Automated Device Sizing to Optimize the Performance of CMOS Operational Amplifiers

Brian Downing

Northern Arizona University

March 28, 2005

■Table of Contents

■Introduction

Analog design is often a major bottleneck in the production of integrated circuits. The ability to overcome this bottleneck can have a major impact on the commercial success of a company. The first company whose product is designed into a socket almost always wins the majority of the business and sets future standards. The competition can subsequently win market share only by offering improved performance or lower cost.

An important component of the analog design bottleneck is the fact that device sizing is normally done by hand. This process is tedious and time consuming. If device sizing could be automated, this would decrease the time to market. Automated device sizing may also lead to improved performance, reduced die size, and lower costs.

Although many attempts have been made to automate the process of device sizing, no solution has yet been widely accepted by industry. On the contrary: when experienced analog designers evaluate the algorithms offered by leading CAD vendors, they often discover basic design errors such as transistors not being biased in the saturation region.

The strategic goal of this project was to design and implement an algorithm to automate device sizing during analog design for CMOS operational amplifiers (op-amps). The organization of this report is as follows. The first section is a review of prior work in automated device sizing for analog circuits. The second section discusses the approach and implementation aspects of this work. The third section presents the results of this work. The fourth section suggests further work that could be done in this area. The fifth section presents the conclusions of the project. The remainder of the report contains acknowledgments, references, and appendixes.

■Review of Prior Work

There has been a surge of interest in automated analog design during the past few years [1]. A variety of different approaches can be found in literature [2-4]. The different approaches can be categorized into three categories: 1) Layout-based, 2) Knowledge-based, and 3) Optimization-based [1].

The layout approach is similar to digital synthesis in that it assembles predefined layouts. This method does not provide enough flexibility to be usable in industry [1].

Knowledge-based methods model the techniques of analog designers. Although this method is very fast, setting up the problem is very time consuming and requires an experienced analog circuit designer.

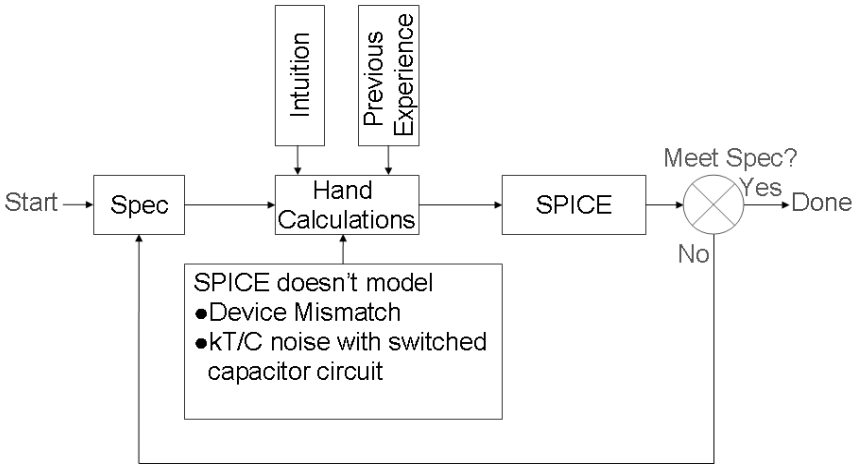
Optimization based methods take a “black box” approach to the problem. An objective function is defined, and independent variables are adjusted to maximize or minimize the objective function. Optimization-based methods can be divided into two sub-categories: a) circuit simulation based optimization and b) analytical equation based optimization [1]. The simulation based method uses a circuit simulator such as SPICE

within the optimization loop to determine circuit performance. The results are very accurate but the approach tends to be slow. The use of simplified analytical equation based models trades off a loss of accuracy for an improvement in speed.

■This Work

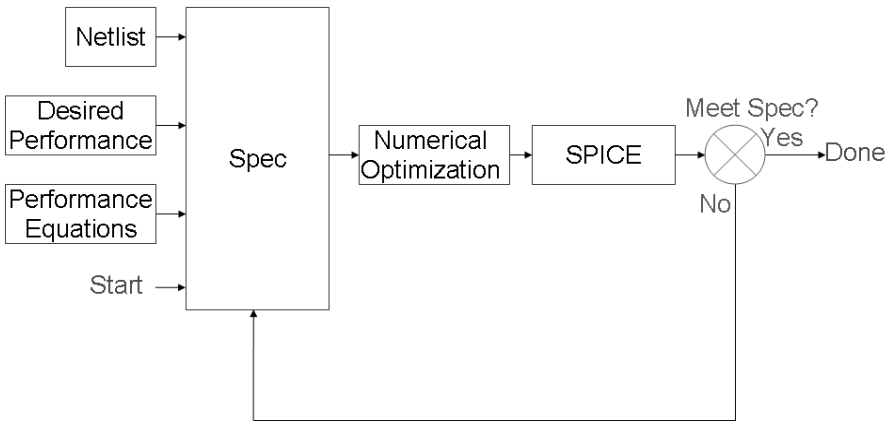
● General Approach

The algorithm that is developed and implemented in this work is knowledge-based and uses analytical equations to describe device behavior. It is modeled after the way seasoned analog circuit designers design circuits. The flow chart below depicts how circuits are typically designed:



This is an iterative process that involves hand calculations, SPICE circuit simulations, and the assessment of solutions. The most challenging part is performing the hand calculations. Another challenging part is evaluating the trade-offs between different specifications. Since optimization software is used in this work, the balance between the different specifications must be defined a priori. It is then taken care of within the software. Another challenging part of analog design is accounting for phenomena that SPICE does not model. For example SPICE does not in its native form model device mismatch and it also does not model kT/C noise in switched capacitor circuits. This work does not consider phenomena that are not modeled in SPICE.

A high level flow chart of the algorithm used in this work is shown below:



The key difference is that the hand calculations are replaced with numerical optimization software. In order to make this replacement the equations for the performance have to be input into the software. Since hand calculations are part of the design process this does not represent additional work for the designer. In fact it saves time since the designer does not have to make approximations and then verify that the approximations are valid. When circuits are designed the equations must be greatly simplified to be useful to the designer. Making these approximations is a tedious job and typically costs ~20% accuracy when compared to detailed SPICE models. Since all the calculations are performed by software this eliminates the approximation errors.

There is typically a wide range of device sizes that will meet a given specification. The use of numerical optimization software allows the computer to account for all the trade-offs to determine a better solution than a designer is likely to achieve using intuition to guess where the optimal solution is. Typically in industry the designer will stop once a solution that meets all specifications is identified. This is mainly because of time pressures as companies are trying to beat their competitors into the market. A secondary reason is that because of the non-linear relationships and competing objective functions it is a very difficult process to solve for a more optimal solution.

In order to manufacture a circuit the layout must be on grid. For this work the grid is ignored. Others have solved this problem by either snapping to a grid after the optimization is complete or by programming the problem as an integer problem where the integer is set by the grid spacing.

•Implementation

The project was implemented in three phases of escalating complexity. The first phase focuses on an NMOS common source amplifier with a PMOS active load. For SPICE circuit simulations a level 1 model is used. This allows the optimization software to use the same equations as the SPICE circuit simulator.

The second phase is the same common source amplifier, but using a BSIM3 SPICE model. First order hand calculations are used. To fit the first order equations to the SPICE circuit simulations a technique termed 'making coefficients of equations' is used. This is discussed further later.

Phase 3 is a two-stage operational-amplifier with a BSIM3 SPICE model. The first stage is an NMOS differential amplifier and the second stage is a PMOS common source amplifier.

All the software was selected so the project could be completed on a PC platform. The top level programming language is Matlab [5]. Mathematica [6] is used to derive equations and for small signal analysis. All SPICE simulations are run with TopSPICE [7]. The optimization part is written in AMPL [8] which is a high level optimization modeling language. AMPL requires a solver for the optimization problem. The decision to use AMPL was made so that different solvers could be used without reformulating the optimization problem.

For this project multiple nonlinear AMPL solvers are used. The problem was formatted in its native form meaning that the width and length of the transistors are in units of microns. Due to this poor scaling the optimization solvers sometimes give solutions that do not meet the constraints. To avoid this problem the solutions are verified against the constraints and solutions that do not meet the constraints are rejected. The first solver is DONLP2 [9] which uses a variant of the SQP-method (sequential quadratic programming). The second solver is MINOS [10] which uses a projected Lagrangian approach. The third solver is Knitro [11] which allows the user to select between the interior point method and the active set algorithm. The fourth solver is SNOPT [12] which uses the SQP method. The fifth solver is LOQO [13] which uses an infeasible primal-dual interior-point method. The sixth solver is IPOPT [14] which uses the interior point method.

One important specification of a circuit is that its DC voltages are in a range where all MOSFET's remain in saturation. For this project a technique is used where the bias voltages and lengths are fixed and the widths are varied to give the desired DC voltages. For this paper this method is called 'adjust widths for DC'. More information can be found in appendix B.

In order to make the project independent of the SPICE model, first order model equations are used and the coefficients are extracted from the SPICE simulations. The following describes how the coefficients of equations are extracted for the drain current.

First Order Drain Current Equation:

$$Id = \frac{\mu\, Cox}{2} \frac{W}{L - 2\, Ld} (Vgs - Vt)^2 (1 + \lambda\, Vds)$$

Coefficient Equation for Drain Current:

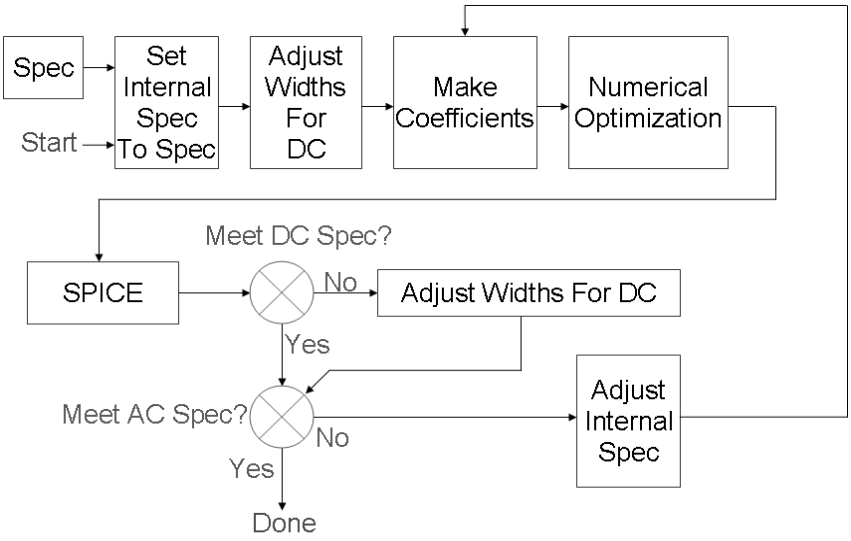
$$Id = Current\ Factor * \left(\frac{W}{L - 2\,Ld} (Vgs - Vt)^2 \right)$$

To calculate the <Current Factor> a SPICE simulation is run and the SPICE data is plugged into the equation. Since this is only an approximation the coefficients are recalculated when a transistor size or bias voltage changes.

The remaining equations and their associated coefficients of equations are defined Appendix A.

The TSMC 0.35µ BSIM3 SPICE models were used during phase 2 and phase 3. These models are "binned", which means the model is different for different device sizes. The software automatically takes care of the binning by changing the model that each transistor calls.

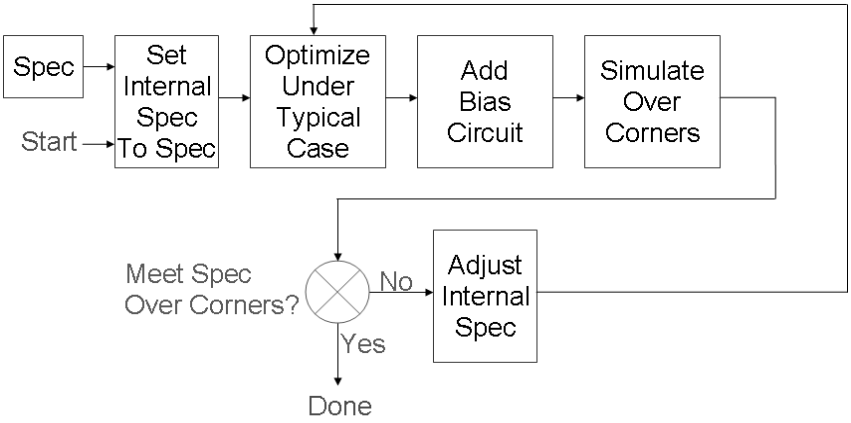
The following diagram shows the optimization procedure under typical conditions:



The first step is to set the internal specification to the user defined specification. This is done to account for the approximations used. For example if the specification for DC gain is 50 V/V and when the optimization is complete the DC gain is 45 V/V then the internal gain is increased for the next iteration. The loop is repeated until either all the specs are met or the max iteration count is reached. The next step is to adjust the widths for DC. This makes sure all the transistors are at their desired DC values for the next steps. Then the coefficients of equations are calculated. This fits the more complex MOSFET equations to the level 1 equations. Now the numerical optimization problem is formulated and executed with AMPL. Multiple solvers for AMPL are run and the optimal solution that meets the constraints is taken. Next SPICE is run with the values returned from the optimization software. If the circuit does not meet the DC performance spec then the widths are adjusted for DC. Now

the AC specs are checked. If the specifications are met then the loop is exited. If the AC specifications are not meet then the internal specifications are adjusted and the loop is repeated as long as the max iteration count is not reached.

The following diagram shows the complete algorithm:



As is done in industry, the circuit is designed for typical conditions, then the bias circuit is designed, and then the circuit is simulated over process, temperature range, and power supply variations. If any of the corner simulations fail then the worst failure case is used to adjust the internal spec for the next iteration. A maximum number of iterations is always specified so that the process does not repeat forever. To design the bias circuit the user specifies the device size of the bias network as a function of the other device sizes in the circuit. For example, with a simple current mirror the user would tell the program to make the current mirror transistor the same size as the load transistor in the amplifier. The program then sweeps the DC reference current until the desired output voltage is reached.

■Results

●Common Source Amplifier with Level 1 SPICE Model

The common source amplifier is optimized with the constraints unity gain frequency, DC gain, and input referred thermal noise. Below are the equations for these 3 constraints.

Unity gain frequency for a common source amp being driven at the gate of the NMOS in Hertz:

$$f_u = \frac{1}{2\pi}\sqrt{\frac{Gm,n^2 - Gdo^2}{Co\,(2\,Cf + Co)}}$$

DC gain:

$$A_{V,DC} = \frac{Gm,n}{Gdo}$$

Input referred thermal noise:

$$V_{NOISE,Input} = \sqrt{4\,k\,T\left(\frac{2}{3\,Gm,n} + \frac{2}{3}\frac{Gm,p}{Gm,n^2}\right)} \quad \text{[15 p.226]}$$

Where:

$Cf = CgdOvl,n$

$Co = Cload + Cdb,n + Cdb,p + CgdOvl,p$

$Gdo = Gds,n + Gds,p$

Gm,n is NMOS transconductance
Gm,p is PMOS transconductance
Gds,n is NMOS output conductance
Gds,p is PMOS output conductance
CgdOvl,n is the NMOS gate to drain overlap capacitance
CgdOvl,p is the PMOS gate to drain overlap capacitance
Cdb,n is the NMOS drain to bulk capacitance
Cdb,p is the PMOS drain to bulk capacitance
Cload is the load capacitance

For the common source amplifier the original spec is DC gain and the goal of the program is to minimize gate area. One unexpected feature is that if the input DC voltage of the NMOS transistor is fixed and the output voltage is fixed then the DC gain is fixed. This is not the case when a higher order MOSFET is used in SPICE. Next bandwidth and thermal noise are added. The program performed as expected with these constraints. When the program is given specifications that are easy to meet (e.g. low bandwidth and high thermal noise) then the software returns the minimum size devices as expected. As the specifications are made more difficult to achieve (increased bandwidth and decreased noise) the device sizes increase in an anticipated fashion.

At this point in the project the only optimization solver used is DONLP2. Most of the subsequent problems that were encountered were traced to DONLP2. The first major problem is that if DONLP2 is given a specification that cannot be meet then it will put a high value in a variable it calls infeasibility but will return a set of lengths and widths as if it were able to solve the problem. Once this problem was discovered it was simple to account for, but understanding the problem took a great amount of time. The second major problem was that DONLP2 was not returning the expected values. The first time this was encountered the issue turned out to be that the default resolution in DONLP2 was too large for the application. Once the tolerance was adjusted the software performed as expected. The second time the problem of DONLP2 returning an unexpected value, it turned out to be that DONLP2 was getting stuck in a local minimum instead of finding the global minimum. This is a limit of the optimization software that can only be corrected by switching to a program that has global convergence.

Results of a test case with the following constraints will be presented. The goal of the test case is to minimize device area from different initial conditions, subject to (s.t.) the following conditions:

$$s.t.\ f_u \geq 1\ Hz$$
$$s.t.\ V_{Noise,Input} \leq 1\ V$$
$$s.t.\ V_{Out,DC} \leq Vpbias + MpVth$$
$$s.t.\ MnId = MpId$$
$$s.t.\ Wp \geq 5\mu$$
$$s.t.\ Lp \geq 5\mu$$
$$s.t.\ Wn \geq 5\mu$$
$$s.t.\ Ln \geq 5\mu$$

Where:

MpVth = PMOS threshold voltage
MnId = NMOS drain current
MpId = PMOS drain current

The following tables show the results that were obtained:

Case 1.1

	Start	End
Wp	5u	5u
Lp	5u	5u
Wn	5u	5u
Ln	5u	5u
Vpbias	3	2.00614

Case 1.2

	Start	End
Wp	5u	5u
Lp	5u	5u
Wn	5u	5u
Ln	5u	5u
Vpbias	4	2.00617

Case 1.3

	Start	End
Wp	5u	5u
Lp	5u	5u
Wn	5u	5u
Ln	5u	5u
Vpbias	2	2.00617

Case 1.4

	Start	End
Wp	100u	5u
Lp	10u	5u
Wn	100u	5u
Ln	10u	5u
Vpbias	3	2.00612

Case 1.5

	Start	End
Wp	100u	5u
Lp	10u	5u
Wn	100u	5u
Ln	10u	5u
Vpbias	2	2.00614

Case 1.6

	Start	End
Wp	50u	5u
Lp	30u	5u
Wn	50u	5u
Ln	30u	5u
Vpbias	3	2.00617

Case 1.7

	Start	End
Wp	50u	5u
Lp	30u	5u
Wn	50u	5u
Ln	30u	5u
Vpbias	2	2.00612

Case 1.8

	Start	End
Wp	10u	5u
Lp	8u	5u
Wn	50u	5u
Ln	30u	5u
Vpbias	3	2.00613

Case 1.9

	Start	End
Wp	10u	5u
Lp	8u	5u
Wn	50u	5u
Ln	30u	5u
Vpbias	2	2.00612

	Case 1.10		Case 1.11		Case 1.12	
	Start	End	Start	End	Start	End
Wp	10u	5u	10u	5u	10u	5.01408u
Lp	8u	5u	8u	5u	8u	7.10616u
Wn	50u	5u	50u	5u	50u	5u
Ln	30u	5u	30u	5u	30u	5u
Vpbias	2.5	2.00615	4	2.00616	1.5	1.5999

Note that all cases except Case 12 found the minimal device sizes, which is what is expected. For case 12 the PMOS transistor starts in the triode region and finishes on the boundary between saturation and triode. Since the equations assume saturation, the minimal device sizes are not found. Since the power supply is 5 volts the choice of Vpbias = 1.5 volt is a very poor starting point.

•Common Source Amplifier with BSIM3 SPICE Model

The second phase of the project treated the common source amplifier with a BSIM3 SPICE model. There are issues running TopSPICE with the TSMC 0.35µ models. Since Northern Arizona University has a non-disclosure agreement concerning the TSMC models the author was not able to send TopSPICE a test case to solve this problem. The decision was therefore made not to vary the models in this project. The code is written so that the models can be varied but the models are always set to typical models. Temperature also causes problems with TopSPICE and the TSMC 0.35µ models. The TSMC models are written for HSPICE and TopSPICE claims to be compatible with HSPICE. However TopSPICE does not implement some of the HSPICE temperature model cards. This was found to cause some very strange results over temperature. The decision was therefore made not to vary temperature in the project. As a result, the corner simulations consist of only varying the power supply.

For the bias circuit it was originally planned that the input bias current would be specified and the program would adjust the bias transistor to get the correct bias voltage. Since the program is optimizing gate area the circuits have large Vgs values. In order to get a large Vgs from a diode connected transistor the width has to be small. In practice it was found that the widths would have to be smaller than the minimum allowed width. Due to transistor sizes leaving their allowable range the design bias network method was changed. Since typically in industry the circuit designer is given the bias current as a specification this project fails to develop a practical bias circuit. But since designing the bias network is a small part of designing an op-amp the program can be used to design everything except the bias network in practical applications.

The following shows the results for the common source amplifier with a BSIM3 SPICE model. The left column shows the optimization variables, AC constraints, DC node voltage constraints, and run time. There are DC constraints on Vpbias to keep the PMOS in saturation. Each test case is labeled with Case X and shows the start and end values for the optimization variables and constraints mentioned above.

The following shows constraints that are not shown in tables:

$s.t.\ V_{Out,DC} \le Vpbias + MpVth$

$s.t.\ Vdd - Vpbias \ge MpVth$

$s.t.\ MnId = MpId$

$s.t.\ MpW \ge 0.4\mu$

$s.t.\ MpL \ge 0.35\mu$

$s.t.\ MnW \ge 0.4\mu$

$s.t.\ MnL \ge 0.35\mu$

Where

MpVth = PMOS threshold voltage
MnId = NMOS drain current
MpId = PMOS drain current
MpW = width of PMOS Mp
MpL = length of PMOS Mp
MnW = width of NMOS Mn
MnL = length of NMOS Mn
Vpbias = bias voltage for PMOS Mp
Vdd = power supply = 3.3 volts

Cases 1 through 18 are for constraints that are very easy to satisfy. They test the ability of the algorithm to converge from different initial points. Cases 19 through 26 reveals how the optimized designs behave when the constraints are made more difficult to meet.

	Case 2.1		Case 2.2		Case 2.3	
	Start	End	Start	End	Start	End
MpW	0.4u	0.401049u	0.4u	0.40292u	0.4u	0.400388u
MpL	0.35u	0.35u	0.35u	0.35u	0.35u	0.35u
MnW	0.4u	0.4u	0.4u	0.4u	0.4u	0.4u
MnL	0.35u	0.35u	0.35u	0.35u	0.35u	0.35u
Vpbias	1.5694	1.571	2.3	1.573	2	1.57
DcGain >= 1	20.1043	20.1216	79.3099m	20.1122	429.458m	20.1108
unityGain >= 1	13.8855M	13.8831M	0	13.8842M	0	13.8845M
thermalNoise <= 1	13.366n	13.3706n	73.8005n	13.3747n	29.046n	13.3677n
Vdc_out = 1.65 +/- 5m	1.64988	1.64652	0.034707	1.648	0.132794	1.64857
Run Time	15.4233 min		8.36568 min		14.4713 min	

The PMOS width is not the minimum width of 0.4μ as would be expected with constraints that are easy to satisfy (small DC gain, small unity gain frequency, and large thermal noise). This is because of the first order DC bias models. The optimization software returns a width of 0.4μ but because it does not meet the DC constraints, the program tweaks in the width of the PMOS transistor to get the desired DC voltage.

	Case 2.4		Case 2.5		Case 2.6	
	Start	End	Start	End	Start	End
MpW	0.4u	0.4u	10u	0.403741u	10u	0.403741u
MpL	0.35u	0.35u	5u	0.35u	5u	0.35u
MnW	0.4u	0.4u	10u	0.4u	10u	0.4u
MnL	0.35u	0.35u	5u	0.35u	5u	0.35u
Vpbias	1.5	1.569	1.4	1.574	2.3	1.574
DcGain >= 1	14.4128	20.0842	1.88677	20.114	64.0863m	20.114
unityGain >= 1	14.1749M	13.8878M	20.6734M	13.8838M	0	13.8838M
thermalNoise <= 1	13.0536n	13.364n	9.1928n	13.3772n	65.2011n	13.3772n
Vdc_out = 1.65 +/- 5m	2.16583	1.65342	2.91497	1.64753	0.028661	1.64753
Run Time	14.5413 min		12.3918 min		10.6275 min	

	Case 2.7		Case 2.8		Case 2.9	
	Start	End	Start	End	Start	End
MpW	10u	0.403741u	10u	0.403741u	10u	0.40292u
MpL	5u	0.35u	5u	0.35u	5u	0.35u
MnW	10u	0.4u	10u	0.4u	1u	0.4u
MnL	5u	0.35u	5u	0.35u	5u	0.35u
Vpbias	2	1.574	1.5	1.574	1.4	1.573
DcGain >= 1	391.273m	20.114	2.32032	20.114	110.48m	20.1122
unityGain >= 1	0	13.8838M	21.9865M	13.8838M	0	13.8842M
thermalNoise <= 1	23.0611n	13.3772n	9.34513n	13.3772n	30.8094n	13.3747n
Vdc_out = 1.65 +/- 5m	0.13062	1.64753	2.86465	1.64753	3.27142	1.648
Run Time	9.59413 min		12.1538 min		8.34668 min	

	Case 2.10		Case 2.11		Case 2.12	
	Start	End	Start	End	Start	End
MpW	10u	0.400388u	10u	0.403741u	10u	0.40292u
MpL	5u	0.35u	5u	0.35u	5u	0.35u
MnW	1u	0.4u	1u	0.4u	1u	0.4u
MnL	5u	0.35u	5u	0.35u	5u	0.35u
Vpbias	2.3	1.57	2	1.574	1.5	1.573
DcGain >= 1	953.214m	20.1108	226.154m	20.114	119.464m	20.1122
unityGain >= 1	0	13.8845M	0	13.8838M	0	13.8842M
thermalNoise <= 1	43.0872n	13.3677n	34.1892n	13.3772n	31.1062n	13.3747n
Vdc_out = 1.65 +/- 5m	3.14578	1.64857	3.24389	1.64753	3.26916	1.648
Run Time	14.5506 min		9.09157 min		9.56558 min	

	Case 2.13		Case 2.14		Case 2.15	
	Start	End	Start	End	Start	End
MpW	100u	0.403741u	100u	0.403741u	100u	0.403741u
MpL	5u	0.35u	5u	0.35u	5u	0.35u
MnW	100u	0.4u	100u	0.4u	100u	0.4u
MnL	5u	0.35u	5u	0.35u	5u	0.35u
Vpbias	1.4	1.574	2	1.574	1.5	1.574
DcGain >= 1	1.84893	20.114	421.762m	20.114	2.26213	20.114
unityGain >= 1	169.735M	13.8838M	0	13.8838M	181.205M	13.8838M
thermalNoise <= 1	2.90312n	13.3772n	7.07296n	13.3772n	2.94974n	13.3772n
Vdc_out = 1.65 +/- 5m	2.92177	1.64753	0.13834	1.64753	2.87298	1.64753
Run Time	12.1428 min		9.37948 min		12.4422 min	

	Case 2.16		Case 2.17		Case 2.18	
	Start	End	Start	End	Start	End
MpW	1u	0.403741u	1u	0.403741u	1u	0.403741u
MpL	5u	0.35u	5u	0.35u	5u	0.35u
MnW	10u	0.4u	10u	0.4u	10u	0.4u
MnL	5u	0.35u	5u	0.35u	5u	0.35u
Vpbias	1.4	1.574	2.3	1.574	2	1.574
DcGain >= 1	76.1636m	20.114	3.53582m	20.114	17.0158m	20.114
unityGain >= 1	0	13.8838M	0	13.8838M	0	13.8838M
thermalNoise <= 1	39.1671n	13.3772n	286.312n	13.3772n	99.6606n	13.3772n
Vdc_out = 1.65 +/- 5m	0.033637	1.64753	0.001687	1.64753	0.008	1.64753
Run Time	12.4297 min		11.0547 min		9.69527 min	

	Case 2.19	
	Start	End
MpW	0.4u	0.4u
MpL	0.35u	1.09785u
MnW	0.4u	0.4u
MnL	0.35u	0.794829u
Vpbias	1.5694	1.41
DcGain >= 50	20.1043	51.7458
unityGain >= 1	13.8855M	6.08698M
thermalNoise <= 1	13.366n	20.1831n
Vdc_out = 1.65 +/- 5m	1.64988	1.65066
Run Time	13.0606 min	

	Case 2.20	
	Start	End
MpW	0.4u	0.4u
MpL	0.35u	0.871419u
MnW	0.4u	0.4u
MnL	0.35u	0.658168u
Vpbias	1.5694	1.428
DcGain >= 40	20.1043	42.1438
unityGain >= 1	13.8855M	7.38023M
thermalNoise <= 1	13.366n	18.3467n
Vdc_out = 1.65 +/- 5m	1.64988	1.6513
Run Time	12.8914 min	

As expected, when the DC gain constraint is increased the length increases while the width stays at minimum geometry.

Case 2.21

	Start	End
MpW	0.4u	0.4u
MpL	0.35u	0.624045u
MnW	0.4u	0.4u
MnL	0.35u	0.527732u
Vpbias	1.5694	1.49
DcGain >= 30	20.1043	31.9443
unityGain >= 1	13.8855M	9.24083M
thermalNoise <= 1	13.366n	16.4646n
Vdc_out = 1.65 +/- 5m	1.64988	1.64997
Run Time	18.4432 min	

Case 2.22

	Start	End
MpW	0.4u	0.426358u
MpL	0.35u	0.35u
MnW	0.4u	1.26543u
MnL	0.35u	0.35u
Vpbias	1.5694	0.9413
DcGain >= 1	20.1043	16.5384
unityGain >= 30M	13.8855M	30.0289M
thermalNoise <= 1	13.366n	8.45258n
Vdc_out = 1.65 +/- 5m	1.64988	1.64912
Run Time	52.3952 min	

When only bandwidth is increased, the width increases and the length stays at minimum geometry. Also the Vpbias drops which means there is a larger Vgs for the PMOS transistor as would be expected for increased bandwidth.

Case 2.23

	Start	End
MpW	0.4u	0.819927u
MpL	0.35u	0.35u
MnW	0.4u	2.01771u
MnL	0.35u	0.35u
Vpbias	1.5694	0.9363
DcGain >= 1	20.1043	16.116
unityGain >= 50M	13.8855M	50.0026M
thermalNoise <= 1	13.366n	6.54209n
Vdc_out = 1.65 +/- 5m	1.64988	1.64933
Run Time	21.4201 min	

Case 2.24

	Start	End
MpW	0.4u	1.73129u
MpL	0.35u	0.35u
MnW	0.4u	3.92324u
MnL	0.35u	0.35u
Vpbias	1.5694	0.9569
DcGain >= 1	20.1043	17.5687
unityGain >= 100M	13.8855M	100.002M
thermalNoise <= 1	13.366n	4.63318n
Vdc_out = 1.65 +/- 5m	1.64988	1.6477
Run Time	17.8939 min	

Case 2.25

	Start	End
MpW	0.4u	3.20814u
MpL	0.35u	0.35u
MnW	0.4u	5.07527u
MnL	0.35u	0.35u
Vpbias	1.5694	1.353
DcGain >= 1	20.1043	19.0941
unityGain >= 1	13.8855M	129.938M
thermalNoise <= 5n	13.366n	4.22208n
Vdc_out = 1.65 +/- 5m	1.64988	1.65154
Run Time	2.64097 min	

Case 2.26

	Start	End
MpW	0.4u	6.73432u
MpL	0.35u	2.59961u
MnW	0.4u	9.03107u
MnL	0.35u	0.959205u
Vpbias	1.5694	0.9009
DcGain >= 50	20.1043	52.1522
unityGain >= 100M	13.8855M	101.553M
thermalNoise <= 5n	13.366n	4.55812n
Vdc_out = 1.65 +/- 5m	1.64988	1.64632
Run Time	29.0641 min	

For case 25 where the thermal noise constraint is decreased, the width increases and the length stays at minimum geometry as expected. For case 26 the DC gain requirement is increased, the bandwidth is increased and the thermal noise is decreased. As is expected for increased DC gain the lengths increase beyond minimum geometry. For increased bandwidth and decreased noise the widths increase beyond minimum geometry.

•A Two Stage Op-Amp

The third phase of the project is the optimization of a two stage op-amp. The first stage is an NMOS differential amplifier and the second stage is a PMOS common source amplifier.

At the start of this project it was assumed that Mathematica could be used to solve for the exact small signal equations. The method used for this project was to write the node equations from the small signal model and use Mathematica to solve theses equations for the exact transfer function. For the common source amplifier Mathematica was able to solve the equations in a very short time. But for the two stage op-amp Mathematica was not able to solve the node equations in a reasonable about of time. To aid in the understanding of the problem, a discussion of the method to solve for the unity gain frequency follows.

The transfer function as a function of frequency (f) can be written as:

$transfer(f)$ = Real Part + j [Imag Part]

The magnitude is:

$$Mag = \sqrt{(Real\ Part)^2 + (Imag\ Part)^2}$$

The phase is:

$$\phi = \tan^{-1}\left(\frac{\text{Imag Part}}{\text{Real Part}}\right)$$

To solve for the unity gain frequency set:

$$Mag = 1$$

To calculate the unity gain frequency, solve for the frequency where the above relationship is met.

The most challenging part of this is breaking the transfer function into the real and imaginary part. Using the Mathematica function ComplexExpand the program ran for over five days without finding a solution. Also a custom function to do this was written but it ran out of computer memory. It was suggested that the program be ran on a UNIX platform instead of a PC platform. Some users have found Mathematica to perform better on a UNIX workstation than a PC. But due to not having access to

a UNIX version of Mathematica this was not tried. This point is as far as the implementation and testing proceeded.

■Further Work

As a result of the experience gained during implementation and testing, several improvements have been identified for future versions of the software. An important improvement would be to scale the problem so all the variables are on the order of 1 before sending the problem to the optimization software. For this project the variables were left in their native form and the optimization software was allowed to scale the problem. Most of the internal scaling algorithms are developed for large numbers and do not perform well with very small numbers (i.e. for numbers in the μ and n range). For op-amps formatted in their native form, there are noise levels in the nV range, transistor sizes in the μm range, bias voltages of a few volts range, DC gains in the range of 3k to 10k V/V, and bandwidths in the MHz range. This is a huge range of numbers that causes problems if not scaled properly. For this project the scaling problem was worked around by using multiple solvers and rejecting solutions that failed to meet the constraints. Ideally the problem should be scaled before sending it to the optimization solver and then scaled back after the optimization is complete.

The second area for improvement is to implement the BSIM3 DC model instead of using the level 1 model. This would minimize the change that the adjust widths for DC function has to make. But as seen in the "Common Source Amplifier with BSIM3 SPICE Model" section in cases 1 through 18, the optimal width is 0.4μ and the largest width returned is 0.403741μ. This suggests that implementing the BSIM3 model would be a great amount of work for a small return.

Finally, as noted in the op-amp section, Mathematica was unable to solve the op-amp equations in a reasonable amount of time. The first step would be to use Mathematica on a computer platform like UNIX which is known for better memory management than Windows. Another approach would be to custom write functions to do this. Also there are add on packages for Mathematica that may have better performance [16]. The final approach would be to make approximations. Since the small signal parameters and DC model already use first order approximations, these additional approximations may not contribute an unacceptable error.

■Conclusion

This report has described the design and implementation of an algorithm to automate the device sizing in CMOS Op-Amps. As can be seen through the common source amplifier with level 1 SPICE models and the common source amplifier with BSIM3 SPICE models, the basic algorithm works well. The primary shortcoming of this method is that the exact equations for a two stage op-amp could not be solved in a reasonable amount of time. This is an issue that can be addressed in future work.

■Acknowledgments

The author thanks the following people for their contribution to this work:

Dr. Peter Blakey, Northern Arizona University, project advisor

Dr. Paul Flikkema, Northern Arizona University, member project committee

Dr. Phillip Mlsna, Northern Arizona University, member project committee

Dr. Elizabeth Brauer, Northern Arizona University, access to TSMC models

Dr. Hans Mittelmann, Arizona State University, selection of optimization software

Dr. Peter Spellucci, Technische University, Germany, support and guidance for DONLP2

Robin Lougee-Heimer and Andreas Waechter, IBM, support for IPOPT

Richard Waltz, Ziena Optimization, support for Knitro

Robert Vanderbei, Princeton University, support for Loqo

Dr. Collin McAndrew, Freescale Semiconductor, guidance implementing higher order SPICE models in this project

Ira Miller, Freescale Semiconductor, analog design guidance

Kendall Moore, IC Media, whose op-amp design methods provided the starting point for this work

■Appendix A – Coefficients of Equations

Below are the first order equations for MOSFETs that are used in this work and the equivalent coefficient of equation.

First Order Drain Current Equation:

$$Id = \frac{\mu\, Cox}{2} \frac{W}{L - 2\, Ld} (Vgs - Vt)^2 (1 + \lambda\, Vds)$$

Coefficient Equation for Drain Current:

$$Id = Current\ Factor * \left(\frac{W}{L - 2\, Ld} (Vgs - Vt)^2 \right)$$

First Order Gm Equation:

$$gm = \sqrt{2\, \mu\, Cox \frac{W}{L - 2\, Ld}\, Id}$$

Coefficient Equation for Gm:

$$gm = \sqrt{Gm\ Factor \left(\frac{W}{L - 2\, Ld}\, Id \right)}$$

First Order Gds Equation

$$gds = \frac{Ve\ L}{Id}$$

Coefficient Equation for Gds:

$$gds = Gds\ Factor * \left(\frac{L}{Id} \right)$$

First Order Cbd Equation

$$Cbd = W * Cdbo$$

Coefficient Equation for Cbd:

$$Cbd = Cdb\ Factor * W$$

First Order Cbs Equation

$$Cbs = W * Cdso$$

Coefficient Equation for Cbs:

$$Cbs = Cds\ Factor * W$$

First Order Cgd overlap Equation

$$Cgd, ov = W * Cgd, ov$$

Coefficient Equation for Cgd overlap:
$$Cgd,ov = Cgd,ov\ Factor * W$$

First Order Cgs overlap Equation
$$Cgs,ov = W * \ Cgs,ov$$

Coefficient Equation for Cgs overlap:
$$Cgs,ov = Cgs,ov\ Factor * W$$

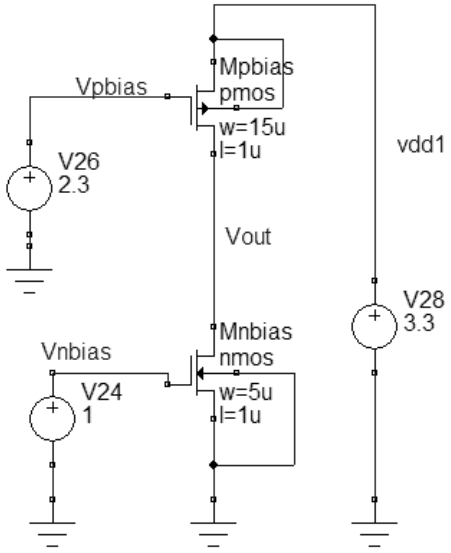
First Order Cgd Equation
$$Cgd = W * Lov * Cgd$$

Coefficient Equation for Cgd:
$$Cgd = Cgd\ Factor * W$$

■Appendix B – Adjust Widths for DC

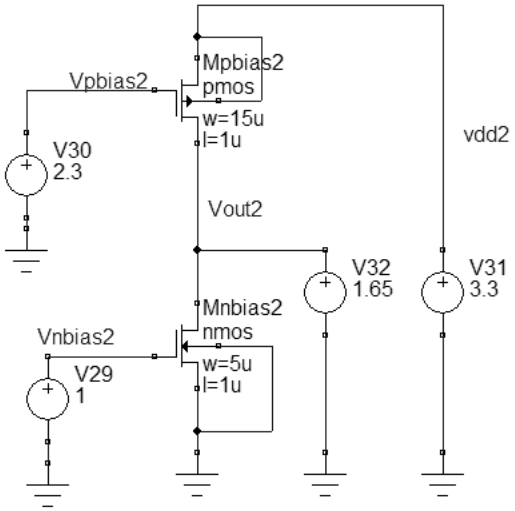
Adjusting widths for DC is a technique where the bias voltages and lengths are fixed and the widths are varied to give the desired DC voltages. This guarantees that the transistors are biased correctly.

To illustrate the adjust widths for DC technique an example is worked. To start with select vdd = 3.3 volt, Vnbias = 1 volt, Vpbias = 2.3 volt, Wn = 5µm, Ln = 1µm, Wp = 15µm, and Lp = 1 µm.



From SPICE simulation Vout,dc = 54.6mV.

Next a DC Voltage source is placed in the circuit to drive the output to the desired DC voltage. For this example select Vdc,out = 1.65 volt.



From SPICE Simulations Id,NMOS = 87.6µ amp and Id,PMOS = 21.5µ amp. To adjust for the desired output voltage the following relationship is used.

$$\frac{Width\ Orginal}{Current\ Orginal} = \frac{Width\ New}{Current\ New}$$

Adjust the PMOS to have the same current as the NMOS therefore

$$Wp = \frac{15u * 87.6u}{21.5u} = 61um$$

When SPICE is re-run without the voltage source at the output and with the new Wp the output voltage is 2 Volt. This is more than the desired output voltage of 1.65 volt but the process can be repeated till the desired accuracy is obtained.

■Appendix C – Software Interface

The final version of the software is written in an object oriented style. The top object is a netlist object that contains all the SPICE objects, DC constraints objects, optimization constraints objects, corner simulations objects, and bias circuit objects.

The following describes the Matlab function calls to run the CS Amp with a current mirror load. Like SPICE this program supports unit postfix i.e. u = 10^-6. The Matlab command is between the <>.

<netlist = add_tsmc35_process(0);>

This returns a netlist object with the first object being a process object. The TSMC 35 process object contains model paths, minimum and maximum geometry size, and the different bins. Also contained is the TopSPICE install directory, and TopSPICE options. The paths in this object have to be changed to work on another computer. The last parameter is a debug parameter. If set to 1 the function prints out what is happening else it prints nothing.

<netlist = add_netlist_element(netlist, 'Mn out nbias 0 0 nmos w=0.4u l=0.35u optimizeWidth=true optimizeLength=true', 0);>

This adds a MOSFET object to the netlist object. The format is the standard SPICE except for the optimizeWidth and optimizeLength parameters. These parameters tell the program it can optimize the length and width. In case these values aren't specified then the default value is true. The last parameter is for debugging, 1 = debug mode and 0 = production mode.

<netlist = add_netlist_element(netlist, 'Mp out pbias rail rail pmos w=0.4u l=0.35u optimizeWidth=true optimizeLength=true', 0);>

This adds another MOSFET object to the netlist object. See above for more details.

<netlist = add_netlist_element(netlist, 'Vpbias pbias 0 DC 1.57871 optimizeVoltage=true', 0);>

This adds a voltage source object to the netlist object. This follows the standard SPICE format except for the optimizeVoltage parameter which tells the software to optimize the voltage. The default value is true. The last parameter is for debugging.

<netlist = add_netlist_element(netlist, 'Vdd rail 0 DC 3.3 optimizeVoltage=false', 0);>

This adds the power supply to the netlist object. Since the power supply is set by the system the software does not optimize the power supply.

<netlist = add_netlist_element(netlist, 'Vnbias nbias 0 DC 1 optimizeVoltage=false', 0);>

This adds the input voltage source to the netlist object. Since the input voltage is set from the driving circuit, the voltage is not optimized.

<netlist = add_netlist_element(netlist, 'c1oad out 0 1p optimizeCap=false', 0);>

This adds a capacitor object to the netlist object. This follows the standard SPICE format except for the optimizeCap parameter which tells the software to optimize the capacitance value. The default value is true. The last parameter is for debugging. Since the load capacitance is set by what the circuit is driving it is not optimized.

<netlist = add_netlist_element(netlist, '.op', 0);>

This adds an op point object to the netlist object. This follows the standard SPICE format. The last parameter is for debugging.

<netlist = add_dc_constraint_object(netlist, 'out=1.65 vary=Mp dontVary=Mn', 0);>

This adds a dc constraint object to the netlist object. The voltage at node "out" is set to 1.65 volts by adjusting the width of transistor Mp and making the current equal to the current in transistor Mn. The section on Adjust Widths for DC explains the method used.

<netlist = add_dc_constraint_object(netlist, 'VoltageTolerance=5m', 0);>

This is the tolerance for the DC output voltage.

<netlist = add_dc_constraint_object(netlist, 'MaxIterations=10', 0);>

This is how many times the program will try to solve for the DC voltage that is within the voltage tolerance. For the TSMC process it would typically take between 3 and 4 iterations to get the output voltage within 5mV.

<netlist = add_opt_object(netlist, 'installDir=xx, 0);>

This is where the optimization software is installed. The <xx> represents the install path. The last parameter is for debugging.

<netlist = add_opt_object(netlist, 'resultsDir=xx', 0);>

This is where the optimization results are stored. The <xx> represents the directory to store the results. The last parameter is for debugging.

<netlist = add_opt_object(netlist, 'optFileName=OptFile', 0);>

This sets the name for the input file for the AMPL optimization software. The last parameter is for debugging.

<netlist = add_opt_object(netlist, 'logFile=AmplLog.txt', 0);>

This sets the name for the output file for the AMPL optimization software. The last parameter is for debugging.

<netlist = add_opt_object(netlist, 'maxIteration=10', 0);>

This sets in maximum number of iterations the software will try to optimizing the circuit. The last parameter is for debugging.

<netlist = add_opt_object(netlist, 'equalityTolerance=1n', 0);>

Because of numerical precision there could be an numerical error in the equality constraints. If an equality constraint is within the equalityTolerance value then they are considered to be equal. The last parameter is for debugging.

<netlist = add_opt_object(netlist, 'option=solver donlp2', 0);>

This sets the AMPL solver to Donlp2. The code is written so that different solvers can be used. The last parameter is for debugging.

<netlist = add_opt_object(netlist, 'option=donlp2_options "silent=1 epsx=1e-007"', 0);>

This sets options for Donlp2 solver. By setting the variable silent=1 this limits the amount of output produced by Donlp2. The variable epsx is the error tolerance. The last parameter is for debugging.

*<netlist = add_opt_object(netlist, 'minimize area: MnW*MnL+MpW*MpL', 0);>*

This is the objective function for the optimization. Here gate area is minimized. The last parameter is for debugging.

<netlist = add_opt_object(netlist, 'param MaxThermalNoise=10n', 0);>

This is a parameter for optimization. Here the MaxThermalNoise is set to 10nV. All the coefficients of equations are added as param automatically. See the section on Making Coefficients of Equations for more information. The last parameter is for debugging.

<netlist = add_opt_object(netlist, 'param MinUnityGainFreq=80M', 0);>

This is a parameter for optimization. Here the MinUnityGainFreq is set to 80MHz. The last parameter is for debugging.

<netlist = add_opt_object(netlist, 'param MinDcGain=100', 0);>

This is a parameter for optimization. Here the MinDcGain is set to 100 V/V. The last parameter is for debugging.

<netlist = add_opt_object(netlist, 'varA MnId=MnCurFac(MnW/(MnL-2*LdNmos))*(Vnbias-MnVth)^2', 0); >*

This adds the NMOS current to the optimization object in the netlist object. For the optimization there are a varA and a varB. VarA are written to the AMPL output file first. Next the small signal parameters are written as a function of the coefficients of equations and device geometry. Finally varB is written to the AMPL output file. The last parameter is for debugging.

<netlist = add_opt_object(netlist, 'varA MpId=MpCurFac(MpW/(MpL-2*LdPmos))*(Vdd-Vpbias-MpVth)^2', 0);>*

This adds the PMOS current to the optimization object in the netlist object.

<netlist = add_opt_object(netlist, 'varB Cf=MnCgdOvl', 0);>

This adds a varB to the optimization object in the netlist object. Cf is the feedback capacitance from the output to the input.

<netlist = add_opt_object(netlist, 'varB Co=c1oad + MnCdb + MpCdb + MpCgdOvl', 0);>

This adds a varB to the optimization object in the netlist object. Co is the total capacitance at the output

<netlist = add_opt_object(netlist, 'varB Gdo=MnGds + MpGds', 0);>

This adds a varB to the optimization object in the netlist object. Gdo is the output resistance of the common source amplifier.

*<netlist = add_opt_object(netlist, 'varB unityGain=sqrt(-Gdo^2 + MnGm^2)/(2*sqrt(Co*(2*Cf + Co))*pi)', 0);>*

This adds a varB to the optimization object in the netlist object. This is the equation for the unity gain frequency.

*<netlist = add_opt_object(netlist, 'varB thermalNoise=sqrt(4*k*(T+270)*(2/(MnGm*3) + 2*MpGm/(3*MnGm^2)))', 0); >*

This adds a varB to the optimization object in the netlist object. This is the equation for the input referred thermal noise. Note the long channel approximation is used for the thermal noise equation.

<netlist = add_opt_object(netlist, 'varB DcGain=MnGm/(Gdo)', 0);>

This adds a varB to the optimization object in the netlist object. This is the equation for the DC Gain of the circuit.

<netlist = add_opt_object(netlist, 's.t. SatP : out <= Vpbias + MpVth', 0);>

This adds a constraint to the optimization object in the netlist object. This makes sure the PMOS remains in saturation. The minimum and maximum geometry constraints are added automatically.

<netlist = add_opt_object(netlist, 's.t. EqualCurrent : MnId == MpId', 0);>

This adds a constraint to the optimization object in the netlist object. This makes the current in the PMOS and NMOS transistor equal.

<netlist = add_opt_object(netlist, 's.t. Gain : DcGain >= MinDcGain', 0);>

This adds a constraint to the optimization object in the netlist object. This makes sure the DC gain is greater than or equal to the allowable minimum.

<netlist = add_opt_object(netlist, 's.t. Thermal : thermalNoise<=MaxThermalNoise', 0);>

This adds a constraint to the optimization object in the netlist object. This makes sure the input referred thermal noise is less than or equal to the allowable maximum.

<netlist = add_opt_object(netlist, 's.t. CUnityGainFreq : unityGain >= MinUnityGainFreq', 0);>

This adds a constraint to the optimization object in the netlist object. This makes sure the unity gain frequency is greater than or equal to the allowable minimum.

<netlist = add_simulation_object(netlist, 'name=typ mosfet=NmosTyp_PmosTyp vdd=3.3 temperature=25 enable', 0);>

This adds a simulation object in the netlist object. This simulation is the typical simulation and has the name "typ". The mosfet model is "NmosTyp_PmosTyp ", the power supply is 3.3 volt, and the temperature is 25C. The enable parameter tells the software to run this simulation when SPICE is called.

<netlist = add_simulation_object(netlist, 'name=bcs mosfet=NmosTyp_PmosTyp vdd=3.6 temperature=25', 0);>

This adds a simulation object in the netlist object. This simulation is the best case simulation. For this case the power supply is raised to 3.6 volts.

<netlist = add_simulation_object(netlist, 'name=wcs mosfet=NmosTyp_PmosTyp vdd=3.0 temperature=25', 0);>

This adds a simulation object in the netlist object. This simulation is the worst case simulation. For this case the power supply is dropped to 3.0 volts.

<netlist = add_simulation_object(netlist, 'maxIteration=10', 0); >

If the circuit doesn't meet spec when simulated over the corners then the spec is adjusted and the process is repeated. The maxIteration variable tells the maximum times the process will be repeated.

<netlist = add_bias_object(netlist, 'remove=Vpbias', 0);>

This adds a bias object in the netlist object. The bias network is created after the optimization. This tells the software to remove the voltage source Vpbias when creating the bias network.

<netlist = add_bias_object(netlist, 'element = Ibias pbias 0 DC 10u', 0);>

This adds a bias object in the netlist object. This tells the software to add a current source when creating the bias network.

<netlist = add_bias_object(netlist, 'element = Mb pbias pbias rail rail pmos', 0);>

This adds a bias object in the netlist object. This tells the software to add a transistor when creating the bias network. Since the lengths and widths are not specified they will be initialized to minimum geometry and will be over written later.

<netlist = add_bias_object(netlist, 'constraint = node=out Mb.l=MpL Mb.w=MpW vary=Ibias copyCurrent=Mp.id percentVar=20 numSteps=10', 0);>

This adds a bias object in the netlist object. This tells the software how to design the bias network. The parameter "node=out" tells the software to make the voltage at node "out" the same after adding the bias network. The parameter "Mb.l=MpL" makes the length of Mb equal to the length of Mp. The parameter "Mb.w=MpW" makes the width of Mb equal to the length of Mp. The parameter "vary=Ibias" tells the software to sweep Ibias and measure the bias current that gives the desired output voltage. The parameters "copyCurrent=Mp.id percentVar=20 numSteps=10" tells the software how much to vary Ibias. The parameter "copyCurrent=Mp.id" is the center point of the sweep of Ibias. The parameter "percentVar=20" and "numSteps=10" tells the software to vary the current ±20% in 10 steps.

<netlist = add_bias_object(netlist, 'maxIteration=10', 0);>

This sets the max number of iterations for generating the bias circuit. If when the bias network is completed the bias voltage is not within the dc spec set by the DC

constraints then the number of steps is doubled and the process repeated until developing the bias circuit or reaching the max number of iterations.

<netlist = run_Masters_Project(netlist, 0);>

This runs algorithm outlined in the section titled Common Source Amp and BSIM3 SPICE Models

■Appendix D – Matlab Functions

The following table lists the Matlab functions and the number of lines of code for each function. The number of lines of code includes comments, debug code, and white space.

	Matlab Function	Lines of Code
1	add_ac_object	104
2	add_bias_object	215
3	add_cap_object	65
4	add_current_object	108
5	add_dc_constraint_object	202
6	add_dc_sweep_object	101
7	add_mosfet_object	176
8	add_netlist_element	72
9	add_op_object	43
10	add_opt_object	295
11	add_process_01	61
12	add_simulation_object	146
13	add_tsmc35_process	237
14	add_tsmc35_process_no_bin	61
15	add_voltage_object	108
16	adjustTransistorSizeForDcVoltage	287
17	bin_mosfet	179
18	change_dc_voltage	74
19	check_all_spec	57
20	check_all_transistors_in_sat	81
21	check_and_adjust_spec	325
22	check_and_adjust_spec_for_all_sims	217
23	check_dc_bias_voltage	101
24	check_geo_sizes	114
25	check_opt_equations	335
26	check_spec	211
27	convert_string_to_float	26
28	copy_new_values_into_netlist	204
29	enable_simulation	69
30	evaluate_equation	122
31	get_ideal_voltage_from_dc_constraint	64
32	get_node_voltage	56
33	get_opt_position	28
34	get_param_constraint	44
35	get_simulated_parameter_value	63
36	get_simulation_position	28
37	getNetlistValue	46
38	is_in_netist	41
39	is_spice_ran	42
40	is_valid_netlist	37
41	isMosfetInNetlist	38
42	make_bias_network	375

43	makeCoef	136
44	print_device_sizes_and_specs	83
45	print_device_sizes	60
46	print_error_message	28
47	print_node_voltages	50
48	printNetlist	20
49	printObject	444
50	printOptVariables	93
51	printParam	37
52	read_ampl_results	219
53	read_value_from_netlist	84
54	readTopSpiceAC	158
55	readTopSpiceDcSweep	157
56	readTopSpiceMosfetOpPoint	405
57	remove_blanks_from_string	13
58	remove_dc_sweep_from_netlist	74
59	remove_instance_from_netlist	81
60	reset_SPICE_Parameters	59
61	run_all_simulations	458
62	run_Masters_Project	155
63	run_opt	744
64	run_opt_with_spice	222
65	run_top_spice	454
66	separateStringIntoParts	67
67	top_cs_amp	99
68	top_cs_amp_various_starting_points	148
69	top_two_stage_op_amp	64
70	write_instance_dot_value_to_netlist	139
71	write_param_constraint	46
72	write_param_constraint_internal	46
73	writeNetlistValue	46
	Total	10147

Note that this project contains more than 10,000 lines of code.

■References

[1] Sahu, Biranchinath and Dutta, Aloke K. Automatic Synthesis of CMOS Operational Amplifiers: A Fuzzy Optimization Approach, IEEE Proceedings of the 15th International Conference on VLSI Design, 2002.

[2] M. Ismail and J. Franca, Introduction to Analog VLSI Design Automation, Kluwer Academic Publishers, London, 1990.

[3] L.R. Carley and R. A. Rutenbar, How to automate analog IC design, IEEE Spectrum, vol. 25, no 8, pp. 26-30, August 1998

[4] J. H. Huijsing, R. J. Plassche, and W. Sansen, Analog Circuit Design, Kluwer Academic Publishers, London, 1993.

[5] Matlab website http://www.mathworks.com

[6] Mathematica website http://www.wolfram.com

[7] TopSPICE website http://www.penzar.com

[8] AMPL website http://www.ampl.com

[9] DONLP2 website ftp://plato.asu.edu/pub/ampl/

[10] MINOS website http://www.sbsi-sol-optimize.com/

[11] Knitro website http://www.ziena.com/knitro.html

[12] SNOPT website http://www.sbsi-sol-optimize.com/

[13] LOQO website http://www.sor.princeton.edu/~loqo/

[14] IPOPT website http://www.coin-or.org

[15] Razavi, B. Design of Analog CMOS Integrated Circuits. McGraw-Hill Education, 2001.

[16] Analog Insydes website http://www.analog-insydes.de/