Chapter 1

MODELING UNCERTAINTY IN NONLINEAR ANALOG SYSTEMS WITH AFFINE ARITHMETIC

Wilhelm Heupke[1], Christoph Grimm[2], and Klaus Waldschmidt[1]

[1]*Department of Computer Engineering*
University of Frankfurt

[2]*Institute for Microelectronic Systems*
University of Hannover

Abstract This paper describes a semi-symbolic method for the analysis of mixed signal systems. Aimed at control and signal processing applications, it delivers a superset of the set of all reachable values. The method that relies on affine arithmetic is precise for linear systems, but in the case of nonlinear systems, approximations are needed. As for each approximation a new term is added, the number of approximation terms increases during simulation and therefore slows down the simulation. This leads to a quadratic time complexity in the number of time steps. A method to avoid this and an example implementation based on SystemC-AMS together with its performance are presented.

Keywords: affine arithmetic, intervals, SystemC-AMS, simulation, uncertainty, tolerance

1. Introduction

Today's automotive, telecom, and ambient intelligence applications consist of sensors, actuators, analog and digital circuits, and a large portion of software. At the system level designers usually specify and model such applications by continuous-time block diagrams with directed signal flow. For the verification and analysis of such systems, most notably a transient simulation is used: Input stimuli are specified and the simulator computes the output signals. The transient simulation allows designers to get important insight into the behavior of the modeled system and provides a simple functional verification. However, within the design process of many of the above mentioned systems, much time is spent for analyzing the impact of uncertainties introduced by different realization variants:

- Static variations of operating conditions
  (e. g. production tolerances)
- Dynamic variations of operations conditions
  (e. g. temperature drift)
- Quantization and rounding in
  digital signal processing and analog/digital conversion
- Physical effects in analog circuits (e. g. noise).

One big problem is, that some deviations are compensated and do not have a large influence on some output of interest, while another deviation of same value will be amplified and thus violates the specification.

The established analog or mixed-signal simulators at the electrical level provide different methods that help designers to evaluate the impact of deviations: noise analysis, sensitivity analysis, worst case analysis, Monte Carlo analysis, AC analysis, and sometimes combinations thereof. These analyses are either based on the fact that analog circuits have a working point and can be linearized (AC analysis, noise analysis), require monotonicity (sensitivity analysis), or use a very high number of simulation runs to find corner cases (worst case simulation) and to compute statistical properties at the outputs (Monte Carlo analysis).

Although these analyses are very useful, they have several drawbacks: Methods based on linearizations are usually restricted to analog circuits and are not applicable to mixed-signal systems or even DSP software. In order to overcome these problems designers have to provide discrete models and additional models that are used for AC analysis. Furthermore, time domain simulations are used in combination with FFT methods to get information about the spectral distribution of noise, for example. Unfortunately, transient simulations and Monte Carlo methods do not provide information about the contribution of single sources of uncertainty to the total uncertainty at e. g. outputs in a direct and easy way; usually the interpretation is rather difficult. A direct and easy interpretation is of particular interest for the case of design automation at system level.

The above mentioned classical analyses are aimed towards the electrical level and are based on linearization and linear equation solvers. The method proposed in this paper is intended for a system level simulation with a discrete time static dataflow model of computation, which is implied by the use of SystemC-AMS. One should be aware that there is no automatic way to use the electrical level models at the system level or vice versa, yet.

Compared with purely numerical simulation, the symbolic or formal techniques provide designers with more information, e. g. about the origin of a deviation [Henzinger and Ho, 1995; Zhang and Mackworth, 1996; Henzinger, 1996; Hartong et al., 2002]. Unfortunately, symbolic or formal techniques are far away from being applicable to the design of complex and heterogeneous systems [Stauner et al., 1997]. In this situation, semi-symbolic techniques are very attractive if they combine advantages of symbolic techniques with the general applicability of simulation. A promising approach is the use of affine arithmetic [Andrade et al., 1994] for semi-symbolic analysis [Fang et al., 2003; Lemke et al., 2002] or even a semi-symbolic simulation [Heupke et al., 2003].

Fang and Rutenbar [Fang et al., 2003] are doing a static analysis of rounding errors in DSP algorithms with affine arithmetic. In [Heupke et al., 2003; Grimm et al., 2004a] we use affine arithmetic for semi-symbolic transient simulations of complex signal processing systems. The simulation result is a numerical output together with a symbolic, affine approximation of the contributions of different (symbolic) sources of uncertainty. An important advantage of the proposed method is the safe inclusion of all reachable values by the affine expression, therefore delivering reliable results. On the other hand the increasing number of terms and the resulting over-approximation caused by each nonlinear operation are a disadvantage. In this paper we introduce a method for semi-symbolic simulation with affine arithmetic that efficiently handles these approximation terms.

Section 2 gives a brief introduction to affine arithmetic and semi-symbolic simulation with affine arithmetic as described in [Heupke et al., 2003]. Section 3 introduces a method to handle the over-approximation terms in semi-symbolic simulation based on affine arithmetic. This enables affine arithmetic to reach the same asymptotic time complexity conventional numerical simulation has. Section 4 demonstrates the applicability of the method by a simple example.

Figure 1.1 shows a system we implemented as a test in order to validate the behavior by transient simulations with affine expressions as data type.

It contains two elements that can be troublesome:

The first is feedback. Another range arithmetic, the interval arithmetic, will not deliver a meaningful result for the simulation of systems

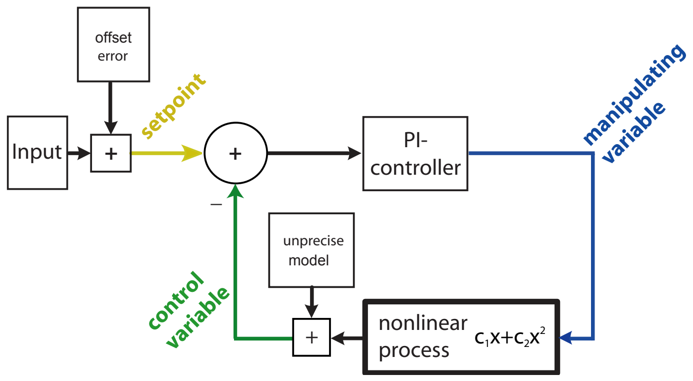

Figure 1.1. Simulated System with Nonlinear Block

with feedback, while affine arithmetic works fine in this respect [Heupke et al., 2003].

The second is the emphasized nonlinear block in the system, which is interesting, as it creates additional approximation terms in each iteration through the loop. This results in a high number of terms, that would slow down the simulation more and more, if nothing is done about that.

2. Semi-Symbolic Simulation with Affine Arithmetic

Affine Arithmetic

Affine arithmetic [Andrade et al., 1994], introduced by Comba et al., is a kind of improved interval arithmetic, and therefore allows us to compute with uncertain values. In each affine expression the influence of independent sources of uncertainty i to a variable $\hat{x}$ with the central value x_0 is represented by a symbolic sum of terms $x_i \epsilon_i$. Noise symbols ϵ_i represent arbitrary, but for one simulation run fixed values from the interval $[-1, 1]$. The partial deviations x_i then scale these intervals. The ϵ_i are a symbolic representation and a certain value is never assigned to them.

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

Basic mathematical operations are defined by

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

and

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

The operation $\mathrm{rad}(\hat{x})$ is the radius of the affine expression $\hat{x}$.

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

The results of linear operations give precise limits and have no over-approximation (no unnecessary expansion of the error interval).

Note that the subtraction of two affine expressions which include the same noise symbols ϵ_m may reduce the partial deviation of the result, in contrast to the same values with different noise symbols. This allows to model error cancellation, for example in feedback loops. In Table 1.1 the variables with a hat denote affine arithmetic variables while the ones written with a capital letter are corresponding interval variables. The diameter is obviously twice the radius for affine forms. In the case of intervals the diameter is the difference between supremum and infimum of the interval.

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 1.1. Affine Expressions and their interval counterparts

Table 1.1 shows the difference between affine arithmetic and interval arithmetic in the case of different or same source of uncertainty. The variables x and y share an uncertainty caused by the same source of uncertainty and therefore both have a term ϵ_1. For demonstration purposes also the influence of this uncertainty is of same magnitude and direction/sign. In contrast to that the variable z has a term ϵ_2 that shows that the uncertainty of z has a different source of uncertainty, although both have the same magnitude in example given in table 1.1. The effect shows up in the subtraction of x-y. Interval arithmetic increases the interval diameter instead of bringing it to zero, while affine arithmetic keeps the correlation and delivers the precise result. This is because the information contained in interval arithmetic is too limited, as the range of values is not the only important information that is needed to describe the influence of uncertainty.

This effect of interval arithmetic may be tolerated sometimes, but a simulation of a control loop, where a too pessimistic result is fed back over and over, results in a permanently increasing diameter and depending on the system will increase exponentially in the worst case. This will deliver with interval arithmetic that the result is $[-\infty, +\infty]$ within a limited number of simulation time steps [Heupke et al., 2003]. For sure this is a safe inclusion, but would be useless pessimistic.

The concept in the form described in this paper can be extended to dynamic uncertainties and therefore to analyze effects like colored noise as described in [Grimm et al., 2004b].

An important aspect is the guarantee, that after each operation, the result is a superset of all reachable values.

Therefore, for example, multiplication of two affine expressions is defined by

$$\begin{aligned}\hat{x} \cdot \hat{y} \quad := \quad & x_0 \cdot y_0 + \sum_{i=1}^{m} (x_0 \cdot y_i + x_i \cdot y_0)\epsilon_i \\ & + \mathrm{rad}(\hat{x}) \cdot \mathrm{rad}(\hat{y}) \cdot \epsilon_{m+1} \quad .\end{aligned}$$

In general, the error introduced by some nonlinear operation is over-approximated by a new noise symbol ϵ_{m+1}.

All nonlinear operations introduce new noise symbols and therefore some systems may present a problem, because of the permanently increasing number of terms. But some systems include strategies to reduce the influence of deviations (e. g. feedback). Caused by these strategies, the influence of these noise symbols converges to zero and for a stable system they are absolutely summable. Section 3 describes how this property delivers a solution for the problem of the increasing number of terms.

SystemC-AMS based Implementation

For the implementation we chose SystemC-AMS, but the concept mentioned below can be implemented in every language that supports operator overloading, e. g. VHDL-AMS.

SystemC-AMS is an extension of the class library SystemC, aimed at supporting the modeling of mixed-signal (analog and digital) systems. It provides means to simulate analog, mixed-signal and signal processing systems as a block diagram. Regarding the abstraction level it is comparable to Matlab. In contrast SystemC allows us to immediately reuse the code portion for these blocks, that have to be implemented in Software later on. Additionally the code of the models, that will be implemented in digital hardware, can be automatically synthesized to create e.g. an ASIC or FPGA implementation. Only for the blocks, that model analog behavior, there is no automatic way for implementation. These blocks are specified by transfer functions or static nonlinear functions implemented in C++. Static dataflow is used as the model of computation.

The implementation of the affine arithmetic is based on the libaa library[Gay, 2003] which defines linear and nonlinear operations on affine arithmetic variables in a class called `AAF` (affine arithmetic forms). It allows us to model computations with uncertainties in general.

Using the `AAF` class with SystemC-AMS is very simple: In SystemC-AMS, as in SystemC, signals are instantiated with a template parameter `T`. This parameter specifies the value type of the signal. For example by `sca_sdf_signal<double> my_signal` a signal with a value range of a variable with double precision is instantiated in SystemC-AMS. Of course, one can as well specify the template parameter `AAF` instead of `double`. This small change is all that is needed with SystemC-AMS to turn the numerical simulation into a semi-symbolic simulation based on affine arithmetic. Instead of using operators defined for `double` values, the compiler will use the operators defined in the `AAF` class, which overload the standard operators. The results of the simulation are affine expressions that semi-symbolically represent possible deviations.

For example, one can write the following code:

```
AAF a(2.0), b(3.0), c(2.0), y;

// constructor which adds a noise symbol
// x_i with partial deviation 0.1:
AAF uncertainty(Interval(-0.1, +0.1));
a = a + uncertainty;
y = (a + b) * (c + uncertainty);
cout << "y = " << y << endl;
```

This simple program produces the following output:

```
y = 10 + (e1*0.7) + (e2*0.01)
```

So after the uncertainty is introduced one can use a variable of type `AAF` like any other variable.

The advantage of semi-symbolic simulation compared with a numerical simulation becomes obvious if the uncertainties at the output of the simulated system exceed some specified range. In this case, the symbolic representation provides designers with the contribution of all sources of uncertainty to the deviation at the output. It also models the effects that are created by the combination of uncertainties. This together allows the designer to identify sources of uncertainty where improvements are most fruitful. As a long term perspective one day a mixed-signal synthesis system can be directed this way, where further optimization is needed.

3. Handling nonlinearity

Each nonlinear operation approximation creates an additional term, as can be seen in the code example. These approximations are a problem for the affine arithmetic, as potentially a high number of very small and thereby insignificant terms in the symbolic expression is created.

This problems shows up especially if the system that is modeled contains a loop and this system has at least one component that creates an approximation in the path of this loop (e. g. by multiplication of two affine expressions).

Then any kind of memory (e. g. some modeled energy storage) in a block within the modeled system will contain most of the approximation terms that are created in each simulation cycle of this loop. If there is a constant number of approximations this means that in each simulation cycle the number of terms increases by this constant number.

To cope with that, we introduce a method that 'cleans up' the affine arithmetic expression variables. It somewhat resembles the garbage collection concept, used to free unneeded memory of variables, that is used in some programming languages like e. g. Java.

If the number of noise symbols in the affine expressions increases above a certain level, the `simplification()` method is called. For all variables in the system, all terms smaller than a cut level l, set by the user, are summed up separately by the ones with a positive and the ones with a negative sign to two special noise terms.

Deleting the terms with an absolute value below this cut level could potentially lead to inaccurate results in the case of a high number of simulation time steps and certain functional blocks, e. g. integrators, because in this example they may grow to a big one over time. Therefore it is better to sum them. This way it delivers a safe inclusion, but it means that the correlation of the individual terms is lost. But it does not exhibit the same problem like interval arithmetic does, as described above, because the correlation of this sum is still valid for all `AAF` variables in the future time steps and the terms with different signs are kept separate. Furthermore these uncertainties are usually far smaller than the nominal values and if l again is far smaller than the other uncertainties, any kind of over-approximation would not create a problem. So the influence of approximations decreases below the level l after several time steps.

Please note that if the simplification method would be called too often, the unneeded over-approximation could potentially show up significantly and in the above mentioned example the concept of feedback that makes these terms converge to 0, would not work. On the other hand if called not often enough, the computation time will increase significantly. Our experiences has shown, that the choice of the time point, when to call the simplification method, was for the example system not very critical.

The method resembles the typical strategy of leaving away smaller terms. But with affine arithmetic we do not have to really leave the smaller terms away, instead we can handle their sum as a new uncertainty. So not only the modeled uncertainties of the real system, but obviously also uncertainties caused by the modeling process, like these simplifications, are analyzed.

Implementation of the simplification method

In the present implementation the simplification method is invoked every 1.000th simulation cycle, but later on it might be automatically invoked by some heuristic. For example the change in the highest index of the noise terms since the last simplification could be used as a criterion, when to call this method. The cut level l is set to a constant small fraction of the smallest explicitly introduced uncertainty by the user.

The change in an affine expression can be seen by the following example of a simplification with $l = 1.0 \cdot 10^{-4}$. First a variable was printed immediately before the simplification:

```
28.9796 + (e1*2.9925) + (e5*0.000856951)
+ (e6*1.14971e-006) + (e7*1.11085e-006)
+ (e8*-1.34821e-007) + (e9*1.07968e-006)
+ (e10*-1.12145e-007)
```

After the simplification the printed variable changed to:

```
28.9796 + (e1*2.9925) + (e5*0.000856951)
 + (e34*3.34024e-006) + (e35*-2.46966e-007)
```

Usually this happens with far more terms, but for demonstration purposes it would be difficult to show. In this case ϵ_{34} sums up the positive insignificant terms and ϵ_{35} sums up the negative insignificant terms.

By handling a list with pointers to all affine variables in the system, it is possible to access all `AAF` variables. This list is added as a static member of the `AAF` class, so that all `AAF` variables share it.

The `AAF` class saves the affine expression in one variable for the central value x_0 and two pointers to dynamically allocated arrays called `coefficients` and `indices`. In a first run across all `AAF` variables and across all coefficients in these variables, the significant terms are collected, based on the cut level l. A term x_i of an affine variable $\hat{x}$ is significant if it fulfills the condition

$$|x_i| > l.$$

The second run goes again across all variables. For each of the variables it is determined how many significant terms are contained, based on the result of the first run. Then two new arrays for the coefficients and the indices are allocated and the significant terms are copied to the new arrays. After that the memory of the old arrays is freed.

Efficiency gained by the simplification

The following text analyses the effort to handle one variable. So the total effort also scales with the number of variables for all similar simulation methods.

Let us assume a system with a loop, n be the number of total simulation time steps and c be a constant that describes the maximum number of nonlinear operations, along the path of the loop. Remember that these nonlinear operations add terms. Further let k be the number of explicitly introduced uncertainties.

With conventional simulation based on the static dataflow model of computation and with variables of type e. g. `double` the space complexity is $O(1)$ and the simulation time is $O(n)$.

In contrast to that in the naive implementation the maximum memory needed for each affine variable is

$$cn + k \subseteq O(n)$$

because in each of the n steps c uncertainties are added by overapproximations and a maximum of k has been added intentionally at the elaboration phase. This means that the space complexity is $O(n)$.

Even worse is the resulting time complexity. This is because in each simulation time step each term of an affine variable needs to be handled, e. g. an arithmetic operation has to be performed for it by the CPU:

$$\begin{aligned} \sum_{i=1}^{n}(ci + k) &= cn(n+1)/2 + kn \\ &= n^2/2 + (c+k)n \\ &\subseteq O(n^2) \end{aligned}$$

System theory requires for every stable system that every bounded input delivers a bounded output. Obviously every technically meaningful system to be implemented is stable. Furthermore a discrete system is stable if and only if the impulse response is absolutely summable:

$$\sum_{i=-\infty}^{\infty} |h(i)| < \infty$$

This implies an important aspect: The impulse response of the opened control loop converges to zero. So every introduced over-approximation term will converge to zero with the number of iterations through the control loop in the given example. This means that we can apply a trick that copes with the terms caused by the over-approximation: From time to time we sum up all approximation terms that got extremely small (smaller than l) by a simplification method, thereby keeping the safe inclusion, but reducing the number of terms. On the other hand, this means, if the number of terms can not be reduced, we get a strong indication that the system might be unstable.

This simplification method, if called every m simulation time steps, is a substantial step forward regarding efficiency, because in the m time steps between two simplification operations, a maximum of c terms adds in each time step. To this adds the number of k explicitly introduced terms. As c, m and k are all constants, we get asymptotically the same space complexity like pure numerical simulation:

$$cm + k \subseteq O(1)$$

The time complexity of the simulation with the simplification method needs $cm+k$ computations in one simulation time step in the worst-case of the time step before the next simplification method call. This happens in the worst-case n times. To this adds the effort of the simplification method, called n/m times. The simplification method itself needs in the first and the second pass to touch every term. This gives a total time complexity of $O(n)$, also the same complexity numerical simulation has:

$$\begin{aligned} & (cm + k)n + 2n/m(cm + k) \\ = \; & (cm + k + 2(cm + k)/m)n \\ \subseteq \; & O(n) \end{aligned}$$

4. Experimental Results

The system shown in Figure 1.1 was implemented in SystemC-AMS and the `AAF` class. With this setup transient simulation runs were performed.

number of time steps	**computation time without simplification [s]**	**computation time with simplification [s]**
1,000	1	1
2,000	4	2
4,000	16	5
8,000	61	10
16,000	244	20
32,000	999	40
64,000	4083	79
128,000	–	159
256,000	–	319
512,000	–	640
1,024,000	–	1275

Table 1.2. Measured Computation Time

Table 1.2 shows the time needed for the simulation with and without using the simplification method. The time interval that was simulated was the same for all values in the table and was scaled to deliver time results that are easy to interpret. Only the step width in time was changed for each row. The simplification method was called every 1,000th time step, respectively never in the case of no simplification.

The table shows the clear advantage of the simplification method, as the computation time increases linear with the number of simulated time steps, if the simplification method is used. It is very clear to see a quadratic increase of the needed computation time for the simulation without the use of the simplification method, that shows up as a four fold increase of the required computation time for a two fold increase in the number of time steps. So it gets obvious, that affine arithmetic would be much harder to use without this simplification method for long simulation runs in the presence of feedback and nonlinearity.

For a visual representation, we convert affine expressions to intervals, by use of the `rad` operator. These intervals can be plotted as shown in Figure 1.2 as a range. In the case of an uncertainty that is substantially smaller than the central value, two separate traces with different scalings are plotted. We use for both types of plots the program gnuplot, as usual waveform viewers do not support interval type signals. Figure 1.2 shows the step response of a feedback loop that contains a nonlinear control path, which is shown in Figure 1.1.

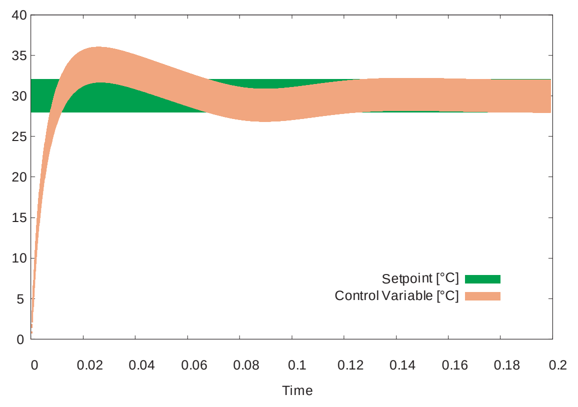

Figure 1.2. System within the stable area

Figure 1.3 shows the step response close to the stability border and Figure 1.4 the same system, but beyond the stability border. Interesting to note are typical chaotic effects of nonlinear systems near the stability border, that show up very clearly in the uncertainty and which are not linear with the central value in Figure 1.3.

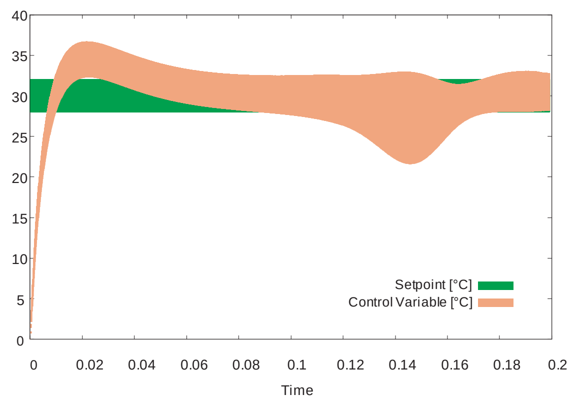

Figure 1.3. System near the stability border

5. Conclusion

Without the described method semi-symbolic simulation with affine arithmetic has quadratic time complexity. On the other hand, with the presented method, simulation with affine arithmetic has linear time complexity, even in the presence of nonlinearities and feedback. This

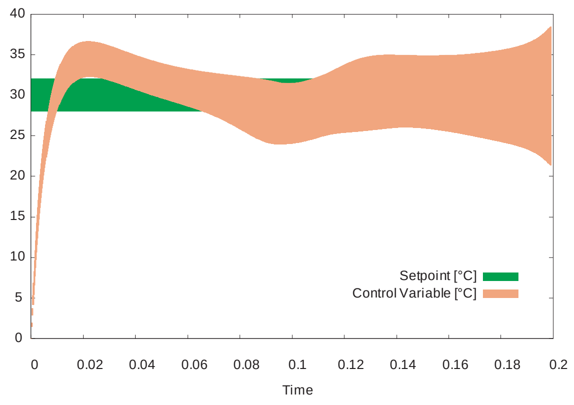

Figure 1.4. System beyond the stability border

means that affine arithmetic is feasible for simulation even with a large number of time steps in nonlinear feedback systems.

Compared with purely numerical simulation these extensions allow designers to analyze the noise and sensitivity to different sources of uncertainty, such as thermal or quantization noise. Compared with analyses in ‘analog’ simulators, the described method is applicable to digital signal processing methods and to discrete time approximations of analog circuits. This allows designers an analysis of heterogeneous systems that include large fractions of DSP software. The symbolic representation of the contributions to the deviations at the outputs can be interpreted easily and delivers a safe inclusion, an important aspect for the design of security critical systems, that could create otherwise dangerous situations if their deviation is too large.

Compared with formal approaches for model checking or property refinement of hybrid systems [Henzinger and Ho, 1995; Henzinger, 1996; Heupke et al., 2003] it allows us to model and verify properties such as robustness or precision, that are key issues in the design of analog and mixed-signal systems.

References

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

Fang, C.F., Rutenbar, R.A., Püschel, M., and Chen, T. (2003). Towards Efficient Static Analysis of Finite-Precision Effects in DSP Applications via Affine Arithmetic Modeling. In *Design Automation Conference (DAC 2003)*, Anaheim, USA.

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

Grimm, Christoph, Heupke, Wilhelm, and Waldschmidt, Klaus (2004a). Refinement of Mixed-Signal Systems with Affine Arithmetic. In *Design, Automation and Test in Europe 2004 (DATE '04)*, Paris, France.

Grimm, Christoph, Heupke, Wilhelm, and Waldschmidt, Klaus (2004b). Semi-Symbolic Modeling and Analysis of Noise in Heterogeneous Systems. In *Forum on Specification and Design Languages (FDL '04)*, Lille, France.

Hartong, Walter, Hedrich, Lars, and Barke, Erich (2002). Model Checking Algorithms for Analog Verification. In *Design Automation Conference '02 (DAC 2002)*, New Orleans, Louisiana.

Henzinger, Thomas A. (1996). The theory of hybrid Automata. *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*, pages 278–292.

Henzinger, Thomas A. and Ho, Pei-Hsin (1995). Hytech: The cornell hybrid technology tool. In Antsaklis, Panos, Kohn, Wolf, Nerode, Anil, and Sastry, Shankar, editors, *Hybrid Systems II*, volume 999 of *Lecture Notes on Computer Science*, pages 265–293. Springer, Berlin.

Heupke, Wilhelm, Grimm, Christoph, and Waldschmidt, Klaus (2003). A New Method for Modeling and Analysis of Accuracy and Tolerances in Mixed-Signal Systems. In *Proceedings of the Forum on Design Languages (FDL'03)*, Frankfurt, Germany.

Lemke, Andreas, Hedrich, Lars, and Barke, Erich (2002). Analog Circuit Sizing Based on Formal Methods Using Affine Arithmetic. In *ICCAD 2002*.

Stauner, Thomas, Müller, Olaf, and Fuchs, Max (1997). Using HyTech to Verify an Automotive Control System. In Maler, Oded, editor, *Hybrid and Real-Time Systems — International Workshop, HART '97*, volume 1201 of *Lecture Notes on Computer Science*, pages 139–153. Springer, Berlin.

Zhang, Ying and Mackworth, Alan K. (1996). Specification and verification of hybrid dynamic systems with timed ∀-automata. In Alur, Rajeev, Henzinger, Thomas A., and Sontag, Eduardo D., editors, *Hybrid Systems III*, volume 1066, pages 587–603. Springer, Berlin.