Design of pipeline analog-to-digital converters via geometric programming

Maria del Mar Hershenson
Barcelona Design, Inc.

Abstract

In this paper we present a method for the design of analog-to-digital converters (ADCs). This method computes the sizes of the different components (transistors, capacitors, etc.) in a predefined ADC topology so that the design specifications are met in the desired process technology.

The method is based on formulating the ADC design constraints such as specifications on power, signal-to-noise ratio (SNR), area, and sampling frequency in special convex form in terms of the component sizes of the ADC and intermediate design variables. More specifically, we cast the problem of sizing the components of the ADC as a geometric program. Therefore, all design constraints are formulated as posynomial inequality or monomial equality constraints. Very efficient numerical algorithms are then used to solve the resulting geometric program and to compute the component sizes of an ADC that meets the desired specifications. The synthesis method is fast, and determines the globally optimal design; in particular the final solution is completely independent of the starting point (which can even be infeasible), and infeasible specifications are unambiguously detected.

This paper introduces the concept of hierarchical problem formulation within a geometric programming framework. This modular formulation allows a high re-use of the ADC posynomial model.

1 Introduction

Over the last five years, we have seen a consistent growth in the mixed-signal system-on-chip (SOC) market. Technical advances in integrated circuit (IC) manufacturing processes have made it possible for true electronic systems [1], such as cameras and radio systems, to be integrated in a single silicon substrate. Since these electronic systems need an interface between their digital components and the real world, there is a need for analog interface circuits to be integrated on the same die with the digital components.

The fact that analog and digital circuitry have to co-exist in a single substrate has effectively shortened the required design time for the analog circuitry. When a new process technology is available, digital circuitry can be ported to it quickly with the help of sophisticated computer-aided design tools. However, since the analog part is designed manually, it is ported very slowly. In fact, a simple technology port for an analog circuit can mean a complete redesign.

In the past twenty years, there has been extensive research on the area of analog design automation (see [2]). Approaches to analog design automation can be classified in three groups.

- *Simulation based methods*. These methods evaluate the performance of the circuit with a circuit simulator like SPICE and search the design space using different types of optimization algorithms. For example, some methods use simulated annealing ASTRX/OBLX [3], others gradient search DELIGHT.SPICE [4], others a combination of different search methods (MAELSTROM [5]), *i.e.*The main drawback of this approach is the long time associated with some simulations.

- *Knowledge based methods*. This methods encapsulate the designers knowledge in some form of design plan. Some of the most widely known are based on using special heuristics (like IDAC [6, 7] and OASYS [8] but other expert systems have also been used (like genetic algorithms SEAS [9]). The main disadvantage of this method is the long time needed to set-up the correct heuristic.

- *Equation based methods*. In these methods the circuit performance is described with some sort of analytical design equations. The circuit problem is then cast as an optimization problem, which is then solved by a numerical algorithm. Some implementation examples include OPASYN [10], OPTIMAN [11]. The largest drawback of these methods is their inaccuracy (since the analytical models tend to be too simple). A special case of equation based methods is a method based on formulating the problem as a geometric program. In GPCAD [12] and [13], the authors describe how CMOS op-amps and RF circuits can be modeled in posynomial form with a high level of accuracy. Since a geometric program can be cast as a convex optimization problem, it can be solved globally in a a very short time.

The vast majority of previous analog design automation methods have been tested in small size circuits, such as op-amps. However, in order to meet SOC industry demands, the design of mid size blocks such as data converters need to be automated.

Here we present a method for the design of a pipeline ADC. A pipeline ADC has many more design variables than an op-amp (several hundred versus several tens). Although, there has been some work on automating filter design or ADC design [14], the difference here is that we will size *all* design variables *simultaneously* by solving a single geometric program. In other works, we don't make decisions at the system level and then design the circuits at the transistor level; we simultaneously decide on system level and transistor level variables. Even though, we solve just one large geometric program, the ADC design is posed in a hierarchical manner that allows re-use of the formulation when different building blocks are used.

The contribution of this paper is to show that the design of a mixed-signal system composed of several building blocks can be formulated in an efficient manner as a geometric program. The modular formulation presented here allows to effectively model a

0-7803-7607-2/02/$17.00

mixed-signal circuit. The fact that we can then solve the problem globally (rather than first at the system level and then at the transistor level) results in much more optimal designs.

The paper is organized as follows. In §2, we describe the current custom design methodology for ADCs. In §3, we describe geometric programming, the optimization problem which is the basis of the method. In §4, we describe the geometric programming hierarchical formulation of the ADC design problem. In §5, we describe how to use the method to design a specific pipeline ADC. We start by introducing the ADC architecture used, then we describe the choice of design variables and finally we present the posynomial models for the performance metrics. In §6, we give design examples for the different ADCs and show some design trade-off curves. In §7, we end up with some conclusions and ideas on how to extend the method.

2 Traditional custom ADC design

In a traditional custom design flow, a designer begins with the specifications for the ADC that he needs to to design. He starts by choosing a suitable architecture or topology. After that, the next step is component sizing, in which the designer determines the sizes or values of the components for a given topology or architecture that achieve the requirements or specifications on the performance indices. Even though the numbers of design variables and performance constraints is often "small" by digital circuit design standards (a few hundreds), this task can be very challenging, since in most cases all of the performance indices are affected by all of the design variables.

Component sizing in an ADC is typically done in the following manner. First, design choices are made at the system level or top level. For example, the designer first chooses how many stages and how many bits per stage a pipeline ADC should have and how to distribute the overall power budget amongst stages. Once design choices are made at the system level, the designer drills one level down and proceeds to make additional design choices. For example, in the pipeline ADC example, he will choose what type of amplifier and comparator he is going to use and he will decide how to split the power budget of a stage within the stage. The designer keeps drilling down until he has to choose component dimensions and values of bias voltages and currents.

At each level of hierarchy, the design choices are made in the following manner. In general, the first step is to create a simple mathematical model for the circuit. This model can be written in a language such as MATLAB [15] or it can simply be a hand model and it is used to provide a starting design point. The second step is to verify the initial design point with a simulator. At the transistor level this is typically done using a simulator tool such as SPICE. Typically, each building block is designed separately (by running a large number of simulations) and the overall system is maybe (*only* maybe) simulated once because of the very long simulation times. Unfortunately, this design methodology is not optimal.

It is important to notice that the designer does not simultaneously design all stages but rather makes a lot of choices. For example, if he has a 100mW power budget for a 10 bit pipeline ADC he may decide to spend 20mW in the first stage, 15mW in the second stage and 8.1mW in the remaining stages. This choice may limit the performance he gets. It could be that if he had taken into account the limitations of the circuits he can build, he would have selected 30mW in the first stage, 22mW in the second stage and 6mW thereafter. It is hard to know how to split the power without knowing the power/performance tradeoff in the op-amps.

3 Geometric Programming

Geometric programming (GP) is a special type of convex optimization problem (see [16]). To describe geometric programs, we first introduce two functions.

Let x be a vector $(x_1, \ldots, x_n)$ of n real, positive variables. A function f is called a *posynomial* function of x if it has the form

$$f(x_1, \ldots, x_n) = \sum_{k=1}^{t} c_k x_1^{\alpha_{1k}} x_2^{\alpha_{2k}} \cdots x_n^{\alpha_{nk}}$$

where $c_j \geq 0$ and $\alpha_{ij} \in \mathbf{R}$. When there is only one term in the sum, *i.e.*, $t = 1$, we call f a *monomial* function. Note that posynomials are closed under addition, multiplication, and nonnegative scaling. Monomials are closed under multiplication and division.

A *geometric program* is an optimization problem of the form

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 1, \quad i = 1, \ldots, m, \\ & g_i(x) = 1, \quad i = 1, \ldots, p, \\ & x_i > 0, \quad i = 1, \ldots, n, \end{array} \tag{1}$$

where $f_0, \ldots, f_m$ are posynomial functions and $g_1, \ldots, g_p$ are monomial functions.

If f is a posynomial and g is a monomial, then the constraint $f(x) \leq g(x)$ can be handled by expressing it as $f(x)/g(x) \leq 1$. In a similar way if g_1 and g_2 are both monomial functions, then we can handle the equality constraint $g_1(x) = g_2(x)$ by expressing it as $g_1(x)/g_2(x) = 1$.

A geometric program can be reformulated as a convex optimization problem, by changing variables ($y_i = \log x_i$) and considering the logs of the functions involved.

There are several methods for solving geometric programs. One option is to solve the exponential form of the geometric program using a general purpose optimization code such as NPSOL or MINOS. These general purpose codes will in principle find the globally optimal solution, but codes specifically designed for solving geometric programs offer greater computational efficiency [18]. Recently, Kortanek et al. have shown how the most sophisticated primal-dual interior-point methods used in linear programming can be extended to GP, resulting in an algorithm with efficiency approaching that of current interior-point linear programming solvers [19]. We use a simple primal barrier method, which is described in [17].

4 Design methodology for ADC modeling

In this section we describe the methodology for modeling the design of a pipeline analog-to-digital converter as a geometric program.

The design constraints are formulated hierarchically. First, the system level design constraints of the ADC are formulated in terms of the input specifications of the stages and design variables at the system level (see §5.2 for more detail). For example, the total SNR is written as a posynomial function of the input-referred noise and gain of each stage. Second, the design constraints of the stages of the ADC are formulated as a function of their input and output specifications and their design variables. For example, the input-referred noise of a stage is written as a posynomial function of the input-referred noise of the op-amp and kT/C noise of the stage capacitors. Finally, the design constraints of all stage components (op-amp, comparator, *i.e.*) are written as posynomial functions of their design variables. For example, the input-referred noise of an op-amp is described with a posynomial function in terms of transistor dimensions.

This hierarchical formulation results in a modular description for the geometric program. It results in better maintainability of

the implementation and enables re-use of code when implementing the method for different ADC topologies. For example, if we want to use a different op-amp in the ADC only the much smaller op-amp module of the code needs to be updated.

In summary, the methodology consists of the following steps:

1. *Selecting ADC topology.* Depending on the application of the ADC a certain topology for the ADC is chosen.

2. *Defining levels of hierarchy and corresponding input, output and design variables.* As we will describe in § 5.2, the ADC is divided into three levels of hierarchy. At each level we define:

   - Input variables. These are the input specifications to a circuit block. For example, in the top level the input variables are just the input specifications to the converter like the SNR specification.
   - Design variables. These are the variables that are computed at that level of hierarchy. For example, in the top level a design variable would be the number of stages the ADC needs.
   - Output variables. These are the specifications imposed in the circuits a level below. For example, in the top level the required noise level of each stage would be an output variable.

   Each hierarchy level has a minimal number of defining input, output and design variables that are sufficient to describe the behavior of the ADC and the interaction between levels.

3. *Writing (posynomial) ADC design equations at each hierarchy level in terms of input, output and design variables of sub-blocks.* At this step, the design constraints of the ADC are put in posynomial inequality form in terms of the input, output and design variables of the sub-blocks. This step and the previous step introduce a hierarchical methodology for writing ADC design equations in terms of the ADC component sizes.

4. *Formulate problem as geometric program and solve for component sizes using numerical algorithm.* Once design constraints are put in posynomial form, the ADC design problem is cast as a geometric program and hence it can be readily solved using efficient numerical algorithms.

   Note that even though the formulation is hierarchical, the resulting geometric program is solved in a *flat* manner. In other words, design equations at all levels are solved simultaneously. This can result in a very large optimization problem but given the efficiency of geometric program solvers it does not become an issue.

5 Posynomial model of a pipeline ADC

5.1 Pipeline ADC architecture

To simplify the discussion we consider a specific op-amp topology, the 1 bit per stage ADC of Figure 1. In practice the 1 bit per stage ADC is implemented with a 1.5 bit per stage but the assumption of 1 bit per stage simplifies the explanation of the method.

Pipeline ADCs consist of a set of stages connected in series. Figure 1 shows a single-ended implementation of a one bit per stage resolution pipeline ADC stage. Each stage consists of four capacitors (two in a single ended implementation), a digital to analog converter (DAC), an operational amplifier and ten switches (six in a single ended implementation). The basic operation of the one bit per stage resolution stage is as follows. The analog input is sampled into capacitors C_1 and C_2 during the sampling phase. During the transfer phase, the ADC performs a coarse quantization of the input signal which is subtracted from the held signal and then amplified. The residue of the subtraction/amplification operation is passed down to the next stage for fine quantization. Much more detail about the basic operation of pipeline ADC can be found in [20].

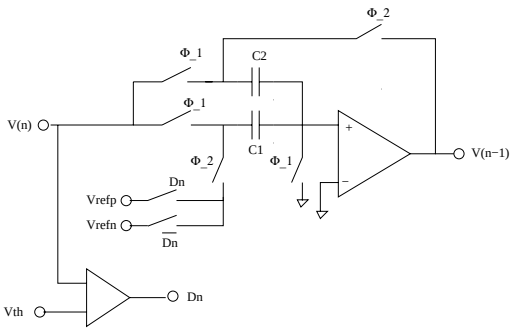

Figure 1: Single-ended implementation of a pipeline ADC stage

Although we have limited ourselves to same number of bits per stage, the method can be extended to architectures composed of stages with different bits of resolution (*e.g.*, first stages two bits and last stages one bit).

5.2 Design and I/O variables

In this section we show how a variety of performance measures and constraints can be formulated using geometric programming. We define three hierarchy levels (see Figure 2),

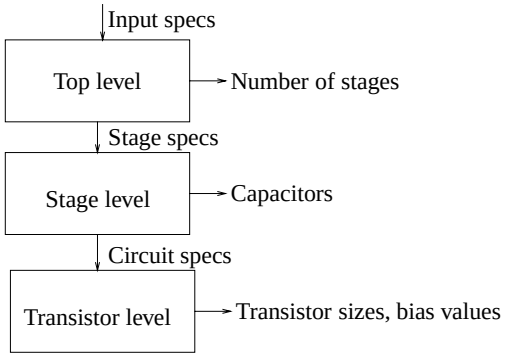

Figure 2: Hierarchical design of a pipeline ADC

- **Hierarchy level one: System level.** The input specifications to this level are the input specifications of the converter:
  - Bits of resolution
  - Power consumption
  - Input signal bandwidth
  - Maximum sampling frequency
  - Clock characteristics
  - Supply voltage

- Input range ($V_{\rm ref}$)
- Input signal common mode
- Dynamic range (DR)
- Linearities (INL and DNL)
- Signal to noise plus distortion ratio ($SNDR$).
- Dimensions of bounding area

The design variables are

- Number of stages

The output specifications are

- Single pipeline stage noise
- Single pipeline stage power
- Single pipeline stage area
- Single pipeline stage offset
- Single pipeline stage nonlinearities
- Specifications for clock generation circuitry, bias circuitry and voltage reference generation circuitry.
- Dimensions of bounding area for each stage

We focus our attention on the pipelined stages. The additional circuitry (clock generation, bias, and voltage reference generation) will be handled with detail in the circuit implementation phase of this project.

- **Hierarchy level two: Stage level.** The second level of hierarchy is composed of the building stages. In this level, the input specifications are the same as the output specifications of the top level. The design variables are the capacitor sizes.

  The output specifications are

  - Op-amp specifications,
    - Unity-gain bandwidth
    - Settling time
    - Load capacitance
    - Power
    - Gain
    - Slew rate
    - Output swing and common mode level
    - Area
    - Nonlinearities
    - Input-referred noise
    - Offset voltage
    - Dimensions of bounding area
  - Switches specifications
    - On resistance
    - Turn-on time
    - Area
    - Nonlinearities
    - Dimensions of bounding area
  - Comparator specifications
    - Power
    - Area
    - Speed
    - Offset voltage and meta-stability range
    - Input-referred noise
    - Dimensions of bounding area.

- **Hierarchy level three: Circuit level.** The circuit level is the third and last level of hierarchy. The input specifications are the same as the output specifications of the previous level.

  The design variables are

  - Width of each transistor
  - Length of each transistor
  - Number of fingers of each transistor
  - Values of passive components (capacitors and resistors)
  - Value of bias currents and bias voltages.

5.3 Performance metrics

1. *System level design constraints*

   - If we assume equal resolution per stage, the number of stages M is defined by the posynomial,

     $$M = N/B, \tag{2}$$

     where N is the number of bits of resolution of the converter and B is the number of bits of resolution per stage. Note that equation 2 is a monomial.
     In the development of this CAD tool, we have assumed that the bits per resolution per stage (B) are equal in all stages. B is an input to the tool and not a design variable. Therefore, in order to evaluate whether one, two or three bits of resolution per stage is better, one must run the CAD tool three times with three different topology specifications ($B = 1$, $B = 2$ and $B = 3$).
   - The dynamic range is given by (see [20]),

     $$DR = 10 \log \frac{\left(2^{N-1}\Delta\right)^2/2}{n_p}, \tag{3}$$

     where n_p , the noise power at the input of the converter, is given by

     $$n_p = n_Q + \sum_{i=1}^{M} \frac{n_{stage_i}}{G^{2(i-1)}}, \tag{4}$$

     where n_{stage_i} is the input referred noise of the ith stage and n_Q is the quantization noise given by

     $$n_Q = \frac{\Delta^2}{12} = \frac{(V_{\rm ref}/2^N)^2}{12}. \tag{5}$$

     Therefore if we want to impose a condition on a maximum allowed dynamic range, we would impose the following posynomial constraint,

     $$n_Q + \sum_{i=1}^{M} \frac{n_{stage_i}}{G^{2(i-1)}} < \frac{\left(2^{N-1}\Delta\right)^2/2}{10^{\frac{DR_{\max}}{10}}}. \tag{6}$$

   - The SNDR is given by

     $$SNDR = 10 \log \frac{\left(2^{N-1}\Delta\right)^2/2}{n_p + h_p}, \tag{7}$$

     where h_p, the harmonic power at the input of the converter, is given by,

$$h_p = \sum_{i=1}^{M} \frac{h_{stage_i}}{G^{2(i-1)}}. \tag{8}$$

Therefore if we want to impose a condition on a maximum allowed SNDR, we would impose the following posynomial constraint,

$$n_Q + \sum_{i=1}^{M} \frac{n_{stage_i}}{G^{2(i-1)}} + \sum_{i=1}^{M} \frac{h_{stage_i}}{G^{2(i-1)}} < \frac{\left(2^{N-1}\Delta\right)^2/2}{10^{\frac{SNDR_{\max}}{10}}}. \tag{9}$$

- The power of the converter is given by the posynomial,

$$P = \sum_{i=1}^{M} P_i + P_\eta, \tag{10}$$

where P_i is the power of the ith stage and P_η is the power consumed by the clock generation circuitry, bias circuitry, voltage reference generation circuitry and buffers.

- The sampling frequency translates into the following monomial constraint,

$$f_{sampling} \leq f_{stage,i}. \tag{11}$$

This condition imposes a minimum operating frequency for each stage.

- The area of the converter is given by the posynomial,

$$A = \sum_{i=1}^{M} A_i + A_\eta + A_{\text{route}}, \tag{12}$$

where A_i is the area of the ith stage, A_η is the area of the clock generation circuitry, bias circuitry, voltage reference generation circuitry, buffers and decoupling capacitors and A_{route} is the area due to the routing of each stage and additional circuitry.

- The floorplan of the converter can also be taken into account. For example, consider the simple transistor layout of Figure 3, where a bounding box defined by dimensions x_d and y_d is given to lay out a 10 bit ADC. This ADC is composed of ten stages, digital circuitry and bias circuitry. Assuming we have the fixed floorplan of Figure 3 we add constraints to make sure that all building blocks fit in the bounding box. For this simple layout, the constraints would be

$$\begin{aligned}
x_{\text{clock}} + x_1 + x_2 &\leq x_d \\
x_5 + x_4 + x_3 &\leq x_d \\
x_6 + x_7 + x_8 &\leq x_d \\
x_9 + x_{10} + x_{\text{bias}} &\leq x_d \\
y_{\text{clock}} + y_5 + y_8 + y_9 &\leq y_d \\
y_1 + y_4 + y_7 + y_{10} &\leq y_d \\
y_2 + y_3 + y_6 + y_{\text{bias}} &\leq y_d.
\end{aligned}$$

2. *Stage level design constraints*

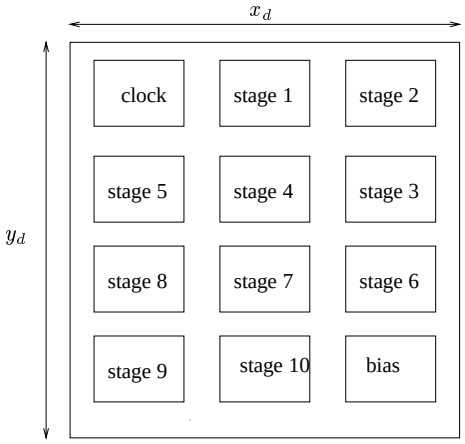

Figure 3: Floorplan of a 10 bit ADC

- The output of the gain block is given by

$$V(n-1) = \left(\frac{A_0}{1+\frac{C_1}{C_2}+A_0}\right)\left(1-e^{\frac{-t}{\tau}}\right) \cdot \left[\left(1+\frac{C_1}{C_2}\right)V(n) + \frac{C_1}{C_2}\left(-D(n)V_{\text{ref,p}} - D(n)V_{\text{ref,n}}\right)\right], \tag{13}$$

where A_0 is the op-amp open-loop gain and τ is the closed-loop time constant. This time constant is related to the unity-gain bandwidth ($UGBW$) and open-loop pole (p_1) of the op-amp by,

$$\tau = \frac{1+\frac{C_1}{C_2}}{p_1\left(1+\frac{C_1}{C_2}+A_0\right)} \approx \frac{1}{2\pi UGBW}\left(1+\frac{C_1}{C_2}\right). \tag{14}$$

In order to operate at the required frequency, the following posynomial constraint must be imposed for the first stage,

$$T_{\text{settle}} + T_{\text{slew}} + T_{\text{clk}} < \frac{1}{2f_{\text{sampling}}}, \tag{15}$$

where $T_{\text{settle}} = 2\tau(N+1)$, T_{clk} is the delay due to the clocking scheme and the worst value for T_{slew} is given by,

$$T_{\text{slew}} = \frac{\Delta V_{\text{in,max}}}{SR} = \frac{V_{\text{ref}}}{2SR}. \tag{16}$$

Note that N is full precision for the first stage, N/G_1 for the second stage, $N/(G_1G_2)$ for the third stage and so on. This means that the settling time constraint (15 is more constraining for the first stages.
Note also that T_{settle} is a posynomial, T_{clk} is an input specification and therefore a constant and that T_{slew} is a posynomial. This means that equation (15) is also a posynomial.

- The load capacitance of the operational amplifier in stage ith (CL_i) during the charge transfer phase is given by,

$$CL_i = C_{\text{transfer,stage},i} + C_{\text{sampling,stage},i-1} + C_p = \frac{\left(C_{\text{input}} + C_{1,i}\right)C_{2,i}}{C_{\text{input}} + C_{1,i} + C_{2,i}} + (C_{2,i+1} + C_{1,i+1} + C_{\text{comp}}) + C_p, \tag{17}$$

where $C_{\rm input}$ is the input capacitance of the op-amp in stage ith, $C_{\rm comp}$ is the input capacitance of the comparator in the $i+1$th stage and C_p is the parasitic capacitance from the switches.Note that equation (17) is not posynomial. Even though this equation is not posynomial it can be very well approximated when the bits per resolution per stage is known. For example if $B = 1$, $C_1 = C_2$ and we can write the posynomial,

$$\begin{aligned} CL_i &= \frac{C_{1,i}}{2}\left(1+\frac{C_{\rm input}}{C_{1,i}}\right) \\ &\quad + (C_{2,i+1}+C_{1,i+1}+C_{\rm comp}) + C_p, \end{aligned} \tag{18}$$

- The power of the stage is given by the posynomial

$$P = P_{\rm op-amp} + P_{\rm comparator}, \tag{19}$$

where $P_{\rm op-amp}$ is the power consumed by the op-amp and $P_{\rm comparator}$ is the power consumed by the comparator.

- The stage gain is given by

$$\text{Gain} = 1 + \frac{C_1}{C_2} = 2^B \tag{20}$$

where B is the bits per resolution per stage. Since we fix B at the beginning of the problem, we can write the following monomial constraint,

$$\frac{C_1}{C_2} = 2^B - 1 \tag{21}$$

- The area of a stage is given by the posynomial equation,

$$\begin{aligned} A &= A_{\rm op-amp} + A_{\rm comparator} + 2A_{C_1} + \\ &\quad 2A_{C_2} + \sum_j A_{{\rm switch},j} + A_{\rm route}, \end{aligned} \tag{22}$$

where $A_{\rm op-amp}$ is the op-amp area, $A_{\rm comparator}$ is the comparator area, A_{C_1} is the area of capacitor C_1, A_{C_2} is the area of capacitor C_2, $A_{{\rm switch},j}$ is the area of each switch, and $A_{\rm route}$ is the routing area.

- The noise for the ith stage is given by the sum of the thermal noise in the switches and the amplifier noise (see [21] for a detailed derivation),

$$\overline{e_i^2} = 2kT\left(\frac{2}{C_1}+\frac{2}{C_2}\right) + \overline{e_{{\rm amp},i}^2}, \tag{23}$$

where $\overline{e_{{\rm amp},i}^2}$ is the input-referred op-amp noise given by

$$\overline{e_{{\rm amp},i}^2} = \frac{S_o}{4\tau}, \tag{24}$$

where S_o is the op-amp input-referred white noise density (given by a posynomial, see [12]) and τ is the closed loop time constant given by equation (14).

Since S_o is a posynomial and τ a monomial, equation (23) is a posynomial.

- The output swing of the op-amp has to be at least as large as the reference signal. In order to guarantee high linearity, we leave some extra margin. The output swing constraints on the op-amp are given by the following monomial constraints,

$$\begin{aligned} V_{\rm out,max} &\geq (1+\alpha)\left(V_{\rm CM}+\tfrac{V_{\rm ref}}{2}\right) \\ V_{\rm out,min} &\leq (1-\alpha)\left(V_{\rm CM}-\tfrac{V_{\rm ref}}{2}\right), \end{aligned} \tag{25}$$

where α is a factor that accounts for the linearity margin (typically a few hundred millivolts are reasonable), $V_{\rm CM}$ is the common mode output voltage and $V_{\rm ref}$ is the reference voltage.

- In order to minimize the nonlinearities due to the op-amp gain nonlinearity and reduce the finite op-amp gain effect, the gain of the amplifier has to be sufficiently high. We can achieve this by imposing the monomial constraint,

$$A_{o,i} \geq 2^{N_i+1}, \tag{26}$$

where $A_{o,i}$ is the op-amp gain of the ith stage and N_i is the precision bits required in the ith stage.

- In order to achieve a good settling behavior, the phase margin for the amplifier has to be sufficiently high. In practice, guaranteeing that the phase margin is at least seventy degrees ensures a good settling behavior.

- The causes of nonlinearities in a stage are several [22]. The cause of DC-nonlinearity include: amplifier and comparator offsets, capacitor mismatch and nonlinearity, amplifier finite gain and nonlinearity, and switch charge injection. Since we are imposing conditions (26 and 25), we can neglect the nonlinearity due to the amplifier gain. Capacitor mismatch and nonlinearity data are process dependent parameters and as such they are an input to the tool. Therefore we can write (see [22]),

$$\begin{aligned} h_{\rm stage} &= (e_C V_{\rm ref})^2 + (\text{offset}_{\rm op-amp})^2 + \\ &\quad (\text{offset}_{\rm comparator})^2 + (q_{\rm ci})^2, \end{aligned} \tag{27}$$

where e_C is the percent mismatch and nonlinearity of the capacitors, $\text{offset}_{\rm op-amp}$ is the input offset of the op-amp, $\text{offset}_{\rm comparator}$ is the input offset voltage of the comparator and $q_{\rm ci}$ is the total charge injection due to the switches of the stage (see [23]).

Dynamic nonlinearity has not been modeled. In practice, this is a difficult performance metric to model and simulate.

- The switches must have an enough small associated time constant,

$$R_{\rm on}C_{\rm sw} \ll 1/f_{\rm stage}, \tag{28}$$

where $C_{\rm sw}$ is the capacitance seen by each switch. If we impose a factor of ten difference between the switching time constant and the sampling frequency equation (28) is a posynomial.

- By fixing a priori the floorplan of each stage we can impose posynomial constraints on the floorplan (in a similar way to what we did in the top level of hierarchy).

3. *Circuit level design constraints*

The problem of modeling an op-amp as a geometric program has been described in previous papers [12] and will not be described here. Modeling a comparator and switch operation can be done in a similar manner. Recently, new methods for automatically developing posynomial circuit models

have been reported [24]. These new methods can also be used successfully (especially because of the small number of variables associated with the comparator and the switches).

One could envision defining a fourth level of hierarchy, defined by the actual transistors. In other words, in the third level we would formally write op-amp specifications in terms of transistor parameters and in the fourth level we would relate transistor parameters to transistor sizes. Transistor behavior can be modeled in posynomial manner. A simple posynomial model (*GP1 model*) was described in [12] but more complex posynomial models can also be developed.

6 Design examples

We have implemented a simple CAD tool for the design of pipeline ADCs with one bit per resolution stages. We have modeled in posynomial form a gain boosted op-amp [12], a dynamic comparator and a set of simple NMOS switches. The input to the CAD tool are the converter specifications: number of bits, maximum SNR, maximum sampling frequency, maximum area and maximum power. The output are the sizes of the capacitors and the transistor sizes for the op-amp, comparator and switches.

The tool is written in C code. After reading a set of specifications, a geometric program consisting of all the posynomial and monomial equations shown. For a 12 bit ADC, the problem contains 2364 variables and it is solved in under 10 minutes on a 400MHz Pentium PC with 128MB of memory.

In Figure 4 we show the tradeoff curve power versus sampling frequency for a 12 bit ADC and a 10 bit ADC. We impose an SNR of 70dB for the 12 bit ADC and of 58dB for the 10 bit ADC (in both cases a loss of 4dB due to circuit noise). This is done by repeatedly solving the ADC design problem (minimize power) while varying the sampling frequency specification. We observe (as expected) that the higher the required sampling frequency the more power is needed. What this curve allows us to understand is how exactly these specifications trade-off. Note that less power is required for a 10 bit ADC but it also has a smaller SNR. Typical capacitor mismatches limit the resolution of high speed ADCs to 10 bits and calibration techniques are needed to achieve higher resolution. Here we have simulated the use of calibration by using a very small capacitor mismatch parameter.

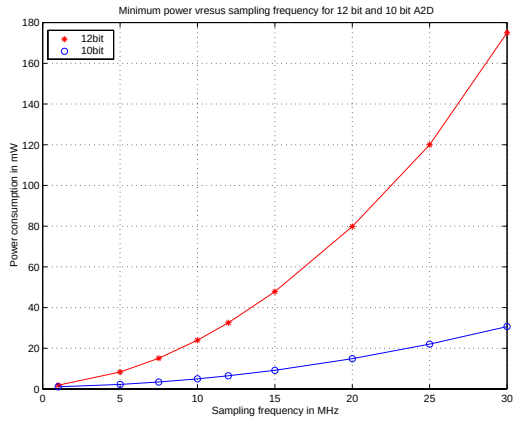

Figure 4: Minimum power for 12 bit and 10 bit ADC.

Recently, there has been some work done to define optimum scaling in pipeline ADC [25]. However very simple models were used to determine this optimum scaling. Using the approach presented here we can quickly determine what is the optimum scaling. Figure 5 and figure 6 show the optimum power per stage and optimum capacitance scaling for the case of a 12 bit ADC operating with a maximum sampling frequency of 20MHz. We observe that from the fifth stage on, all stages are the same size and scaling only takes place in the initial stages. The reason is that after the fifth stage, the op-amp behavior is determined by its own parasitics rather than by the switching capacitances which are quite small in later stages (small capacitances in later stages are possible because their kT/C noise is attenuated by the gain of the initial stages). The fact that the last stages are identical, allows to reduce the size of the problem by almost half. The idea is to assume the first six stages are unique and the last six stages are identical.

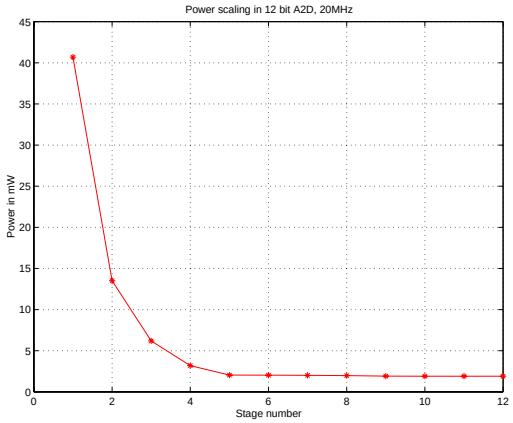

Figure 5: Power scaling for 12 bit ADC, 20MHz.

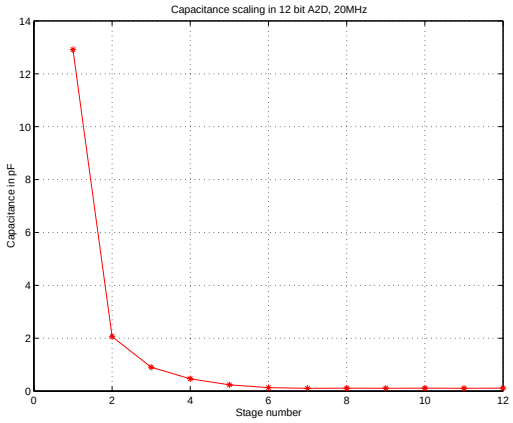

Figure 6: Capacitor scaling for 12 bit ADC, 20MHz.

7 Conclusions and extensions

In this work we have shown how to use geometric programming to design a pipeline ADC. Since the method is very efficient, we can simultaneously size all pipeline stages. Since we a solving a convex problem, the results obtained are globally optimal.

An important feature of geometric programming based design is the ability to develop *robust* designs, *i.e.*, designs that meet the required specifications are met for a variety of different technology parameter values and operating conditions. To do this, one just needs to replicate the design constraints for the different scenarios, which is practical only because the computational effort for solving geometric programs grows approximately linearly with the number of constraints. The ability of creating robust designs is perhaps the most important feature of geometric programming based design since most of the time, designers are more concerned with having a circuit that works over all corners than a globally optimal circuit.

The method presented does not create new ADC topologies. What it does is size a previously defined ADC architecture. However, the modularity of the method allows to effectively create new topologies. For example, if we have a library of ten op-amps and five comparators, we can envision several possible combinations (all stages have same op-amp and comparator; first four stages use a certain op-amp and certain comparator and the rest of stages use a different type, *i.e.*). In order to evaluate all different combinations one does not need to re-formulate the entire ADC design problem. One only needs to formulate the module in question as a geometric program following the convention of required input, output and design variables. The search of all possible combinations is a combinatorial problem. Since geometric programs can be solved very fast, one can just search the space by solving each possible ADC architecture. At some point, however this search becomes too long (when too many combinations are possible) so more efficient search schemes would be needed (this investigation is not the purpose of this paper).

The approach we have described is not limited to pipeline ADCs. Hierarchical GP formulation can be used to describe other data converters and other mixed-signal blocks such as phase-locked-loops. The methodology is identical to the one described in §4.

References

[1] B. Martin. Electronic design automation. *IEEE Spectrum*, 36(1):57–61, January 1999.

[2] G. Gielen and R. Rutenbar. Computer-aided design of analog and mixed-signal integrated circuits. *Proceedings of the IEEE*, 88(12):1825–1852, December 2000.

[3] E. S. Ochotta, R. A. Rutenbar, and L. R. Carley. Synthesis of high-performance analog circuits in ASTRX/OBLX. *IEEE Transactions on Computer-Aided Design*, 15:273–293, March 1996.

[4] W. Nye, D. C. Riley, A. Sangiovanni-Vincentelli, and A. L. Tits. DELIGHT.SPICE: An optimization-based system for the design of integrated circuits. *IEEE Transactions on Computer-Aided Design*, 7:501–518, April 1988.

[5] M. Krasnicki, R. Phelps, R. A. Rutenbar, and L. R. Carley. Maelstrom: Efficient simulation-based synthesis for custom analog cells. In *Proceedings of the 31st Annual Design Automation Conference*, pages 945–950, 1999.

[6] M. G. R. Degrauwe, O. Nys, E. Dijkstra, J. Rijmenants, S. Bitz, B. L. A. G. Goffart, E. A. Vittoz, S. Cserveny, C. Meixenberger, G. Van Der Stappen, and H. J. Oguey. IDAC: An interactive design tool for analog CMOS circuits. *IEEE Journal of Solid-State Circuits*, 22:1106–115, December 1987.

[7] M. G. R. Degrauwe, B. L. A. G. Goffart, C. Meixenberger, M. L. A. Pierre, J. B. Litsios, J. Rijmenants, O. J. A. P. Nys, E. Dijkstra, B. Joss, M. K. C. Meyvaert, T. R. Schwarz, and M. D. Pardoen. Towards an analog system design environment. *IEEE Journal of Solid-State Circuits*, 24:1587–1597, December 1989.

[8] R. Harjani, R. A. Rutenbar, and L. R. Carley. OASYS: A framework for analog circuit synthesis. *IEEE Transactions on Computer-Aided Design*, 8:1247–1265, December 1989.

[9] Z. Ning, T. Mouthaan, and H. Wallinga. SEAS: A simulated evolution approach for analog circuit synthesis. In *Proceedings IEEE Custom Integrated Circuits Conference*, pages 5.2.1–5.2.4, 1991.

[10] H. Y. Koh, C. H. Séquin, and P. R. Gray. OPASYN: A compiler for CMOS operational amplifiers. *IEEE Transactions on Computer-Aided Design*, 9:113–125, February 1990.

[11] G. G. E. Gielen, H. C. C. Walscharts, and W. M. C. Sansen. Analog circuit design optimization based on symbolic simulation and simulated annealing. *IEEE Journal of Solid-State Circuits*, 25:707–713, June 1990.

[12] M. Hershenson, S. Boyd, and T. H. Lee. GPCAD: A tool for CMOS op-amp synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, San Jose, CA, November 1998.

[13] M. Hershenson, S. Mohan, S. Boyd, and T. H. Lee. Optimization of inductor circuits via geometric programming. In *36th IEEE/ACM Design Automation Conference*, June 1999.

[14] F. Medeiro, B. Pérez-Verdú, A. Rodríguez-Vázquez, and J. L. Huertas. Towards an analog system design environment. *IEEE Journal of Solid-State Circuits*, 30:762–772, July 1995.

[15] The Mathworks. Matlab 6.1. http://www.mathworks.com/products/matlab/.

[16] R. J. Duffin, E. L. Peterson, and C. Zener. *Geometric Programming — Theory and Applications*. Wiley, 1967.

[17] S. Boyd and L. Vandenberghe. Introduction to convex optimization with engineering applications. Course Notes, 1997. `http://www.stanford.edu/class/ee364/`.

[18] K. O. Kortanek. Geometric programming tutorial. Technical report, INFORMS, San Diego, CA, May 1997.

[19] K. O. Kortanek, X. Xu, and Y. Ye. An infeasible interior-point algorithm for solving primal and dual geometric programs. *Math Programming*, 76:155–181, 1996.

[20] B. Wooley. EE315 course notes. Stanford University, CA, April 2001.

[21] J. M. Ingino. *Continuous calibration for high-accuracy A/D conversion*. PhD thesis, Stanford University, March 1999.

[22] K. Nagaraj. High-speed pipeline A/D converters. Notes for MEAD Microelectronics course on high-speed data converters, November 2001.

[23] B. Razavi. *Principles of Data Conversion*. IEEE, Press, Piscataway, 1994.

[24] W. Daems, G. Gielen, and W. Sansen. Simulation-based automatic generation of signomial and posynomial performance models for analog integrated circuit sizing. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 70–74, San Jose, CA, November 2001.

[25] D. W. Cline. *Noise, speed and power tradeoffs in pipelined Analog to Digital Converters*. PhD thesis, University of California, Berkeley, May 1998.