Robust Analog Circuit Sizing Using Ellipsoid Method and Affine Arithmetic

X. Liu **Wai-Shing Luk** Y. Song P. S. Tang X. Zeng

ASIC & System State Key Laboratory.
Fudan University

ASPDAC, 2007

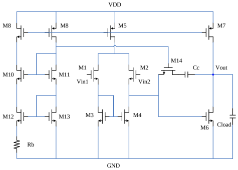
Outline

- Problem Definition of Robust Analog Circuit Sizing
- Robust Geometric Programming
- Ellipsoid Method
- Affine Arithmetic
- Example: CMOS Two-stage Op-Amp
- Numerical Results
- Conclusions

Robust Analog Circuit Sizing Problem

Problem Definition

- Given a circuit topology and a set of specification requirements:



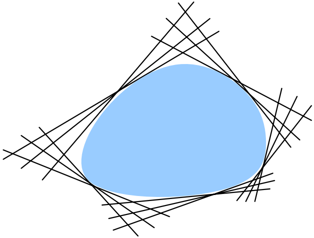
Constraint	Units	Spec.
Device Width	μm	$\geq$ 2.0
Device Length	μm	$\geq$ 1.0
Estimated Area	μm^2	minimize
. . .	. . .	. . .
CMRR	dB	$\geq$ 75
Neg. PSRR	dB	$\geq$ 80
Power	mW	$\leq$ 3
Noise, Flicker	nV/$\sqrt{\text{Hz}}$	$\leq$ 800

- Find the values of design variables that meet the specification requirements in worst case scenarios and optimize the circuit performance.

Robust Optimization Formulation

This problem can be formulated as:

$$\begin{array}{ll} \text{minimize} & \sup_{q \in \mathbb{Q}} f_0(x, q) \\ \text{subject to} & f_j(x, q) \leq 0 \\ & \forall q \in \mathbb{Q} \text{ and } j = 1, 2, \cdots, m, \end{array}$$



where

- $x \in \mathbb{R}^n$ represents a set of design variables (such as L, W),
- q represents a set of varying parameters (such as T_{OX})
- $f_j \leq 0$ represents the jth specification requirement (such as phase margin $\geq 60°$),

Geometric Programming in Standard Form

- We further assume that $f_i(x, q)$'s are convex for all $q \in \mathbb{Q}$ because of geometric programming (GP) technique.
- Geometric programming is an optimization problem that has the following standard form:

$$
\begin{array}{ll}
\text{minimize} & p_0(y) \\
\text{subject to} & p_i(y) \leq 1, \quad i = 1, \ldots, l \\
& g_j(y) = 1, \quad j = 1, \ldots, m \\
& y_k > 0, \qquad k = 1, \ldots, n,
\end{array}
$$

where

- p_i's are posynomial functions and g_j's are monomial functions.

Posynomial and Monomial Functions

- A monomial function is simply:

$$g(y_1, \ldots, y_n) = c y_1^{\alpha_1} y_2^{\alpha_2} \cdots y_n^{\alpha_n}, \quad y_k > 0.$$

  where c is non-negative and $\alpha_k \in \mathbb{R}$.
- A posynomial function is a sum of monomial functions:

$$p(y_1, \ldots, y_n) = \sum_{s=1}^{t} c_s y_1^{\alpha_{1,s}} y_2^{\alpha_{2,s}} \cdots y_n^{\alpha_{n,s}}, \quad y_k > 0,$$

- A monomial can also be viewed as a special case of posynomial in which there is only one term of the sum.

Geometric Programming in Convex Form

- Many engineering problems can be formulated as GP.
- A Matlab package "GGPLAB" and an excellent tutorial material are available from Boyd's website.
- GP can be converted into a convex form by changing of variables $x_k = \log(y_k)$ and replacing p_i with $\log p_i$:

$$\begin{array}{lll} \text{minimize} & \log p_0(\exp(x)) & \\ \text{subject to} & \log p_i(\exp(x)) \leq 0, & i = 1, \ldots, l \\ & a_j^T x + b_j = 0, & j = 1, \ldots, m \end{array}$$

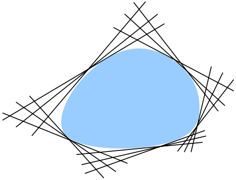
where

- $\exp(x) = (e^{x_1}, e^{x_2}, \cdots, e^{x_n})$, $a_j = (\alpha_{1,j}, \cdots, \alpha_{n,j})$ and $b_j = \log(c_j)$.

Robust Geometric Programming

- GP in the covex form can be solved efficiently by interior point methods.
- In robust version, coefficients c_s are functions of q.

- The robust problem is still convex, but non-differentiable in general. Moreover, there are an infinite number of constraints.
- Alternative approach: Ellipsoid Method.



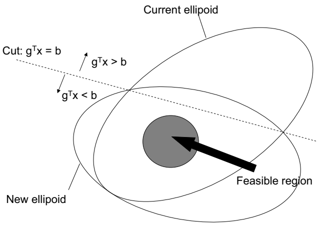
Basic Idea of Ellipsoid Method

- Idea: construct an ellipsoid that bounds the optimal solution.
- At the beginning, an initial ellipsoid is given which is large enough to contain the optimal solution.
- For each iteration, divides the current ellipsoid into two parts and constructs a new smaller ellipsoid that includes only one of the two parts according to x_c, the center of the ellipsoid (see the next slide)
- The process is repeated until the ellipsoid is small enough, or no feasible solution is detected.

Basic Idea of Ellipsoid Method (Cont'd)

For each iteration, divides the current ellipsoid into two parts in such a way that:

- if x_c is infeasible (see the figure), select the part that contains the feasible region.
- if x_c is feasible, select the part that contains the optimal solution.



Ellipsoid Method for Robust Convex Programming

while not converge **do**
    if for some j, $\sup_{q \in \mathbb{Q}} f_j(x_c, q) > 0$ **then** /* x_c infeasible */
        let $f_{\max}(x) = \arg\sup_{q \in \mathbb{Q}} f_j(x_c, q)$.
        find $g = \nabla f_{\max}(x_c)$;
        if $f_{\max}(x_c) - \sqrt{g^T A g} > 0$ **then**
            return infeasible.
        end if
    else /* x_c feasible */
        let $f_{\max}(x) = \arg\sup_{q \in \mathbb{Q}} f_0(x_c, q)$.
        find $g = \nabla f_{\max}(x_c)$;
        if $\sqrt{g^T A g} < \mathrm{tol}$ **then**
            return.
        end if
    end if
    update $\mathit{Ellipsoid}(x_c, A)$.
end while

How to find $\sup_{q \in \mathbb{Q}} f_j(x, q)$ efficiently?

- $\sup_{q \in \mathbb{Q}} f_j(x, q)$ is in general difficult to obtain.
- Provided that variations are small or nearly linear, we propose using Affine Arithmetic (AA) to solve this problem.
- Features of AA:
  - Handle correlation of variations by sharing *noise symbols*.
  - Enabling technology: template and operator overloading features of C++.
  - A C++ package "Libaffa" is publicly available.

Affine Arithmetic for Worst Case Analysis

- An uncertain quantity is represented in an affine form (AAF):

$$\hat{a} = a_0 + a_1\varepsilon_1 + a_2\varepsilon_2 + \cdots + a_k\varepsilon_k = a_0 + \sum_{i=1}^{k} a_i\varepsilon_i,$$

  where $\varepsilon_i \in [-1, 1]$ is called *noise symbol*.
- Exact results for affine operations ($\hat{a} + \hat{b}$, $\hat{a} - \hat{b}$ and $\alpha \cdot \hat{a}$)
- Results of non-affine operations (such as $\hat{a} \cdot \hat{b}$, $\hat{a}/\hat{b}$, $\max(\hat{a}, \hat{b}), \log(\hat{a})$) are *approximated* in an affine form.
- AA has been applied to a wide range of applications recently when process variations are considered.

Affine Arithmetic for Optimization

In our robust GP problem:

- First, represent every elements in q in affine forms.
- For each ellipsoid iteration, $f(x_c, q)$ is obtained by *approximating* $f(x_c, \hat{q})$ in an affine form:

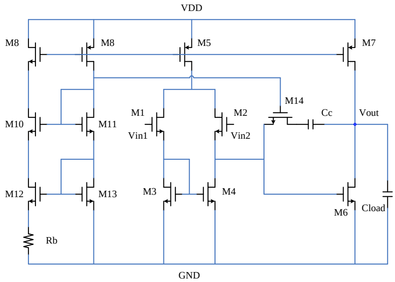
$$\hat{f} = f_0 + f_1\varepsilon_1 + f_2\varepsilon_2 + \cdots + f_k\varepsilon_k.$$

- Then the maximum of $\hat{f}$ is determined by:

$$\varepsilon_j = \begin{cases} +1 & \text{if } f_j > 0 \\ -1 & \text{if } f_j < 0 \end{cases} \qquad j = 1, \cdots, k.$$

Example: CMOS Two-Stage Op-Amp

$$L_1 = L_2$$
$$L_3 = L_4 = L_6 = L_{12} = L_{13}$$
$$L_5 = L_7 = L_8 = L_9$$
$$L_{10} = L_{11} = L_{14}$$
$$W_1 = W_2$$
$$W_3 = W_4 = (A/2) \cdot W_{13}$$
$$W_5 = A \cdot W_{13}$$
$$W_6 = B \cdot W_{13}$$
$$W_7 = B \cdot W_8$$
$$W_9 = W_8$$
$$W_{11} = W_{10}$$
$$W_{12} = 4W_{13}$$



Performance Specification

Constraint	Units	Spec.
Device Width	μm	≥ 2.0
Device Length	μm	≥ 1.0
Estimated Area	μm^2	minimize
Input CM Voltage	x V_{DD}	$[0.45, 0.55]$
Output Range	x V_{DD}	$[0.1, 0.9]$
Gain	dB	≥ 80
Unity Gain Freq.	MHz	≥ 50
Phase Margin	degree	≥ 60
Slew Rate	V/μs	≥ 50
CMRR	dB	≥ 75
Neg. PSRR	dB	≥ 80
Power	mW	≤ 3
Noise, Flicker	nV/$\sqrt{\mathrm{Hz}}$	≤ 800

Example: Open-Loop Gain

- Open-loop gain A_v can be approximated as a monomial function:

$$A_v = \frac{2C_{ox}}{(\lambda_n + \lambda_p)^2} \sqrt{\mu_n \mu_p \frac{W_1 W_6}{L_1 L_6 I_1 I_6}}$$

  where I_1 and I_6 are monomial functions.

- Corresponding C++ code fragment:

```
// Open Loop Gain
monomial<AAF> OLG =
  sqrt(KP*KN*W[1]/L[1]*W[6]/L[6]/I[1]/I[6])
  *2.0/(LAMBDAN+LAMBDAP)/(LAMBDAN+LAMBDAP);
```

Results of Design Variables

Variable	Units	GGPLAB (Std.)	Our (Std.)	Robust
W_1	μm	44.8	44.9	45.4
W_8	μm	3.94	3.98	3.8
W_{10}	μm	2.0	2.0	2.0
W_{13}	μm	2.0	2.0	2.1
L_1	μm	1.0	1.0	1.0
L_8	μm	1.0	1.0	1.0
L_{10}	μm	1.0	1.0	1.0
L_{13}	μm	1.0	1.0	1.0
A	N/A	10.4	10.3	12.0
B	N/A	61.9	61.3	69.1
C_c	pF	1.0	1.0	1.0
I_{bias}	μA	6.12	6.19	5.54

Performances

Performance (units)	Spec.	Std.	Robust
Estimated Area (μm^2)	minimize	5678.4	6119.2
Output Range (x V_{DD})	[0.1,0.9]	[0.07,0.92]	[0.07,0.92]
Comm Inp Range (x V_{DD})	[0.45,0.55]	[0.41,0.59]	[0.39,0.61]
Gain (dB)	$\geq$ 80	80	[80.0, 81.1]
Unity Gain Freq. (MHz)	$\geq$ 50	50	[50.0, 53.1]
Phase Margin (degree)	$\geq$ 60	86.5	[86.1, 86.6]
Slew Rate (V/μs)	$\geq$ 50	64	[66.7, 66.7]
CMRR (dB)	$\geq$ 75	77.5	[77.5, 78.6]
Neg. PSRR (dB)	$\geq$ 80	83.5	[83.5, 84.6]
Power (mW)	$\leq$ 3	1.5	[1.5, 1.5]
Noise, Flicker (nV/$\sqrt{\mathrm{Hz}}$)	$\leq$ 800	600	[578,616]

Conclusions

- Our ellipsoid method is fast enough for practical analog circuit sizing (take < 1 sec. running on a 3GHz Intel CPU for our example).
- Our method is reliable, in the sense that the solution, once produced, always satisfies the specification requirement in the worst case.
- Source code is available at http://sme.fudan.edu.cn/faculty/personweb/luweicheng/ellipsoid+AA/

Questions

S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi.
A tutorial on geometric programming.
Available at http://www.stanford.edu/~boyd/gp_tutorial.html.