Refinement of Mixed-Signal Systems with Affine Arithmetic

Ch. Grimm, W. Heupke, K. Waldschmidt
University Frankfurt, Professur Technische Informatik
Robert-Mayer-Strasse 11-15; 60054 Frankfurt, Germany
grimm|heupke|waldsch@ti.informatik.uni-frankfurt.de

Abstract

This paper describes a framework for the refinement of control and signal processing functions. The design starts with an executable specification, and allowed deviations thereof. Refinement steps introduce models of analog or digital implementations, and augment the 'ideal' behavior with different sources of uncertainty. The framework verifies and analyzes the influence of these uncertainties on system properties using affine arithmetic.

1. Introduction

Today embedded and ambient intelligence systems use complex mixed-signal circuits to realize control and signal processing functions. Such functions are specified using continuous-time block diagrams. Simulators like Matlab/Simulink, or even model checking techniques (e.g.[7]) permit verification of continuous-time specifications. However, a number of design steps modify the system's behavior on the way to a mixed-signal realization, e.g.:

- determination of quantization and sample frequencies,
- digital filter or controller synthesis,
- analog signal processing with noise and tolerances,
- small nonlinearities and limitation of realizations.

Whether these design steps have an impact on the system's behavior is validated by a number of simulation runs [5]. A more complete verification would be possible with equivalence checking. Unfortunately, equivalence checking is not yet applicable to complex and heterogeneous systems.

Between formal verification and simulation is a wide field for semi-formal and symbolic techniques. For the analysis of signal processing and mixed-signal systems, property refinement, and analysis with affine arithmetics are new, promising approaches.

Property refinement [2, 6, 10, 11] is a design methodology. Behavior is specified by a non-deterministic relation $B \subseteq (I \rightarrow O)$ which maps inputs I to outputs O. Property refinement transforms a specification to another representation with behavior $B^R \subseteq B$. Therefore safety properties are preserved. However, tools for modeling typical 'errors' of mixed-signal circuits are missing up to now [11].

Symbolic analysis with affine arithmetic [3, 8, 9] is used for static analysis of floating point errors of DSP algorithms [3], or for circuit sizing [9]. In [8] we demonstrated the more general applicability for transient simulation of signal processing systems. An essential problem in these approaches is the lack of a methodology for specification and verification.

A combination of (semi-symbolic) simulation techniques with property refinement could be desirable for designers if it provides both a seamless methodology that ensures correctness from system level down to circuit design, and a powerful, meaningful and easy-to-use framework for semi-symbolic simulation. In the following we describe a methodology and a framework for the specification, refinement and verification of signal processing systems based on such a combination.

The idea is as follows: The design starts with a specification of the 'ideal' behavior, and a specification of allowed deviations thereof (tolerances). Design steps add sources of uncertainty, e.g. due to chosen quantizations, to the executable specification. The presented framework permits

- to verify that the resulting behavior of a design is within the specified tolerances.
- to analyze the influence of different sources of uncertainty (e.g. noise) to the output's uncertainty.

Within the framework both tolerances of specifications, and uncertainties of circuits are modeled using affine expressions.

Section 2 introduces a methodology for specification of signal processing systems, and for modeling of possible realizations with affine arithmetic. Section 3 describes the implemented framework for analysis and verification. Section 4 illustrates the applicability by a case study.

2. Specification and Modeling with Affine Arithmetic

For the specification of the intended behavior, and for modeling analog or digital implementations thereof we use non-deterministic functions $B \subseteq (I \rightarrow O)$. Non determinism allows us

1. to specify behavior with the freedom to choose implementations with slightly different behavior, and
2. to model parasitic effects of analog components or numerical errors of digital implementations.

We model non-deterministic functions by affine arithmetic.

2.1. Affine Arithmetic

Affine arithmetics [1] is a semi-symbolic technique that allows us to compute with uncertain values. In each affine expression the influence of independent sources of uncertainty i to a size with the 'ideal', central value x_0 is represented by a sum of terms $x_i\varepsilon_i$. Noise symbols ε_i are unknown values from the interval $[-1,1]$, and partial deviations x_i scale the sources of uncertainty:

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

Basic mathematical operations are defined by:

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

and

$$c\hat{x} = cx_0 + \sum_{i=0}^{n} cx_i\varepsilon_i$$

Results for the above linear operations give formally precise limits, and have no over-approximation (that means, we do not expand the error interval more than necessary). Non-linear operations can be defined which are formally precise, but which usually introduce over-approximation. Whether this over-approximation is a limitation depends on the application.

2.2. Specification and Tolerances

Non-deterministic functions can be difficult to understand. In order to have a specification that can be interpreted easily we explicitly distinguish the specification of ideal, deterministic behavior, and the specification of allowed deviations (tolerances) thereof. The ideal behavior is specified by a block diagram with continuous-time semantics (figure 1 left) and computes a deterministic output $o(t) \in O$ from inputs $i(t) \in I$.

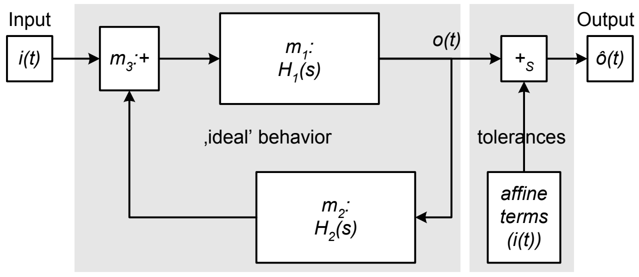


Figure 1. Specification of ideal behavior and tolerances.

Allowed deviations are specified by tolerance schemes for *characterizing inputs* such as impulse or step responses in different operating conditions. For linear systems the tolerance scheme of a non-deterministic impulse response fully specifies the allowed behavior. For non-linear systems a characterizing input is required at each operating point. Such tolerance schemes are known and accepted by designers, and we only formalize specification of tolerances using affine expressions.

We model tolerances as a sum of delayed affine terms which add an uncertainty to the deterministic output $o(t)$. The non-deterministic output $\hat{o}(t) \subseteq O$ is obtained by sampling $o(t)$ at discrete points in time t_i, and adding tolerances to these values:

$$\hat{o}(t) = \sum_{i=0}^{n} (\varepsilon_i o_i + o(t_i) + o_{off,i}) \mathrm{rect}(t_i, t_{i+1})$$

The tolerances consist of an offset $o_{off,i}$ and a noise term $\varepsilon_i o_i$. The resulting expression is multiplied with a function $\mathrm{rect}(t_1,t_2)$ which we assume to be a function which is 1 between t_1 and t_2, and 0 else. Figure 2 shows an example for a step response.

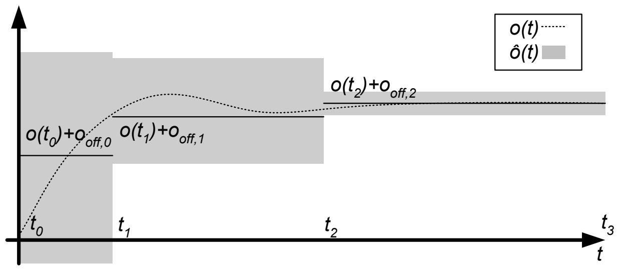


Figure 2. Example for step response.

Note, that one can compute the non-deterministic impulse response from a non-deterministic step response (e.g.

for control applications) by derivation, or from a tolerance scheme in the frequency domain (e.g. telecom applications) by Fourier transformation.

2.3. Modeling Implementation and Uncertainties

As an implementation we consider mixed-signal systems which typically consist of analog signal processing components, signal converters, and digital signal processing. Design steps replace blocks with ideal behavior by

1. models of an implementation, e.g. a discrete-time filter, or an analog circuit.
2. a specification of ideal behavior with tolerances, as described in section 2.2.

In both cases we use affine expressions to model deviations from the ideal behavior. Figure 3 shows an example of such a model. In this model the specification shown in figure 1 is refined by replacing the blocks m_1, m_2, m_3 with ideal behavior by implementations which introduce uncertainties and deviations from the ideal behavior.

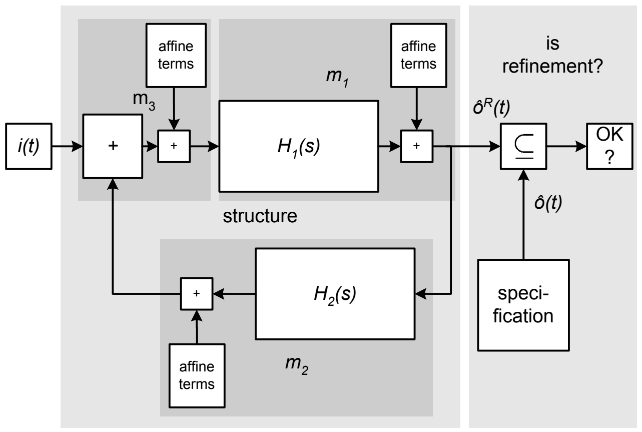


Figure 3. Example for a model of an implementation.

Uncertainties and deviations from ideal behavior are modeled by adding affine terms (noise symbol and partial deviation). We distinguish static uncertainties such as tolerances for components, and dynamic uncertainties such as noise. Static uncertainties add a constant deviation to a signal which is the same for all points in time. Dynamic uncertainties add a random value to a signal which can be different for each point in time. In this case we use noise symbols $\varepsilon[n]$ with index to access a noise symbol at a given point in time n. Single dynamic noise symbols $\varepsilon[n]$ from which only the j last values are needed can be mapped to $j+1$ static noise symbols $\varepsilon_x \dots \varepsilon_{x+j+1}$.

Furthermore, we introduce uncertainties that are Gauss distributions. In order to distinguish different reasons of uncertainty we use different noise symbols: ε for the common interpretation as an interval, and γ for Gaussian distribution. With such noise terms the sum is computed by:

$$\hat{x}+\hat{y} = (x_0+y_0)+\sum_{i=1}^{n}\gamma_i\sqrt{x_i^2+y_i^2}$$

In the following we discuss methods for modeling typical uncertainties of non-ideal implementations by adding affine terms to a signal $\hat{y} = y_0 + \sum_{i=0}^{m}\varepsilon_i y_i$.

Production tolerances: Analog implementations often have a static deviation $\pm e$ from the ideal behavior. This is modeled by adding a term $\varepsilon_{m+1}e$:

$$\mathrm{tol}(\hat{y},e) = \hat{y}+\varepsilon_{m+1}e$$

Quantization: Quantization adds a noise symbol γ_{m+1} with a partial deviation of a half quantization unit $Q/2$, which models the worst case deviation:

$$\mathrm{quant}(\hat{y},Q) = \hat{y}+\varepsilon_{m+1}[n]Q/2$$

Truncation of numerical operations, such as multiplication can be handled in the same way.

Nonlinearities: Non-linear operations such as $\hat{y}*\hat{x}$ can add new noise symbols that model the error. However, this can lead to an explosion of the number of symbolic terms, and to an increasing estimated error, because error cancellation can not be detected. We prefer a linear over estimation of the noise as follows which does not increase the number of terms:

$$\hat{y}*\hat{x} = (y_0*x_0)+\sum_{i=1}^{m}\varepsilon_i[n]p(y_i+x_i)$$

where p is a lower border for the derivation of $x*y$ in the interval covered by $\hat{x}$ and $\hat{y}$. Note, that this is formally exact but that we, depending on the nonlinearity, expand the interval more than necessary (over-approximation).

Noise: Noise is a dynamic uncertainty. A simple model of noise is to use the standard deviation as partial deviation x_i. A simple model for white noise is a sequence of statistical independent values with standard deviation σ:

$$\mathrm{noise}(\hat{y},\sigma) = \hat{y}+\gamma_1[n]\sigma$$

Colored noise is generated by filtering white noise in a FIR filter as follows:

$$\hat{y}[n] = y_0[n]+\sum_{i=0}^{m-1}\gamma_1[n-i]c_i y_1[n-i]$$

Note that each past value of y_1 has its own noise symbol $\gamma_1[n-i]$.

Numerical error: For verifying analog components or systems, simulation is used. Analog simulators compute approximations of a solution of differential and algebraic equations. The local truncation error (LTE) is added to the ideal behavior in each time step. For block diagrams this can be modeled by adding an affine term with a lower bound of LTE in each time step to each delayless loop. The analysis of numerical errors from simulation can be helpful to get confidence into simulation results.

The above effects are just examples or simple templates provided by our framework. Of course one can add new models of errors or noise which are more appropriate for a given application.

A restriction is the number of affine terms. In general, models must ensure that the number of affine terms does not increase significantly in a simulation run by: 1.) Restricting access to dynamic noise terms to few past values, 2.) Restricting the semi-symbolic simulation to a bounded time, if the number of terms increases with time steps. Another potential problem is over estimation. Most notably in recursive computations such as control loops over estimation of the errors can lead to either an exponential explosion of the error estimations, or to an explosion of the number of error terms.

Both problems cannot be avoided in general by this methodology. It is a task of the modeling process to formulate appropriate models. However, the design example in section 4 demonstrates that these problems are not prohibitive, and can be avoided easily.

3. A Framework for Refinement and Semi-Symbolic Simulation

In order to verify that an implementation with behavior B_R is a refinement of a specification B we show that possible outputs are within the specified tolerances O_t. We restrict this verification to the characterizing input signals I_c from the specification. For these input signals we compute output signals $O_c = B_R(I_c)$ of the implementation by semi-symbolic simulation with affine arithmetic. A design is a refinement of the specification, if $O_c \subseteq O_t$ which can be seen directly in plots of the tolerances O_t and the outputs O_c.

Implementation For semi-symbolic simulation and verification we use a transient simulation run where data types such as bit vectors or floating point numbers are replaced by affine expressions. This can be implemented by

- A simulator that is available in source code, that supports templates, and that can simulate a given model.
- An abstract data type ‘affine expression’ which provides an efficient internal representation and which defines operations used in a given model.

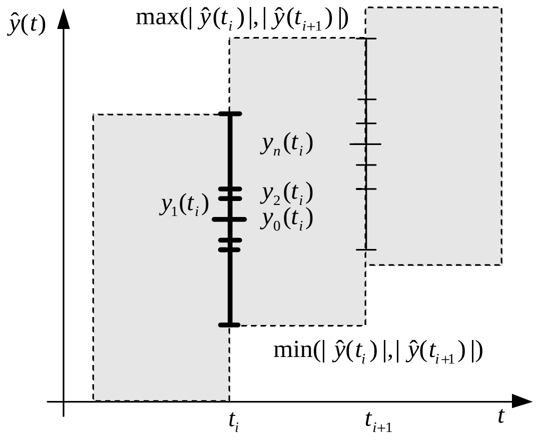
SystemC seems to be the best candidate for this purpose. We combine a SystemC-AMS prototype [6, 12] as simulator with the affine arithmetic library from [4] which provides a ’AAF’ (affine arithmetic form) class in C++. The AAF class symbolically handles affine terms and provides overloaded operations such as ’+’, for example. SystemC(-AMS) is a C++ library, and therefore allows designers to use templates and operator overloading. AMS extensions of SystemC support simulation of continuous-time block diagrams. In SystemC ports and signal of a model are instantiations of the classes

- `sc_in/out<class T>` and/or
- `sc_signal<class T>` resp. `sca_signal<class T>`.

In instantiations of these classes the template class `T` specifies the type of the signal’s values. This type can a be a bit vector (`sc_bit_vector`), for example. In order to combine affine arithmetic with SystemC, we include the affine library which provides the class `AAF`, and instanciate signals and ports with this class as template parameter. For example, designers can declare signals, and apply arithmetic operations, or add uncertainties as follows:

```
sca_signal<AAF> a, b, c;
c = a + quant(b,6)+ noise(3);
```

Visualization The framework for semi-symbolic verification also provides means to write affine terms into VCD files, and to visualize such outputs. For visualization we have extended GTKwave to support analog signals, and to visualize affine expression signals.



$\hat{y}(t)$
$\max(|\hat{y}(t_i)|,|\hat{y}(t_{i+1})|)$
$y_n(t_i)$
$y_1(t_i)$
$y_2(t_i)$
$y_0(t_i)$
$\min(|\hat{y}(t_i)|,|\hat{y}(t_{i+1})|)$
t_i
t_{i+1}
t

Figure 4. Visualization of affine signals.

Figure 4 explains the graphical representation of an affine signal $\hat{y}(t)$. Affine signals are represented as sums of intervals above and below the signal’s central value $y_0(t)$.

Values between two sample points t_i, t_{i+1} are assumed to be between the maximum/minumum of both affine values[1]. As shown in figure 4 simulation runs with affine signals provide for each point in simulated time information about ranges of possible output values. Furthermore, designers can see which of the sources of uncertainty in the design causes how many of the resulting uncertainty.

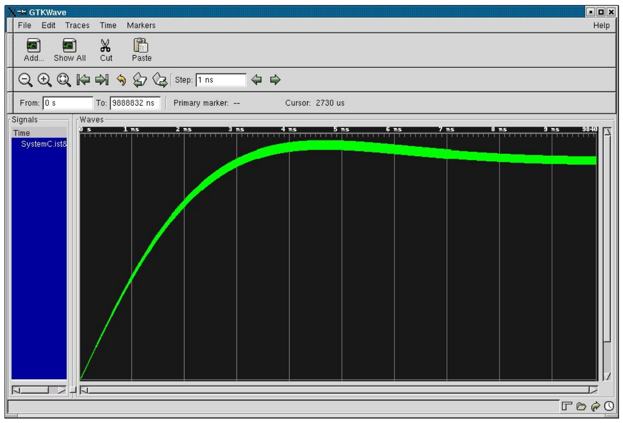

GTKWave
File Edit Traces Time Markers Help
Add... Show All Cut Paste
Step: 1 ns
From: 0 s To: 9888832 ns Primary marker: -- Cursor: 2730 us
Signals
Time
SystemC.ist&
Waves

Figure 5. Visualization of the maximum range of affine output with extended GTKwave.

Figure 5 shows a screenshot of an enhanced version of GTKwave which is able to visualize analog affine signals as shown in figure 4. Actually, only one analog signal is supported. Drawing both specification of the output of characterization *and* actual outputs in one plot and graphical visualization of the different noise symbols is subject of current work.

4. Design Example

To demonstrate the applicability of the method we analyze a control system with feedback. Feedback loops are the main cause of potential problems that we expect: Explosion of the number of affine terms, and/or too large overapproximation. Systems with feedback were problematic in first experiences with more simple analog interval arithmetic [3, 8], where error cancellation was not possible. Furthermore, the concept of error cancellation via feedback is one of the most fundamental concepts of mixed-signal systems and can be found — in different flavors — for example in ΣΔ converters, or in noise shaping. Therefore, a simple system with feedback is an illustrative and yet meaningful example to validate the applicability.

1 This (unfortunately) does not cover the case of a local extremum between t_i, t_{i+1}.

In order to model a control system we describe the system shown in Figure 1 as usual in SystemC with the following differences:

- We use the type AAF instead of `double` to model signals.
- In order to verify the impact of design steps on the system behavior we add a models of an assumed deviations from the ideal behavior.

Figure 6 shows the system we implemented to validate transient simulation with affine expressions.

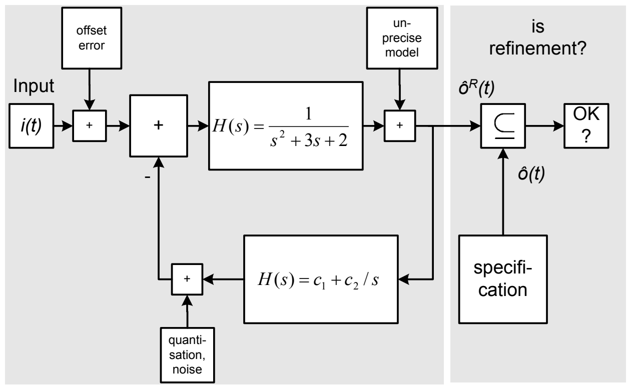


Figure 6. Design Example.

The model includes three different errors modeled by affine expressions: 1.) A potential offset error of a comparator which adds a constant error, 2.) potential errors due to not captured effects in a model, and 3.) quantization noise. These errors occur in different points in the design, and their impact on the dynamic behavior is analyzed by a simulation run.

Figure 7 shows the visualization of some simulation results with gnuplot. The impact of all these errors on the output signal can be displayed separately:

- The offset error in the adder is not reduced by the control system.
- Even large static errors introduced in the control loop (e.g. due to unprecise modeling) fall towards 0.

This is exactly what we expect from a PI controller. If the errors on the output are too high designers can take appropriate measures to bring the output signals directly into the range allowed by the specification.

5. Discussion and Future Work

Although affine arithmetic and its application in circuit or system design is very new, and only few experiences ex-

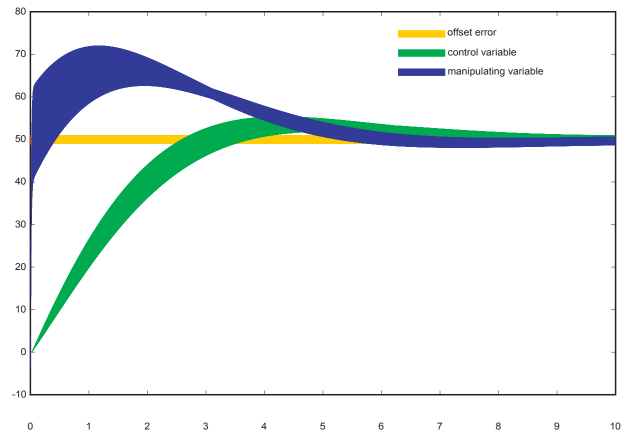


Figure 7. Visualization of simulation results.

ist, these experiences are very promising. Roughly speaking, we made similar experiences compared with [3] for static analysis of DSP algorithms: In practical case studies the number of terms remains limited, and the value range does not grow too much. In our experience the analysis of dynamic properties by a transient simulation with affine arithmetic provides very useful information with only very few simulation runs.

Compared with noise analysis, transient simulation with affine expressions is not restricted to linear systems. In general the methodology is applicable even to complex mixed-signal systems e.g. in ambient intelligence or control systems which combine large software systems with analog and digital signal processing. However, careful modeling of the errors is required in order to avoid an increasing number of terms with simulated time. Compared with Monte-Carlo techniques simulation with affine expressions provides more information: The contribution of each single noise/error source to the total deviation from the central value. Although the case study presented shows that 'error cancellation' does work, we are working on more impressive and complex case studies such as the noise analysis of a $\Sigma\Delta$ converter.

The presented results provide an easy-to-use framework for the refinement of signal processing systems to mixed-signal implementations, because by very few simulation runs, and in a systematic way we can show that a given implementation is a property refinement of a specification. Note that simulation with affine arithmetic could also be an interesting candidate for a combination with methods for property checking of hybrid systems [7].

Although this work most notably deals with interactive design and refinement the methodology and transient simulation with affine arithmetic can be very useful for design automation at system level. For design automation of mixed-signal systems at high level of abstraction the analysis of precision and parasitic effects are actually very important problems which must be solved before developing methods for design automation at system level.

Acknowledgements The authors would like to thank F. Rammig, J. Romberg, and L. Hedrich for valuable discussions. The work was supported by funds of the Deutsche Forschungsgemeinschaft (DFG) under reference number WA 357/14 within the priority program 'Bewertung und Analyse hybrider Systeme'.

References

[1] M. Andrade, J. Comba, and J. Stolfi. Affine Arithmetic (Extended Abstract). In *INTERVAL '94, St. Petersburg, Russia*, 1994.

[2] D. Cansell and D. Méry. Integration of the proof process in the system development through refinement steps. In *Forum on Specification & Design Languages (FDL'02)*, Marseille, France, Sep 2002.

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

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

[5] A. Graupner, S. Getzlaff, R. Schüffny, W. Schwarz, and K. Lemke. Statistical Analysis of Parallel Analog Structures. In *Workshop on System Design Automation (SDA 2000), pp 91-98*, 2000.

[6] C. Grimm. Modeling and Refinement of Mixed Signal Systems with SystemC. In *SystemC – Methodologies and Applications*. Kluwer Academic Publisher (KAP), June 2003.

[7] T. A. Henzinger and P.-H. Ho. Hytech: The cornell hybrid technology tool. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II*, volume 999 of *Lecture Notes on Computer Science*, pages 265–293. Springer, Berlin, 1995.

[8] W. Heupke, C. Grimm, and K. Waldschmidt. A New Method for Modeling and Analysis of Accuracy and Tolerances in Mixed-Signal Systems. In *Proceedings of the Forum on Specification and Design Languages (FDL'03)*, Frankfurt, Germany, Sept. 2003.

[9] A. Lemke, L. Hedrich, and E. Barke. Analog Circuit Sizing Based on Formal Methods Using Affine Arithmetic. In *ICCAD 2002*, 2002.

[10] J. Philipps and B. Rumpe. Roots of refactoring. In K. Baclavski and H. Kilov, editors, *Tenth OOPSLA Workshop on Behavioral Semantics. Tampa Bay, Florida, USA, October 15, 2001*. Northeastern University, 2001.

[11] J. Romberg and C. Grimm. Refinement of Hybrid Systems from HyCharts to SystemC-AMS. In *Proceedings of the Forum on Specification and Design Languages (FDL'03)*, Frankfurt, Germany, 2003.

[12] A. Vachoux, C. Grimm, and K. Einwich. SystemC-AMS Requirements, Design Objectives and Rationale. In *Design, Automation and Test in Europe 2003 (DATE 2003)*, Munich, Germany, 2003.