Lecture 04c - Affine Arithmetic

.pull-left[

@luk036

2022-10-12

] .pull-right[

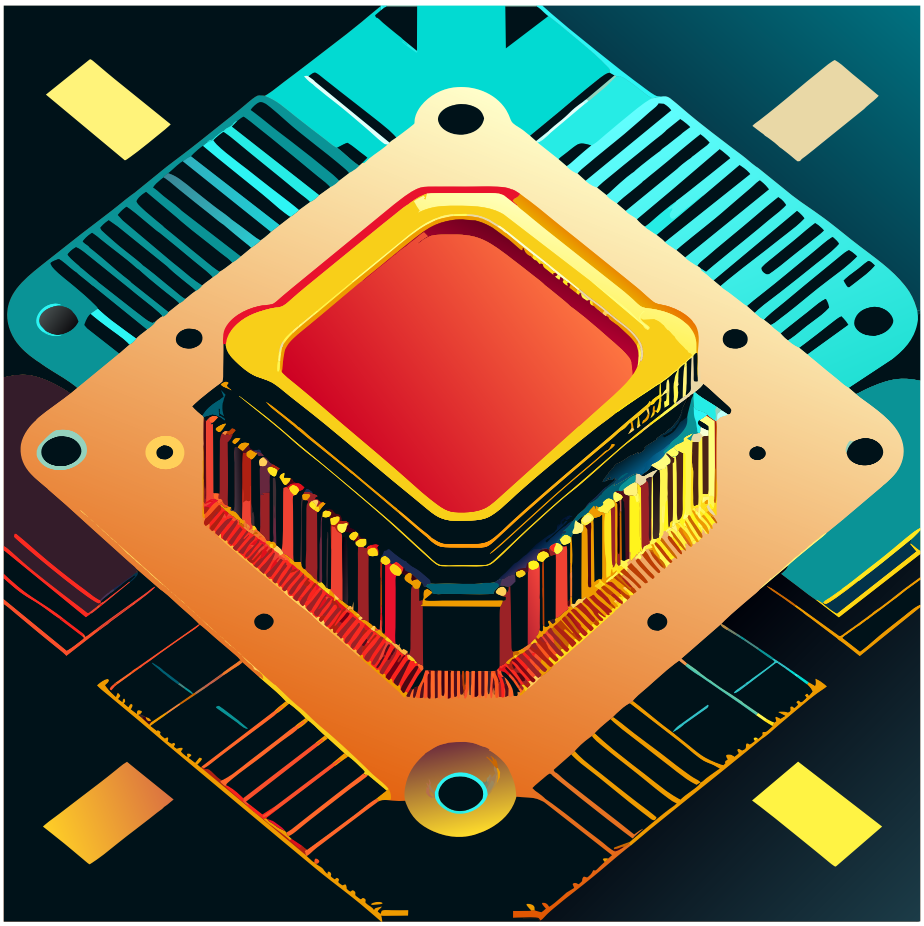

Figure 1: image

]

---

class: center, middle

A simple example: the area of a triangle

A Simple Area Problem

- Suppose the points p, q and r vary within the region of 3 given rectangles.
- Q: What is the upper and lower bound on the area of $\triangle pqr$?

.pull-right[

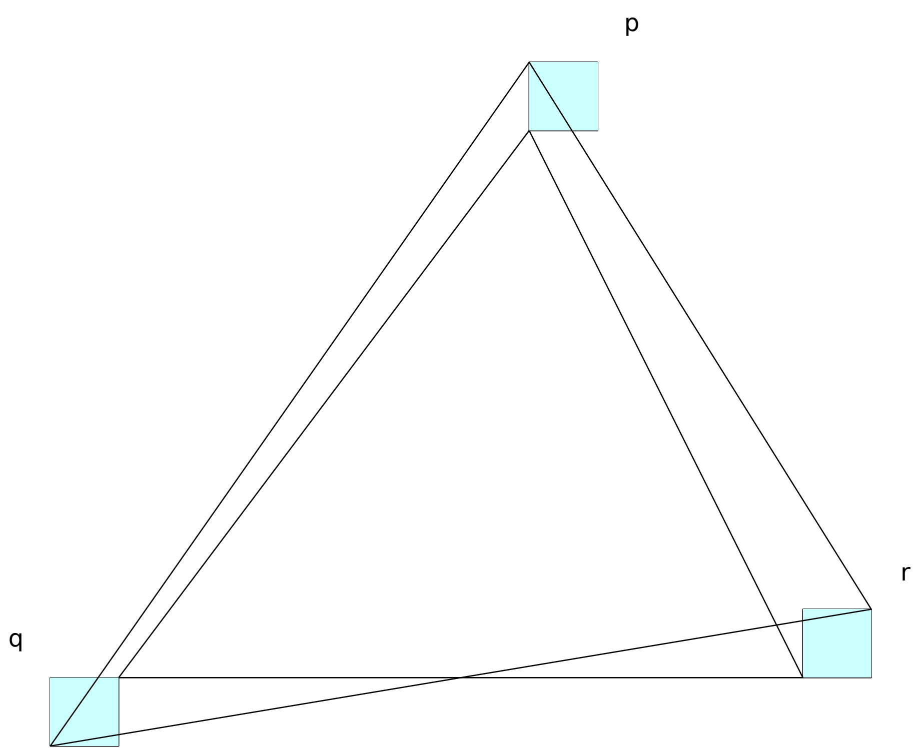

Figure 2: triangle

]

Method 1: Corner-based

- Calculate all the areas of triangles with different *corners*.
- Problems:
  - In practical applications, there may be many corners.

- What's more, in practical applications, the worst-case scenario may not be at the corners at all.

Method 2: Monte Carlo

- Monte-Carlo or Quasi Monte-Carlo:
  - Calculate the area of triangles for different sampling points.
- Advantage: more accurate when there are more sampling points.
- Disadvantage: time consuming

---

class: center, middle

Interval Arithmetic vs. Affine Arithmetic

Method 3: Interval Arithmetic

- Interval arithmetic (IA) estimation:
  - Let px = [2, 3], py = [3, 4]
  - Let qx = [-5, -4], qy = [-6, -5]
  - Let rx = [6, 7] , ry = [-5, -4]
- Area of triangle:
  - = ((qx - px)(ry - py) - (qy - py)(rx - px))/2
  - = [33 .. 61] (actually [36.5 .. 56.5])
- Problem: cannot handle *correlation* between variables.

Method 4: Affine Arithmetic

- (Definition to be given shortly)
- More accurate estimation than IA:
  - Area = [35 .. 57] in the previous example.
- Take care of first-order correlation.
- Usually faster than Monte-Carlo, but ….
  - becomes inaccurate if the variations are large.
- libaffa.a/YALAA package is publicly available:
  - Provides functuins like +, -, *, /, sin(), cos(), pow() etc.

Analog Circuit Example

- Unit Gain bandwidth
  - `GBW = sqrt(A*Kp*Ib*(W2/L2)/(2*pi*Cc)` where some parameters are varying

Enabling Technologies

- C++ template and operator overloading features greatly simplify the coding effort:
- E.g., the following code can be applied to both `<double>` and `<AAF>`:

```
template <typename Tp>
Tp area(const Tp& px, const Tp& qx, const Tp& rx,
        const Tp& py, const Tp& qy, const Tp& ry) {
    return ((qx-px)*(ry-py) - (qy-py)*(rx-px)) / 2;
}
```

- In other words, some existing code can be reused with minimal modification.

Applications of AA

- Analog Circuit Sizing
- Worst-Case Timing Analysis
- Statistical Static Timing Analysis
- Parameter Variation Interconnect Model Order Reduction [CMU02]
- Clock Skew Analysis
- Bounded Skew Clock Tree Synthesis

Limitations of AA

- Something AA can't replace `<double>`:
  - Iterative methods (no fixed point in AA)
  - No Multiplicative inverse operation (no LU decomposition)
  - Not total ordering, can't sort (???)
- AA can only handle linear correlation, which means you can't expect an accurate approximation of `abs(x)` near zero.
- Fortunately the ellipsoid method is one of the few algorithms that works with AA.

Circuit Sizing for Op. Amp.

- Geometric Programming formulation for CMOS Op. Amp.
- Min-max convex programming under Parametric variations (PVT)
- Ellipsoid Method

What is Affine Arithmetic?

- Represents a quantity x with an affine form (AAF):

$$\hat{x} = x_0 + x_1\epsilon_1 + ... + x_n\epsilon_n$$

  where
  - noise symbols $\epsilon_i \in [-1, 1]$
  - central value $x_0 \in \mathbb{R}$
  - partial deviations $x_i \in \mathbb{R}$
  - n is not fixed - new noise symbols are generated during the computation process.
- IA -> AA : [3..4] $\to 3.5 + 0.5\epsilon_1$

Geometry of AA

.pull-left70[

- Affine forms that share noise symbols are dependent:
  - $\hat{x} = x_0 + x_1\epsilon_1 + ... + x_n\epsilon_n$
  - $\hat{y} = y_0 + y_1\epsilon_1 + ... + y_m\epsilon_m$
- The region containing (x, y) is:
  - $Z = \{(x, y) : \epsilon_i \in [-1, 1]\}$
  - This region is a centrally symmetric convex polygon called "zonotope".

] .pull-right30[

]

Affine Arithmetic

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:

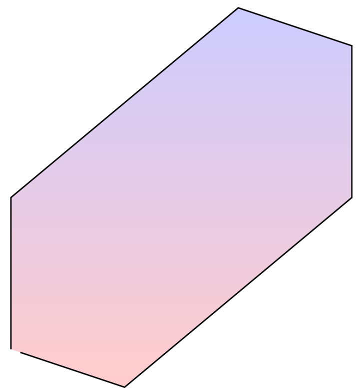

Figure 3: zonotope

- Handle correlation of variations by sharing *noise symbols*.
- Enabling technology: template and operator overloading features of C++.
- A C++ package "YALAA" 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:

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

.pull-left70[

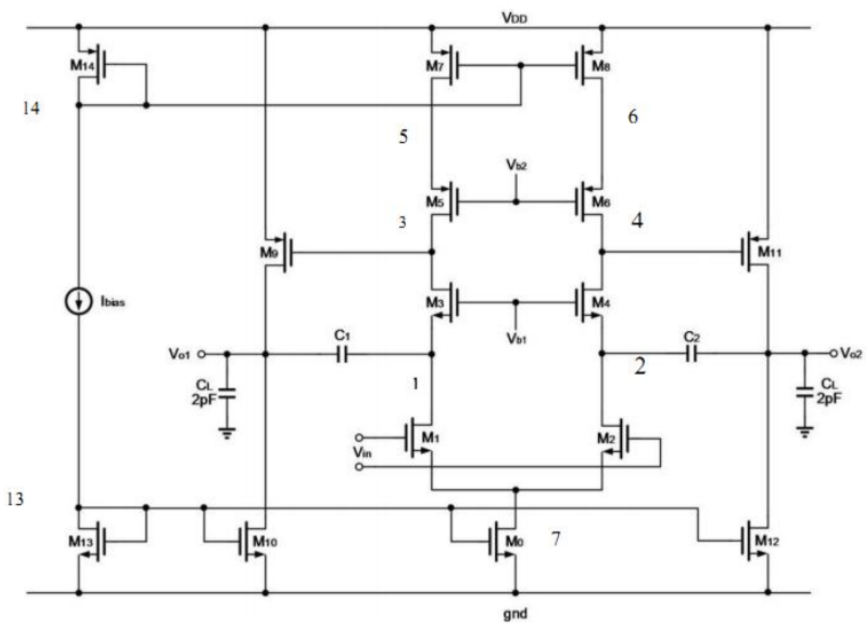

Figure 4: img

]

Performance Specification

.column-2.font-sm.mb-xs[

Constraint	Spec.	Units
Device Width	≥ 2.0	μm
Device Length	≥ 1.0	μm
Estimated Area	minimize	μm^2
Input CM Voltage	$[0.45, 0.55]$	x V_{DD}
Output Range	$[0.1, 0.9]$	x V_{DD}
Gain	≥ 80	dB

Constraint	Spec.	Units
Unity Gain Freq.	≥ 50	MHz
Phase Margin	≥ 60	degree
Slew Rate	≥ 50	V/μs
CMRR	≥ 75	dB
Neg. PSRR	≥ 80	dB
Power	≤ 3	mW
Noise, Flicker	≤ 800	nV/Hz$^{0.5}$

]

Open-Loop Gain (Example)

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

$$A_v = \frac{2C_{ox}}{(\lambda_n + \lambda_p)^2}\sqrt{\mu_n\mu_p\frac{W_1W_6}{L_1L_6I_1I_6}}$$

  where I_1 and I_6 are monomial functions.

- Corresponding C++ code fragment:

```
// Open Loop Gain
monomial<aaf> OLG = 2*COX/square(LAMBDAN+LAMBDAP)*
     sqrt(KP*KN*W[1]/L[1]*W[6]/L[6]/I1/I6);
```

Results of Design Variables

.column-2.font-sm.mb-xs[

Variable	Units	GGPLAB	Our	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

Variable	Units	GGPLAB	Our	Robust
C_c	pF	1.0	1.0	1.0
I_{bias}	μA	6.12	6.19	5.54

]

Performances

.font-sm.mb-xs[

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)	≥ 80	80	[80.0, 81.1]
Unity Gain Freq. (MHz)	≥ 50	50	[50.0, 53.1]
Phase Margin (degree)	≥ 60	86.5	[86.1, 86.6]
Slew Rate (V/μs)	≥ 50	64	[66.7, 66.7]
CMRR (dB)	≥ 75	77.5	[77.5, 78.6]
Neg. PSRR (dB)	≥ 80	83.5	[83.5, 84.6]
Power (mW)	≤ 3	1.5	[1.5, 1.5]
Noise, Flicker (nV/Hz$^{0.5}$)	≤ 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.

Comments

- The marriage of AA (algebra) and Zonotope (geometry) has the potential to provide us with a powerful tool for algorithm design.
- AA does not solve all problems. E.g. Iterative method does not apply to AA because AA is not in the Banach space (the fixed-point theorem does not hold).

- AA * and + do not obey the laws of distribution (c.f. floating-point arithmetic)
- AA can only perform first-order approximations. In other words, it can only be applied to nearly linear variations.
- In practice, we still need to combine AA with other methods, such as statistical method or the (quasi-) Monte Carlo method.