In: Proceedings of the Forum on
Specification and Design
Languages'03; Frankfurt, Sept. 2003;
ECSI.

A New Method for Modeling and Analysis of Accuracy and Tolerances in Mixed-Signal Systems

Wilhelm Heupke, Christoph Grimm, Klaus Waldschmidt
University of Frankfurt
Robert-Mayer-Str. 11-15, D-60325 Frankfurt
{heupke | grimm | waldsch}@ti.informatik.uni-frankfurt.de

Abstract

Tolerances are a very important property of a design. This paper presents a method for simulating tolerances in signal processing and control systems on the system level using affine arithmetic. The method is compard to simulation based on interval arithmetic or Monte-Carlo method and goes into details of the shortcomings of interval arithmetic. An introduction to affine arithmetic is given and the corresponding model of tolerances is described. The concept of the implementation is given and a case study of a control loop modeled on the system level is presented. This small system is simulated using affine arithmetic and is based on an early prototype of SystemC-AMS.

1 Introduction

In analog systems even small errors e.g. due to parameter variations can lead to erroneous behavior of the overall system. However, 'good' analog designs are robust to such tolerances. This means that small changes have no or very little impact on the output. A number of known methods help designers to verify that a designed analog system is robust with respect to different types of inaccuracies: Noise analysis considers the impact of noise sources on the output. However, it is restricted to the frequency domain.

The well-known Monte-Carlo method allows the designer to determine the impact of inaccurate parameters to outputs. As the estimation accuracy depends on the sample size, larger numbers of simulations are required. For practical application these are often prohibitive. The Monte-Carlo method is typically used for analyzing this impact of tolerances but it is a non-deterministic and heuristical method. Furthermore, it is comparably slow. On the other hand a system with tolerances, that is composed of linear components can be simulated more efficiently. Many signal processing chains consist only of linear systems up to some point. We will see how to use affine arithmetic for that application.

Another approach is using nominal values for all variables except for one, which is once simulated with the upper and once with the lower boundary. Then the next tolerance is chosen. It is missing the effect of the combination of several tolerances that might together cross the threshold of a comparator for example. This could be achieved by the calculation of the cross product of all tolerances. This means running the simulation once with the lower boundary and once with the upper boundary of an interval. With more tolerance terms one has to calculate each combination of tolerance term boundaries, leading to an exponential growth of computing time in the number of tolerance sources (Table 1). We will see that for efficient concepts feedback can be a problem when simulating tolerances. In yield analysis such as [AGW93] the impact of static stochastic parameter variations to static parameters of the design is analyzed.

In this work a method for validation by simulation is described. This enables designers to analyze the impact of parameter variations to the dynamic behavior of analog or mixed-

method	complexity	comment
Monte-Carlo	O(1)	non-deterministic, very large constant
interval arithmetic	O(1)	unacceptable for feedback systems
affine arithmetic	O(n)	restricted to linear systems, yet
calculating deltas	O(n)	no effects of combination of tolerances
calculating the cross product	O(2^n)	inefficient for larger number of tolerances

Table 1: Complexity of different methods in the number of tolerances

signal non-conservative systems. These systems are modeled on the system level by components connected by signals. Such results help to specify the required precision/tolerance of components, resolution of converters, and the maximum allowed tolerance level on input signals.

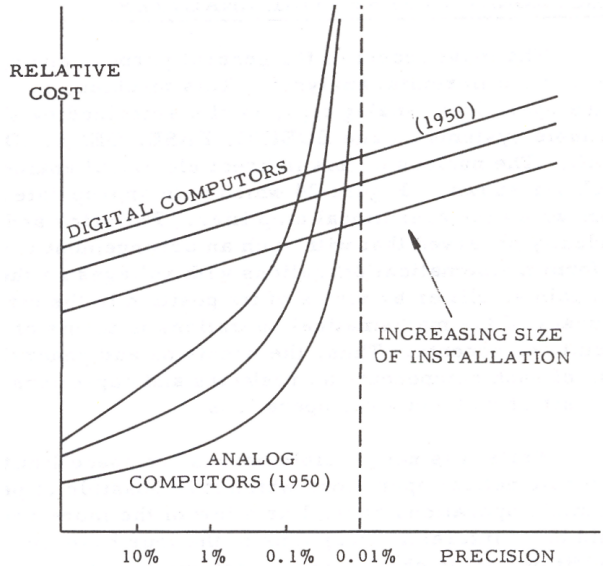

Figure 1: Cost versus accuracy with analog or digital design implementation

Analog signals have a nearly infinite resolution but a very limited precision. The precision is a very important problem, as the cost increases extremely with higher precision. Figure 1 was meant to explain precision problems with analog computers and is dated 1950 [Roe55], but nevertheless it is still true. Raising precision of analog signals leads to a tremendous increase of the cost which is as critical to the market success of a design as precision is.

For example think about analog sensor interfaces - their precision is often really critical. A design of a digital fever thermometer can illustrate the problem. Such a device with a precision of 5 °C will be of no value. On the other hand, a device with a precision of 0.01 °C has no greater value than one with 0,1 °C for this application but is a lot more expensive to manufacture. Therefore, it is important to evaluate the influence of uncertainties at several points in the circuit on the outputs in an early design stage. The main target applications for this approach are signal processing and control systems.

2 Aspects of Interval Arithmetic for Simulation

Many parameter variations that occur in analog systems can be modeled by intervals with an upper and a lower bound. Typical examples are tolerances of parameters of components after production and selection or the drift with temperature. It is very appealing to compute the

impact of such intervals by using interval arithmetic or constraint systems, as for example in [OGW02]. In this publication a constraint solver is described that directly computes the intervals of dependent parameters.

However, for the analysis or simulation of complex dynamic systems, interval arithmetic has the problem of over-approximations. Taking a look at the basic operations on intervals $A = [\underline{a}, \bar{a}]$ and $B = [\underline{b}, \bar{b}]$ will make it clear:

$$A - B = [\underline{a}, \bar{a}] - [\underline{b}, \bar{b}] = [\underline{a} - \bar{b}, \bar{a} - \underline{b}]$$
$$A + B = [\underline{a}, \bar{a}] + [\underline{b}, \bar{b}] = [\underline{a} + \underline{b}, \bar{a} + \bar{b}]$$
$$c * A = [c * \underline{a}, c * \bar{a}]$$
$$d(X) = \bar{x} - \underline{x}$$

Two important properties are:

$$d(c * A) = c * d(A)$$
$$d(A \pm B) = d(A) + d(B)$$

This means it does not matter if a subtraction or an addition of two signals is performed; the diameter of the new interval is always the sum of the two diameters of the operands and thus grows larger with each addition or subtraction. For a control loop this results in at least a linear increase of the interval with each loop iteration. With a typical feedback loop it is even worse. Figure 2 shows a proportional regulator.

The subtraction will increase the diameter of the interval with each iteration and the proportional regulator will increase the diameter by a constant factor which is typically $\gg 1$. This leads to a multiplication of the diameter with a constant larger than one with each iteration and thus an exponential growth of the diameter with the number of iterations of the loop in a time discrete model. The control variable output in the simulation is the interval $[-\infty, +\infty]$ after several iterations. This result encloses the possible values, but is obviously far too large.

This effect would not happen if the subtraction of the control variable from the setpoint would cancel out correlated tolerance terms, but if the subtraction fails to reflect the correlation of the error of the two operands this arithmetic is inappropriate for most systems with loops. But even with straight signal processing chains interval arithmetic can deliver too loose approximations if the path splits up and reconverges. Especially with the above mentioned feedback loops in combination with interval arithmetic, one ends up with the above mentioned problem. It is not clear if the two intervals have the same numerical values or if they are the same. In general the problem with interval arithmetic is that it does not consider that certain signals (and their tolerances) are correlated.

A constant over-approximation in each iteration will lead to a linear growth of the diameter of the interval and a proportional over-approximation will lead to an exponential growth. Interval arithmetic assumes that both operands are not correlated and each value from the interval of the right operand could be subtracted from any value from the interval of the left operand. Consider the calculation $X - X$ which is nearly what happens in the control loop example. Using interval arithmetic this calculation delivers an interval centered around zero with a diameter that is twice as large ($d(X - X) = 2d(X)$) like the diameter of each individual X. This is not realistic as both X are exactly the same and thus the error terms are correlated.

3 An Alternative to Interval Arithmetic

In contrast to interval arithmetic it will be shown that affine arithmetic has the desired behavior of tolerance cancellation. Furthermore, it is efficient regarding computation time. It has a linear increase in the number of tolerances (Table 1).

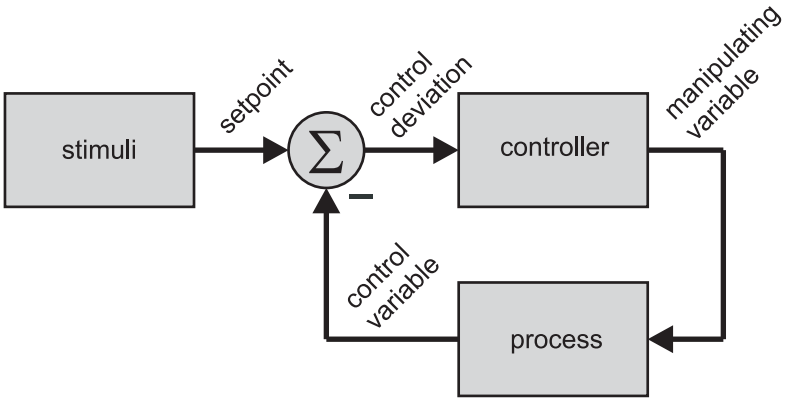

Figure 2: Typical control loop

For analyzing the behavior we simulate systems in a semi-symbolic way. This semi-symbolic simulation delivers "closed-form" results. As described above, interval arithmetic lacks the information which tolerances are correlated and thus always computes the worst-case to make sure that all possible results of the operation are in the resulting interval. An alternative to interval arithmetic is the affine arithmetic that keeps information about the correlation of the tolerances.

One application of affine arithmetic for electronic design automation is circuit sizing [LHB02]. Outside the area of electronic design automation most of the applications of the affine arithmetic are in finding boundaries of numerical errors. Other areas are algorithms for computer graphics, and global optimization. For all components that compute linear functions this arithmetic can be applied but it is also possible to define nonlinear operations. For a first evaluation we concentrate on linear operations. Thus it especially makes sense determining results of tolerances in signal processing applications.

3.1 Affine Arithmetic

Numbers are used in conventional arithmetic. In affine arithmetic [ACS94] the numbers are replaced by expressions of the form

$$\hat{x} = x_0 + \sum_{i=1}^{n} x_i \epsilon_i, \quad \epsilon_i \in [-1, 1].$$

The term x_0 is the nominal value like in conventional arithmetic. Each additional x_i represents the amount of tolerance introduced by a source (input signal or component) indexed with i. One i represents the same tolerance source throughout all variables. ϵ_i is a formal variable. x_i scales this error variable. This means that a certain value is never assigned to ϵ_i during the computation. The coefficient ϵ_i may take any fixed value in the interval [-1,1] and the condition $\epsilon_i \in [-1, 1]$ is a side condition for all the following formulas.

Computation is performed by replacing all elementary operations on numbers by the respective operations on affine forms. The mathematical operations on affine forms that are needed for this paper are defined by

$$\hat{x} \pm \hat{y} = (x_0 \pm y_0) + \sum_{i=1}^{n} (x_i \pm y_i)\epsilon_i$$

$$c\hat{x} = cx_0 + \sum_{i=1}^{n} cx_i \epsilon_i$$

$$d(\hat{x}) = 2\sum_{i=1}^{n} |x_i|$$

3.2 Modeling Tolerances

The following describes how to model tolerances for a simulation using affine arithmetic. Instead of single values, affine arithmetic expressions are used to describe the value of a signal at a point in time. The accuracy of a signal is represented by the sum of tolerance terms. Thus affine arithmetic is used for modeling accuracy/tolerances of analog and mixed-signal systems on the system level. The system consists of modules (components, blocks) and signals. Signals are connections between modules or between modules and inputs/outputs. The function of the modules is restricted to linear input/output behavior.

x_i represents the magnitude of tolerance on the signal x caused by a tolerance source i. In [ACS94] the x_i are called noise symbols. To prevent confusion with noise in the electrical sense we call them tolerance symbols. This tolerance can correlate with any arbitrary manufacturing tolerance, the temperature of the system or a component, etc. or it can be the quantization error of an AD- or DA-converter. For an efficient implementation, each of the error terms $x_i\epsilon_i$ is considered to be independent from each other $x_j\epsilon_j$. Modeling the influence of one error to several signals (e.g. common effects like chip temperature) is accomplished by using one common error variable ϵ_i in several signals to model the correlation. Note that for two different signals x and y with one common error variable ϵ_i, the magnitude of influence of this error variable can be scaled differently for $\hat{x}$ and $\hat{y}$ by x_i and y_i.

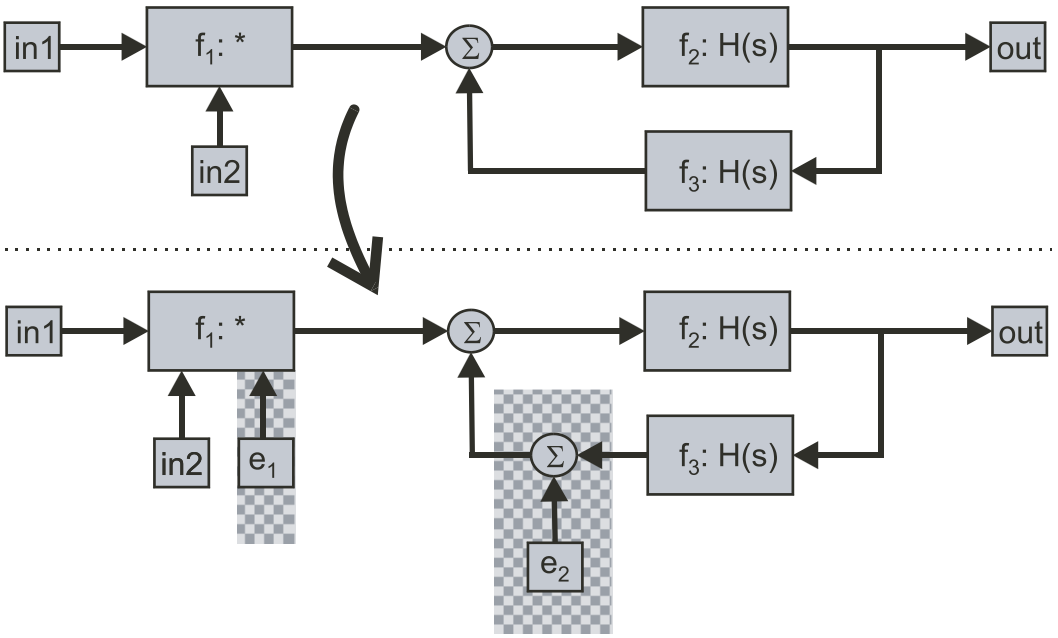

Figure 3: Introducing tolerances into the modeled system

Tolerances are modeled as additive errors at the inputs or outputs of components or in the signal path of modules. However, it is possible to keep track of round-off and truncation errors, numerical errors are not considered in this approach. The method is restricted to additive tolerances introduced at components or signals (Figure 3). Furthermore, correlations of higher order (like quadratic ones) are not considered.

The affine expressions give valuable hints for system design: It is possible to decide the amount of influence each source of tolerance has on any signal. This allows designers to decide which tolerance terms are most critical for system precision, and how much tolerance is accept-

affine form	diameter	interval	diameter
$\hat{x} = 17.3 + 2.5\epsilon_1$	5.0	$X = [14.8, 19.8]$	5.0
$\hat{y} = 15.4 + 2.5\epsilon_1$	5.0	$Y = [12.9, 17.9]$	5.0
$\hat{z} = 15.4 + 2.5\epsilon_2$	5.0	$Z = [12.9, 17.9]$	5.0
$\hat{x} - \hat{y} = 1.9 + 0.0\epsilon_1$	0.0	$X - Y = [-3.1, 6.9]$	10.0
$\hat{x} - \hat{z} = 1.9 + 2.5\epsilon_1 - 2.5\epsilon_2$	10.0	$X - Z = [-3.1, 6.9]$	10.0

Table 2: Affine Expressions and their interval counterparts

able. Thus it is possible to estimate if an output is precise enough to meet the specification. For example this information can help to dimension an AD-converter (how many useful bits are available at the input) or how much tolerance is introduced by the quantization caused by an AD- or DA-converter.

Example: A signal x implemented as an affine form would be written as $\hat{x}$ and the same implemented as an interval as X. Table 2 demonstrates the difference between affine arithmetic and interval arithmetic. The two signals $\hat{x}$ and $\hat{y}$ have the same amount of tolerance caused by the same tolerance source 1. This tolerance would cancel out in a subtraction, whereas the two signals $\hat{x}$ and $\hat{z}$ also have tolerance of the same magnitude, but caused by different tolerance sources 1 and 2. These tolerances in contrast should not cancel out, as they are uncorrelated. The behavior of affine arithmetic is as we expect it to happen. Tolerances caused by the same source can cancel out; tolerances caused by different sources will not cancel out (see the diameter). On the other hand interval arithmetic looses the information about the correlation and creates an over-approximation of the real range.

3.3 Implementation

The implementation is based on an early prototype of SystemC-AMS [VGE03] which supports the template concept. The affine arithmetic itself is implemented as a C++ object class called 'AAF' (affine arithmetic forms) [Gay03]. The concept of operator overloading provided by C++ is used so that equations can be written using the normal operators for addition, subtraction, constant multiplication, and constant division. This shows another advantage of SystemC. As there is full access to the C++ source code of the simulation engine, the models, the simulation setup, etc., it is a lot easier than with other simulation environments to integrate a new arithmetic.

The data structure consists of the nominal value, pointers to the coefficients and indexes, a variable that keeps track of the length of the respective affine form, and a static variable for the class that holds the highest index value j for the ϵ_j. If a new tolerance influence is added to the system, this gets ϵ_j and the variable for the highest index value is incremented by one. The class 'AAF' is given as a template parameter to ports, signals, and modules which would usually get the type 'double' as template parameter in conventional SystemC-AMS.

4 Case Study

To give an example a part of the source code of the control loop design (Figure 2) using affine arithmetic is listed below.

```
// the complete stimulator
```

```
SC_MODULE(stim_t_amb) {
  sc_out<SIMTYPE> out;
  Interval deviationInterval;
  AAF deviation;

  void behavior() {
    out = 50.+deviation;  // temperature in degrees celsius
  };

  SC_CTOR(stim_t_amb) {
    deviationInterval.modlohi(-1,1);
    deviation = deviationInterval;
    SC_THREAD(behavior);
  };
};
// end of the stimulator class

// a code snippet of the controller
sc_in<AAF> setpoint, control_variable;
sc_out<AAF> manipulating_variable;
double kp=3;
...
manipulating_variable=((AAF)setpoint-(AAF)control_variable)*kp;
```

The first part of the source code shows the stimulator class. The constructor is very simple. The first two lines create an AAF with an interval [-1,+1] and after that the SystemC-Thread "behavior" is notified to the simulation kernel. During the simulation the port called "out" will deliver a simple AAF for the setpoint $out = 50 + 1\epsilon_1$.

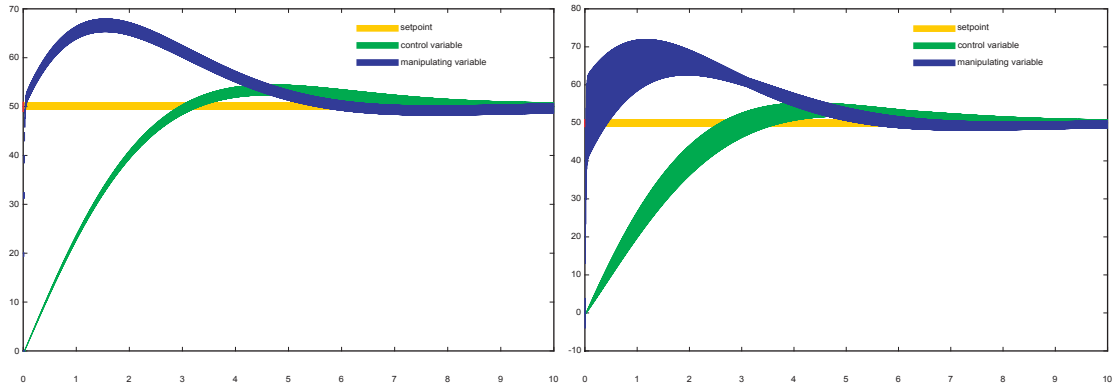

Figure 4: Control loop simulated with affine arithmetic

To give an example the plots in Figure 4 show temperature versus time at three different nodes in the control loop. The controller tries to change the manipulating variable so that the regulating variable will reach 50 ± 1 °C. The left plot shows the simulation result when a tolerance of ± 1 is introduced at the setpoint and the right one shows the same setup like the one for the left plot but with an additional tolerance of ± 10 introduced at the manipulating variable. The tolerance introduced at the setpoint is propagated to the other nodes as expected and the tolerance increases with the signal amplitude as expected but does not increase to $[-\infty, +\infty]$ over time like interval arithmetic would do. On the other hand the tolerance introduced at the control variable is not propagated but canceled after some time as you would expect it

to happen in a real control loop and the control variable converges as expected to the affine expression $50 + 1\epsilon_1$.

5 Discussion and Future Work

A concept for simulating tolerances on the system level was presented and compared to different other concepts regarding complexity and constraints (Table 1). It has been shown that affine arithmetic is feasible for linear systems also in the presence of feedback. Up to now, there are limitations. The most severe restriction of this method is that only linear functions for the components can be simulated. If the components are nonlinear the Monte-Carlo method is still a better method, yet. It simulates quite the physical way by looking only at one tolerance combination at a time. Furthermore, it works even if the function of the model is nonlinear, non-continuous, and not time invariant. On the other hand, many systems consist of components that are linear, continuous, and time invariant. Examples can be found in most signal processing and control systems. In contrast to Monte-Carlo methods with the presented method using the affine arithmetic there is no need to run the simulation a stochastically significant number of times to meet the confidence interval, because the result is an algebraic expression which describes the range of possible values completely. The other restriction is that the error model is, for now, restricted to static tolerances. On the other hand, there are dynamic tolerances (e.g. slew-rate limitations). Future work will, therefore, go into modeling also weak non-linearites like slightly bent characteristic curves and expanding the error model for further types of tolerances.

References

[ACS94] M.V.A. Andrade, J.L.D. Comba, and J. Stolfi. *Affine Arithmetic (Extended Abstract).* INTERVAL '94, St. Petersburg, Russia, 1994.

[AGW93] Kurt J. Antreich, Helmut E. Gräb, and Claudia U. Wieser. *Practical Methods for Worst-Case and Yield Analysis of Analog Integrated Circuits.* International Journal of High Speed Electronics and Systems, 4(3), pp. 261-282, 1993.

[Gay03] Olivier Gay. *Libaa - C++ Affine Arithmetic Library for GNU / Linux.* http://savannah.nongnu.org/projects/libaa, 2003.

[LHB02] Andreas Lemke, Lars Hedrich, and Erich Barke. *Analog Circuit Sizing Based on Formal Methods Using Affine Arithmetic.* ICCAD 2002, 2002.

[OGW02] Peter Oehler, Christoph Grimm, and Klaus Waldschmidt. *A Methodology for the Synthesis of Mixed-Signal Applications.* IEEE Transactions on VLSI Systems. IEEE Press, Vol. 10, Number 6, pp. 953-942, 2002.

[Roe55] Jürgen Roedel. *An Introduction to Analog Computors in A Palimpset on the Electronic Analog Art.* G.A. Philbrick Researches, Inc., Boston, Massachusets, U.S.A., 1955.

[VGE03] Alain Vachoux, Christoph Grimm, and Karsten Einwich. *Analog and Mixed-Signal Modeling with SystemC-AMS.* International Symposion on Circuits and Systems 2003 (ISCAS '03), Bangkok, Thailand, 2003.