{3}\, C_{OX}\, W_{eff}\, L_{eff}$$

$$C_{GDN} = W_{eff}\, C_{GDON}$$

$$C_{GDP} = W_{eff}\, C_{GDOP}$$

where C_{OX} is the capacitance per unit area of the gate oxide:

$$C_{OX} = \frac{\epsilon_{OX}}{t_{OX}}$$

B.3 Small-Signal MOS Model

To simplify the analysis of MOS circuits, we can linearize the device characteristics around the DC bias point determined from the large signal model. The justification for this simplification comes from the fact that most analog circuits perform their function over a limited excitation range. This linearization process results in the small-signal model of the MOS transistor shown in Figure B-3.

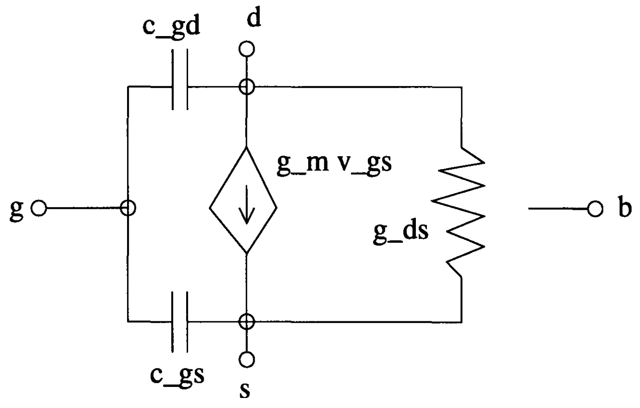


Figure B-3: Small-signal MOS transistor model

Under the assumption that the MOS transistor is in the saturation region, the small-signal channel transconductances g_m are given by

$$g_{mn} = \frac{\partial i_{DN}}{\partial v_{GS}} \approx \sqrt{2\ K_N\ S\ I_{DN}}$$

$$g_{mp} = \frac{\partial i_{DP}}{\partial v_{GS}} \approx \sqrt{2\ K_P\ S\ |I_{DP}|}$$

evaluated at the DC bias point. To compute the small signal channel conductances g_{ds} we need to modify the large-signal current equations as follows:

$$i_{DN} = \beta_N\ (v_{GS} - V_{TN})^2\ (1 + \lambda_N\ v_{DS})$$

$$i_{DP} = -\beta_P\ (v_{SG} + V_{TP})^2\ (1 - \lambda_P\ v_{DS})$$

for NMOS and PMOS respectively, where the $(1 + \lambda\ v_{DS})$ factors are used to model the channel-length modulation effects.

The small-signal channel conductances are given by:

$$g_{dsn} = \frac{\partial i_{DN}}{\partial v_{DS}} \approx \lambda_N \, I_{DN}$$

$$g_{dsp} = \frac{\partial i_{DP}}{\partial v_{DS}} \approx \lambda_P \, |I_{DP}|$$

The channel-length modulation factors λ are functions of the channel length:

$$\lambda_N(L) = \lambda_N(L_{MAW}) \, \frac{L_{eff}(L_{MAW})}{L_{eff}(L)}$$

$$\lambda_P(L) = \lambda_P(L_{MAW}) \, \frac{L_{eff}(L_{MAW})}{L_{eff}(L)}$$

The small signal capacitances $c_{gsn}, c_{gsp}, c_{gdn}$, and c_{gdp} are assumed to be the same as their large signal counterparts, $C_{GSN}, C_{GSP}, C_{GDN}$, and C_{GDP}.

B.4 Design Defaults

A common practice in VLSI circuit design is to fix the channel length L of all transistors in the op amp to a large enough value that (1) will keep the channel modulation parameter λ stable, and (2) will permit good matching for the differential pair and the current mirror under process variations. In my op amp design procedure I set

$$L = L_{factor} \, L_{MAW}$$

where the default value of L_{factor} is 2.

To simplify the op amp analysis, I ignore all temperature dependencies in the model.

The design defaults are summarized in Table B.3.

Symbol	Description	Value	Units
L	MOS Channel length	10	μm
T	Ambient temperature	298.15	K
V_{DD}	Positive power supply voltage	5	V
V_{SS}	Negative power supply voltage	-5	V
C_L	Loading capacitance	20	pF

Table B.3: Design Defaults

Appendix C

Op Amp Analysis Summary

C.1 Performance Functions

Active Area

The exact area occupied by an op amp on a chip can be determined only after all circuit elements and interconnecting paths have been laid out by using a specialized CAD tool. However, we can provide a good relative measure of the op amp area by ignoring the inter-element spacing and the routing paths, summing only the active areas taken by the actual transistors, the compensation capacitor, and the nulling resistor:

$$Active\ area = \sum_i W_i L_i + Area(C_C) + Area(R_Z)$$

$$Area(R_Z) \approx \frac{2(L_{MAW})^2}{R_Z/R_\square}$$

$$Area(C_C) \approx \frac{C_C}{C_{OX}}$$

where W_i and L_i are the channel width and length of the i-th transistor in the op amp. The physical layout of the resistor is assumed to be as shown in Figure C-1.

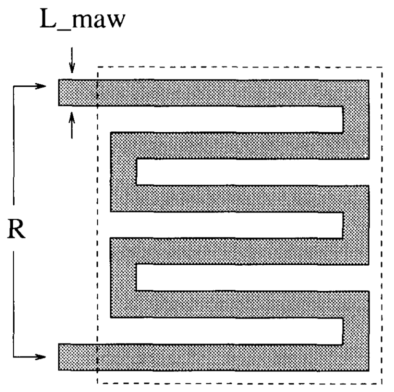


Figure C-1: Top view of CMOS resistor

DC Power Dissipation

The total DC power dissipation of the op amps can be computed by adding the contributions of each power supply rail:

$$DC\ Power = (V_{DD} + |V_{SS}|)\ (I_{(CS)} + I_{(OS)})$$

where $I_{(CS)}$ and $I_{(OS)}$ are the quiescent (DC) currents flowing through the current sink and the output stage respectively.

Common-Mode Input Voltage Range

The common-mode input voltage limits specify the range of common-mode input values over which the op amp continues to sense and amplify the differential-mode signal. These limits can be estimated by the common-mode range over which all transistors in the circuit stay in their high gain operating region (saturation):

$$Input\ CMR^{(+)} = \max(V_+ = V_-)$$

$$Input\ CMR^{(-)} = \min(V_+ = V_-)$$

and all transistors in saturation.

Output Voltage Swings

The output voltage swings are the maximum and minimum values of the output signal for which the op amp continues to amplify the differential-mode signal. The output voltage swing limits can be found by considering the saturation limits for the output-stage transistors:

$$Output\ swing^{(+)} = \max(V_{OUT})$$

$$Output\ swing^{(-)} = \min(V_{OUT})$$

and all transistors in saturation.

Differential-Mode Voltage Gain

The differential-mode gain is defined as the ratio of the amplitudes of the output signal and the differential-mode signal:

$$Differential\ mode\ gain = \left|\frac{v_{OUT}}{v_D}\right| = \left|\frac{v_{OUT}}{v_+ - v_-}\right|$$

The differential-mode gain at any frequency ω can be determined by taking the magnitude of the transfer function:

$$Differential\ mode\ gain\ (\omega) = \|T(s = \jmath\omega)\|$$

For example,

$$Differential\ mode\ gain\ DC = \|T(s = \jmath 0)\|$$

The transfer function $T(s)$ is a function of the stage transconductances g_{mI} and g_{mII}, stage resistances R_I and R_{II}, stage loading capacitances C_I and C_{II} and, if present, compensation components C_C and R_Z. The algebraic form of the transfer function varies depending on the style of the compensation module within the op amp (see the compensation module object equations). The transfer function for each of the three compensation cases is derived using the small-signal op amp equivalent circuit given in Figure C-2.

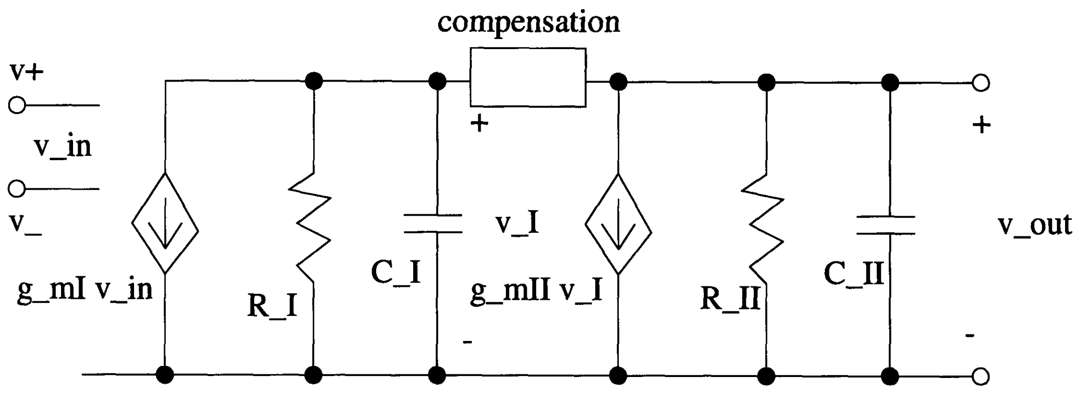


Figure C-2: Op amp small-signal equivalent circuit

Poles and Zeros

In the cases of Miller or nulling resistor compensations, the pole and zero locations were computed by an approximate method assuming widely spaced dominant and output poles. The details of this method can be found in [2].

Systematic Input Offset

Ideally, an op amp would produce a zero volt (mid-power rail) output when both inputs are held at an equal voltage (i.e. when the differential-mode signal is zero). In reality, one of the inputs will have to be offset by a small amount in order for the op amp to achieve a zero volt output. This voltage offset is composed of a systematic input offset voltage (resulting from design) and random input offset (resulting from device mismatches due to process variations).

In the case of nil output stage, the systematic input offset is simply the ratio of the deviation ΔV_{OUT} of the output DC voltage from the zero level and the first stage gain (with a zero differential-mode input signal):

$$Systematic\ input\ offset \approx \frac{\Delta V_{OUT}}{Gain_I}$$

where $Gain_I = g_{mI}\ R_I$.

In the cases of a simple or cascaded output stage, my system first recomputes the current through the common-source transistor I_{OS} and the current-sink transistor I_{OS*}. These two currents must be equal. If they are not, we can determine by how much the gate-source voltage $V_{GS(OS1)}$ of the output stage common-source transistor needs to be adjusted to achieve the equality:

$$\Delta V_{GS(OS1)} = \frac{\sqrt{I_{(OS*)}} - \sqrt{I_{(OS)}}}{\sqrt{\frac{1}{2}\, K_P\, S_{(OS1)}}}$$

where $S_{(OS1)}$ is the size of the common-source device of the output stage (`sos-sz1` or `cos-sz1`).

The systematic input offset is now determined by the ratio of the $\Delta V_{GS(OS1)}$ and the DC differential-mode gain of the first stage:

$$Systematic\ input\ offset \approx \frac{\Delta V_{GS(OS1)}}{Gain_I}$$

Output Resistance

The output resistance of an op amps is simply the output resistance of the output-stage:

$$R_{OUT} = R_{II}$$

Slew Rate

The slew rate is the maximum attainable rate of change of the output voltage. In non-nil compensation cases, it is given approximately by:

$$Slew\ rate \approx \frac{I_{(CS)}}{C_C}$$

In nil compensation cases, the slew rate is approximately the ratio of the bias current through the output stage and its loading capacitance:

$$Slew\ rate \approx \frac{I_{(OS)}}{C_{II}}$$

for a non-nil output stage and:

$$Slew\ rate \approx \frac{I_{(CS)}}{C_I}$$

for a nil output stage.

Common-Mode Rejection Ratio (CMRR)

The CMRR is defined as the ratio of the differential-mode gain and the common-mode gain:

$$CMRR = \frac{Differential\ mode\ gain}{Common\ mode\ gain}$$

The common-mode gain at DC is approximately given by:

$$Common\ mode\ gain\ DC \approx \frac{g_{m(DS)}\ R_{out(CM)}}{1 + 2\ g_{m(DS)} R_{out(CS)}}$$

The CMRR at DC is thus given by:

$$CMRR\ DC = \frac{Differential\ mode\ gain\ DC}{Common\ mode\ gain\ DC}$$

Unity-Gain Bandwidth (UGB)

The unity-gain bandwidth ω_{UGB} (UGB) is the frequency at which the differential-mode gain of the op amp is equal to unity (0dB). In cases of non-nil output stages and non-nil compensation it can be approximately determined by the ratio:

$$\omega_{UGB} \approx \frac{g_{mI}}{C_C}$$

In the case of a non-nil output stage and nil compensation, the UGB is approximately:

$$\omega_{UGB} \approx Gain_I\ Gain_{II}\ Pole_{II}$$

Finally, in cases of a nil output stage:

$$\omega_{UGB} \approx \frac{g_{mI}}{C_I}$$

We can obtain a more accurate estimate of ω_{UGB} by solving the non-linear equation:

$$||T(\jmath\omega)|| = 1$$

where we can use the approximate value of ω_{UGB} as a good initial guess for the numerical method.

Phase Margin

The phase of the transfer function at the unity-gain frequency is an excellent measure of the stability of the op amp, and is called phase margin.

The phase margin is approximately calculated from equations derived from considering the transfer function and the approximate equations for the unity gain bandwidth. The algebraic form of these equations differs depending on the styles of the compensation and output stage in the op amp (see the compensation module object equations).

We can recalculate the phase margin with more accuracy by evaluating:

$$Phase\ margin = \angle - T(\jmath\ \omega_{UGB})$$

A phase margin greater than 60 degrees is desirable for stable negative feedback around the op amp.

A common approach to achieve stability is by compensation, and can be achieved in the op amps by either the Miller, the nulling resistor compensation techniques, or self-compensation in cases of nil output stages (as long as the capacitive load at the output is large enough). The goal of the compensation is to move all poles and zeros, except for the dominant pole, sufficiently beyond the unity-gain frequency ω_{UGB}.

Power-supply rejection ratios (PSRR)

The PSRR is defined as the ratio of the differential-mode gain of the op amp to the ratio of the induced change in the output voltage and the change in the supply voltage:

$$PSRR = \frac{Differential\ mode\ gain}{\Delta v_{out}/\Delta v_{supply}}$$

The $PSRR^{(+)}$ and $PSRR^{(-)}$ (rejection ratios of the positive and negative supplies) at DC are given by:

$$PSRR_{DC}^{(+)} = Gain_I \, \frac{g_{mII}}{G_I \; g_{ds(OS1)}}$$

$$PSRR_{DC}^{(-)} = Gain_I \, \frac{g_{mII}}{G_I \; g_{ds(OS2)}}$$

where $g_{ds(OS1)}$ and $g_{ds(OS2)}$ are the channel conductances of the common-source and the current-sink devices respectively.

Bibliography

[1] H. Abelson and G. J. Sussman, *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, Massachusetts, 1985.

[2] Phillip E. Allen and Douglas R. Holberg, *CMOS Analog Circuit Design*, Harcourt Brace Jovanovich, Saunders College Publishing, Orlando, Florida, 1987.

[3] Robert K. Brayton, Gary D. Hachtel and Alberto L. Sangiovanni-Vincentelli, *A Survey of Optimization techniques for Integrated-Circuit Design*, in PROC. IEEE, vol. 69, pp. 1334-1362, Oct. 1981.

[4] Robert K. Brayton and Robert Spence, *Sensitivity and Optimization*, Elsevier Scientific Publishing Company, New York, 1980.

[5] Richard P. Brent, *Algorithms for Minimization without Derivatives*, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

[6] Yuval Davidor, *Genetic Algorithms and Robotics*, World Scientific, Singapore, 1991.

[7] M. DeGrauwe et al, *IDAC: An Interactive Design Tool for Analog CMOS Circuits*, in IEEE J. SOLID-STATE CIRCUITS, vol. SC-22, No. 6, December 1987.

[8] Randall L. Geiger, Phillip E. Allen, Noel R. Strader, *VLSI Design Techniques for Analog and Digital Circuits*, McGraw-Hill, New York, 1990.

[9] Lance A. Glasser and Daniel W. Dobberpuhl, *The Design and Analysis of VLSI Circuits*, Addison-Wesley, Reading, Massachusetts, 1985.

[10] David E. Goldberg, *Genetic Algorithms*, Addison-Wesley, Reading, Massachusetts, 1989.

[11] Paul R. Gray and Robert G. Meyer, *MOS Operational Amplifier Design - A Tutorial Overview*, in IEEE J. SOLID-STATE CIRCUITS, vol. SC-17, pp. 969-982, Dec. 1982.

[12] Chris Hanson, *MIT Scheme Reference Manual*, Technical Report 1281, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts, 1991.

[13] R. Harjani et al, *Analog Circuit Synthesis and Exploration in OASYS*, in PROC. 1988 IEEE INT'L CONF. ON COMPUTER DESIGN (ICCD), Oct. 1988.

[14] Mohammed Ismail and Jose Franca (editors), *Introduction to Analog VLSI Design Automation*, Kluwer Academic Publishers, Norwell, Massachusetts, 1990.

[15] Han Y. Koh, *Design Synthesis of Monolithic CMOS Operational Amplifiers*, Report No. UCB/CSD 89/507, Computer Science Division, University of California, Berkeley, California, 1989.

[16] Michael R. Lightner and Stephen W. Director, *Multiple Criterion Optimization for the Design of Electronic Circuits*, in IEEE TRANS. CIRCUIT SYST.,vol. CAS-28, March 1981.

[17] Prabir C. Maulik and L. Richard Carley, *Automating Analog Circuit Design using Constrained Optimization Techniques*, IEEE, 1991.

[18] Ognen J. Nastov, *Nominal Design Optimization of VLSI Circuits*, 6.892 Project Report, MIT Dept. of Elec. Eng. and Comp. Sci., Cambridge, Massachusetts, 1992.

[19] Ognen J. Nastov, *Optimal Design of a Two-Stage CMOS Operational Amplifier*, 2.998 Project Report, MIT Dept. of Elec. Eng. and Comp. Sci., Cambridge, Massachusetts, 1993.

[20] Bill Nye et al, *DELIGHT.SPICE: An Optimization-Based System for the Design of Integrated Circuits*, Dept. EECS, University of California, Berkeley, California, 1983.

[21] Alan Parkinson et al, *OptdesX – A Software System for Optimal Engineering Design Users Manual, release 1.0*, Design Synthesis Inc., Provo, Utah, 1992.

[22] Colin R. Reeves, *Using Genetic Algorithms with Small Populations*, in PROC. OF THE FIFTH INT'L CONF. ON GENETIC ALGORITHMS, Morgan Kaufmann Publishers, San Mateo, California, 1993.

[23] Andrew L. Ressler, *A Circuit Grammar for Operational Amplifier Design*, PhD Thesis, Dept. of Elec. Eng. and Comp. Sci., Massachusetts Institute of Technology, 1984.

[24] J. Richardson, M. Palmer, G. Liepins, and M. Hilliard, *Some Guidelines for Genetic Algorithms with Penalty Functions*, in PROC. OF THE THIRD INT'L CONF. ON GENETIC ALGORITHMS, Morgan Kaufmann Publishers, San Mateo, California, 1989.

[25] M. Schoenauer and S. Xanthakis, *Constrained GA Optimization*, in PROC. OF THE FIFTH INT'L CONF. ON GENETIC ALGORITHMS, Morgan Kaufmann Publishers, San Mateo, California, 1993.

[26] William H. Press et al, *Numerical Recipes in C*, Cambridge University Press, Cambridge, UK, 1988.

[27] Neil H. E. Weste and Kamran Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, Addison-Wesley, Reading, Massachusetts, 1988.