A Symbolic Methodology for the Verification of Analog and Mixed Signal Designs

Ghiath Al-Sammane
sammane@ece.concordia.ca
Concordia University
Montreal, Quebec, Canada

Mohamed H. Zaki
mzaki@ece.concordia.ca
Concordia University
Montreal, Quebec, Canada

Sofiène Tahar
tahar@ece.concordia.ca
Concordia University
Montreal, Quebec, Canada

Abstract

We propose a new symbolic verification methodology for proving the properties of analog and mixed signal (AMS) designs. Starting with an AMS description and a set of properties and using symbolic computation, we extract a normal mathematical representation for the system in terms of recurrence equations. These normalized equations are used along with an induction verification strategy defined inside the computer algebra system Mathematica to prove the correctness of the properties. We apply our methodology on a third order $\Delta\Sigma$ modulator.

1 Introduction

Analog and mixed signal (AMS) circuits are important integrated circuits that are usually needed at the interface between the electronic system and the real world. They are mainly used in the interface between digital and analog components, such as analog to digital (A/D) and digital to analog (D/A) converters. The verification of AMS systems, however, is a challenging task that requires lots of expertise and deep understanding of the system behavior. It relies largely on a mixture of some manual calculations over the symbolic analysis expressions and numerical simulation.

Today numerical simulation is the dominant approach that attempts to check the behavior of circuits against a finite number of input signals. However, since exhaustive simulation is impossible, we cannot guarantee the correctness of the system outside the simulated input cases. By contrast, formal verification techniques aim to prove that a circuit behaves correctly for all possible input signals and initial conditions and that none of them drives the system into a non expected behavior. As a middle way technique, symbolic analysis [6] attempts to compute symbolic expressions for transfer functions and for other circuit characteristics in terms of the element parameters. These expressions are then used (in a close connection with numerical methods) to determine the influences of element parameters on circuit behavior, to estimate the error and tolerance analysis and to optimize parameters.

In this paper, we propose a new formal verification methodology for proving the properties of AMS designs using symbolic computation. The AMS system (both the analog and digital parts) as well as the properties are described in terms of recurrence equations that are used along with an induction verification strategy defined inside the computer algebra system *Mathematica* [14] to prove the correctness of the properties.

The main advantage of our methodology is that it can be completely automatized and integrated into the design flow. The idea is to verify the system automatically in the same sense that the formal verification of digital systems is achieved using logical decision procedures like SAT and BDD. We illustrate our methodology on a third order $\Delta\Sigma$ modulator, a widely used circuit in digital signal processing, for which stability analysis remains a challenging problem.

The rest of the paper is organized as follows: we start by discussing relevant related work in Section 2, followed by an overview of the verification methodology in Section 3. In Sections 4 and 5, we define the mathematical representation and the symbolic simulation method used, respectively. The verification strategy is described in Section 6 with an illustrative application shown in Section 7, before concluding with a discussion in Section 8.

2 Related Work

Following the success of formal verification methods for digital designs, researchers started recently tackling the verification of AMS systems. For instance, model checking and reachability analysis were used for validating AMS designs over a range of parameter values and a set of possible input signals. Several methods for approximating reachable sets for continuous dynamics have been proposed in the open literature. They rely on the discretization of the continuous state space and were first proposed in [10] and [7]. In [10], the authors construct a finite-state discrete abstraction of electronic circuits by partitioning the continuous state space into fixed size hypercubes and computed the reachability relations between these cubes using numerical techniques. In [7], the authors tried to overcome the expensive computational method in [10], by combining discretization and projection techniques of the state space, hence reducing its dimension. While the approach in [7] is less precise due to the use of projection techniques, it is still sound. Variant approaches of the latter analysis were adopted in [4], [8] and [5]. For instance, the model checking tools *d/dt* [4], *Checkmate* [8] and *PHaver* [5] were used in the verification of a biquad low-pass filter [4], a tunnel diode oscillator and a $\Delta\Sigma$ modulator [8], and voltage controlled oscillators [5]. Other computational techniques for AMS verification can be found in [9].

A few other approaches have been proposed to solve reachability analysis of AMS systems. For instance, in [4], in order to tackle the state explosion problem due to the exhaustive analysis, the authors proposed using optimization techniques in order to find bounds of the reachability. The idea is to formulate bounded reachability analysis as a hybrid constrained optimization problem that can be solved by techniques such as mixed-integer linear programming. For an overview and comparison of the different projects related to the subject, refer to [15].

In contrast to the above discussed mathematical models, we propose in this paper a representation based on algebraic models that capture the continuous behavior without any approximation or discretization. In fact, all above surveyed formal methods limit the verification of the circuit to predefined time intervals. We overcome this limitation by basing our methodology on mathematical induction.

3 Symbolic Simulation Methodology

Our methodology aims to prove by induction that an AMS description satisfies a set of properties. This is archived via several steps as shown in Figure 1.

The AMS description is composed in general of a digital part and an analog part. For the analog part, it could be described using recurrence equations or a set of differential-algebraic equations (DAEs) that can be converted into an equivalent set of difference equations. For the digital part, it could be described using a hardware description language (HDL) like VHDL. The properties are algebraic relations between signals of the system.

The AMS description and properties are input to a symbolic simulator that performs a set of transformations by rewriting rules in order to obtain a normal mathematical representation called System of Recurrence Equations (SRE to be described in Section 4). These are combined recurrence relations that describe each property blended directly by the behavior of the system. The next step is the proof by induction which is defined over the normal structure of the SRE. If the proof is obtained, then the property is verified. Otherwise, we provide counterexamples for the non-proved properties.

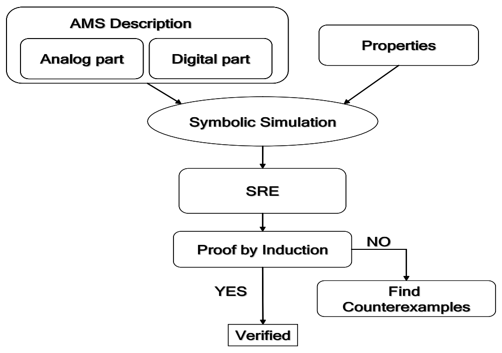

Figure 1. Overview of the Methodology

The symbolic simulator and the proof strategy are implemented inside the computer algebra system *Mathematica* , which provides special functions to simplify and prove algebraic relations. The advantage of using Mathematica over other systems is the availability of numerous built-in functions and proof capabilities.

4 The System of Recurrence Equations (SRE)

A recurrence equation or a difference equation is the discrete version of an analogue differential equation. In conventional system analysis, recurrence equations are used in the definition of relations between consecutive elements of a sequence.

In [1], the notion of recurrence equation is extended to describe digital circuits using the normal form: generalized `If-formula`.

Definition 4.1. Generalized `If-formula`
In the context of symbolic expressions, the generalized `If-formula` is a class of expressions that extend recurrence equations to describe digital systems. Let $\mathbb{K}$ be a numerical domain ($\mathbb{N}, \mathbb{Z}, \mathbb{Q}$, $\mathbb{R}$ and $\mathbb{B}$), a generalized `If-formula` is one of the following:

- A variable $X_i(n)$ or a constant $C \in \mathbb{K}$
- Any arithmetical operation $\diamond \in \{+, -, \div, \times\}$ between variables $X_i(n) \in \mathbb{K}$
- A logical formula: any expression constructed using a set of variables $X_i(n) \in \mathbb{B}$ and logical operators: $not, and, or, xor, nor, \dots$ etc.
- A comparison formula: any expression constructed using a set of $X_i(n) \in \mathbb{K}$ and comparison operator $\alpha \in \{=, \neq, <, \leq, >, \geq\}$.
- An expression $IF(X,Y,Z)$, where X is a logical formula or a comparison formula and Y,Z are any generalized `If-formula`. Here, $IF(x,y,z) : B \times K \times K \longrightarrow K$ satisfies the axioms:
  (1) $IF(True, X, Y) = X$

(2) $IF(False, X, Y) = Y$

Definition 4.2. A System of Recurrence Equations (SRE)
Consider a set of variables $X_i(n) \in \mathbb{K}$, $i \in V = 1, \ldots, k$, $n \in \mathbb{Z}$, an SRE is a system of the form:

$$X_i(n) = f_i(X_j(n-\gamma)), (j, \gamma) \in \varepsilon_i, \forall n \in \mathbb{Z}$$

where $f_i(X_j(n-\gamma))$ is a generalized `If-formula`. The set ε_i is a finite non empty subset of $1, \ldots, k \times \mathbb{N}$. The integer γ is called the delay.

5 The Symbolic Simulation Algorithm

The symbolic simulation algorithm is based on rewriting by substitution. The computation aims to obtain the SRE defined in the previous section. In the context of functional programming and symbolic expressions, we define the following functions.

Definition 5.1. Substitution
Let u and t be two distinct terms, and x a variable. We call $x \to t$ a substitution rule. We use $Replace(u, x \to t)$, read Replace in u any occurrence of x by t, to apply the rule $x \to t$ on the expression u.

The function *Replace* can be generalized to include a list of rules. *ReplaceList* takes as arguments an expression *expr* and a list of substitution rules $\{R_1, R_2, \ldots, R_n\}$. It applies each rule sequentially on the expression.

Definition 5.2. Substitution Fixpoint
A substitution fixpoint $FP(Expr, R)$ is obtained, if: $Replace(expr, R) \equiv Replace(Replace(expr, R), R)$

Algorithm 5.3. Repetitive Substitution
$ReplaceRepeated(Expr, \mathcal{R})$ applies a set of rules $\mathcal{R}$ on an expression *Expr* until a fixpoint is reached.

```
Begin
    $Expr = expr$
    Do
        $Expr_t = ReplaceList(Expr, \mathcal{R})$
        $Expr = Expr_t$
    Until $FP(Expr_t, \mathcal{R})$
End
```

The correctness of this rewriting algorithm is discussed in [1].

Depending on the type of expressions, we distinguish three kinds of rewriting rules:
Polynomial symbolic expressions R_{Math}: are built-in rules intended for the simplification of polynomial expressions ($\mathbb{R}^n[x]$).
Logical symbolic expressions R_{Logic}: are rules intended for the simplification of Boolean expressions and to eliminate obvious ones like $(and(a, a) \to a)$ and $(not(not(a)) \to a)$.
If-formula expressions R_{IF}: are rules intended for the simplification of computations over `If-formulae`. The definition and properties of the *IF* function, like reduction and distribution, are used (see [11] for more details).

- IF Reduction: $IF(x, y, y) \to y$
- IF Distribution: $f(A_1, \ldots, IF(x, y, z), \ldots, A_n) \to IF(x, f(A_1, \ldots, y, \ldots, A_n), f(A_1, \ldots, z, \ldots, A_n))$

Equation rules R_{Eq}: result from converting an SRE into a set of substitution rules.

Algorithm 5.4. Symbolic Computation over *SREs*
A symbolic computation over *SREs* is:

$$Symbolic\_Comp(X_i(n)) = ReplaceRepeated(X_i(n), R_{simp})$$

where, $R_{simp}(t) = R_{Math} \cup R_{Logic} \cup R_{IF} \cup R_{Eq}$

In the case of symbolic expressions over $\mathbb{R}$, the normal form is obtained using a Buchberger based algorithm for the construction of Gröbner base [2].

The objective of the symbolic computation is to obtain a normal form for cases like $a + IF(x > 0, b, a)$. *Symbolic_Comp* rewrites this expression using two rules:

- IF Distribution : $a + IF(x > 0, b, a) \to IF(x > 0, b + a, a + a)$
- Polynomial Addition: $IF(x > 0, b + a, a + a) \to IF(x > 0, b + a, 2a)$

The proof of termination and confluence of the rewriting system formed by all these rules is discussed in [1].

6 The Verification Engine

After the symbolic simulation, we obtain a recurrence relation for each property: $P(n)$. Starting from the fact that $P(n)$ is an `If-formula`, we want to verify $P(n)$ under a set of constraints *Cond* given by the designer. The constraints define the environment of the correctness of the property under verification. We have defined our verification algorithm in Mathematica using functions that reduce a combination of algebraic equations and inequalities under constraints. In following, we describe a set of Mathematica functions that we use in our verification algorithm.

6.1 Mathematica built-in Functions

Definition 6.1. Quantifiers
$ForAll[x, cond, expr]$, and $Exists[x, cond, expr]$ are Mathematica functions that represent quantifiers. *ForAll* states that *expr* is *True* for all x satisfying the condition *cond*. *Exists* states that there exists an x satisfying the condition *cond* for which *expr* is *True*.

Definition 6.2. $Reduce[expr, vars]$
The function *Reduce* simplifies the statement *expr* by solving equations or inequalities for *vars* and eliminating quantifiers. The statement expr can be any logical combination of:

- $lhs = rhs$ Equations
- $lhs \diamond rhs$, where $\diamond \in \{\neq, \leqslant, <, >, \geqslant\}$ Inequalities
- $expr \in dom$ Domain Specifications
- $ForAll[x, cond, expr]$ Universal Quantifiers
- $Exists[x, cond, expr]$ Existential Quantifiers

Reduce gives $True$ if the *expr* is proved to be always true, $False$ if *expr* is proved to be always false and a reduced *expr* otherwise. The implementation of the function *Reduce* is inspired by a real polynomial decision algorithm defined in [12].

Reduce always returns a complete representation of the solution to a system of equations or inequalities. Sometimes, however, we may just want to find particular sample solutions (as is the case for counterexamples). This is possible using *FindInstance*.

Definition 6.3. $FindInstance[expr, vars, assum]$
FindInstance finds an instance of *vars* that makes *expr* $True$ if an instance exists, and gives $\{\}$ if it does not. The result of *FindInstance* is of the form
$\{\{v_1 \rightarrow instance_1, v_2 \rightarrow instance_2, \ldots, v_m \rightarrow instance_m\}\}$
where $vars = \{v_1, v_2, \ldots, v_m\}$ *expr* can contain inequalities, domain specifications and quantifiers. $FindInstance$ may be able to find instances even if *Reduce* cannot give a complete reduction. The implementation of *FindInstance* is based on variants of Newton's, Secant and Brent's methods [13].

6.2 Verification Algorithm

We use mathematical induction to prove that $P(n)$ holds for all natural numbers n. An inductive proof consists of proving the following two subgoals:

- Prove that $P(t_0)$ is true.
- Prove that $\forall k, P(k) \implies P(k+1)$.

Definition 6.4. Property
A property P is a relation of the form: P = $quant(X, cond, expr)$, where $quant \in \{\forall, \exists\}$, X is a set of variables, *cond* is a logical combination of comparison formulae constructed over variables X and *expr* is an `If-formula` that takes values in the Boolean Domain $\mathbb{B}$.

Algorithm 6.5. The Prove Function
We define the function *Prove* that tries to prove a property of the form $quant(X, cond, expr)$ using the function *Reduce*, otherwise it gives a counterexample using *FindInstance*.

```
$Prove(quant(X, cond, expr)) =$
    $If(Reduce(quant(X, cond, expr), var) = True,$
      $True,$
      $FindInstance(cond \wedge \neg expr, var)$
```

Algorithm 6.6. The SplitProve Function
Let P be a property of the form $quant(X, cond, expr)$. We define the function *SplitProve*, which depending on the `If-formula` structure of *expr*, applies the function *Prove* or splits the verification as follows:

- *expr* is a comparison formula C,
  $SplitProve(quant(X, cond, C)) =$
  $Prove(quant(X, cond, C))$
- *expr* is a logical formula of the form $a \diamond b$, with $\diamond \in \{\neg, \wedge, \vee, \oplus, \ldots\}$ and a, b are `If-formulae` that take values in $\mathbb{B}$
  $$SplitProve(P)) = SplitProve(quant(X, cond, a)) \diamond SplitProve(quant(X, cond, b))$$
- *expr* is an expression of the form $IF(q, l, r)$
  $$SplitProve(P) = SplitProve(quant(X, cond \wedge q, l)) \vee SplitProve(quant(X, cond \wedge \neg q, r))$$

Algorithm 6.7. Proof by Induction
Let $P(n)$ be the recurrence equation of the property P written as an `If-formula`. Let $cond_{n_0}$ be the initial condition at time n_0, $cond_n$ the constraints that are true for all $n > n_0$, and X the set of dependency variables of $P(n)$. The proof by induction over n is :
$SplitProve(ForAll(X_{n_0}, cond_{n_0}, P(n_0)))$
$\wedge$
$SplitProve(ForAll(n > n_0 \wedge X_n, n \in \mathbb{N} \wedge cond_n \wedge P(n), P(n+1)))$

7 Application: The $\Delta\Sigma$ Modulator

In this section, we will illustrate the application of the verification methodology and algorithms proposed in the previous section. We have chosen a third-order $\Delta\Sigma$ modulator as an application case study.

Data converters are needed at the interface of analog medium and digital processing units. The principle of the $\Delta\Sigma$ architecture is to make over several stages rough evaluations of the signal, to measure the error, integrate it and then

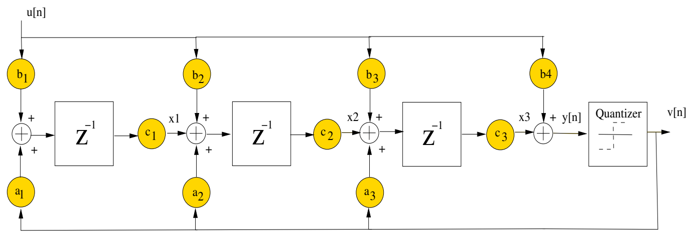

Figure 2. Third Order ΔΣ Modulator

compensate for that error. Higher-order single stage modulators have been proposed to increase resolution by adding more integral and feedback paths. The number of integrators, and consequently, the numbers of feedback loops, indicates the order of a ΔΣ modulator. A third order modulator is shown in Figure 2.

A ΔΣ modulator is said to be stable if the integrator output remains bounded under a bounded input signal, thus avoiding that the quantizer in the modulator to be overloaded. This property is of a great importance since the integrator saturation can deteriorate circuit performance, hence leading to instability. Increasing the signal levels inside the ΔΣ modulator affects stability in the following way. If the signal level at the quantizer exceeds the maximum output level by more than the maximum error value, a quantizer overload occurs. The overload behavior can be checked by observing the quantizer input. The quantizer in the modulator shown in Figure 2 is a one-bit quantizer with two quantization levels, $+1$V and -1V. Hence, the quantizer input should be between -2V and $+2$V in order to avoid overloading [8].

In summary, stability analysis for third and higher-order ΔΣ modulators is a challenging research problem that has also been studied with formal methods [4, 8]. In the remaining of this section, we are interested in finding solutions for the following problem: *Given a set of initial conditions and state space constraints on the system variables, will the quantizer overload (leading to instability)?*

7.1 ΔΣ Modulator Description

The specification of such an AMS system is given usually using vectors recurrence equations:

$$X(k+1) = C\,X(k) + B\,u(k) + A\,v(k)$$

where A, B and C are matrices providing the parameters of the circuit and $u(k)$ is the input. For the third order modulator in Figure 2, we obtain the following recurrence equations for the analog part of the system:

$$\begin{aligned} x_1(k+1) &= x_1(k) + b_1 u(k) + a_1 v(k) \\ x_2(k+1) &= c_1 x_1(k) + x_2(k) + b_2 u(k) + a_2 v(k) \\ x_3(k+1) &= c_2 x_2(k) + x_3(k) + b_3 u(k) + a_3 v(k) \end{aligned}$$

$v(k)$ is the digital part of the system. As in the specification of the ΔΣ modulator, we consider only the behavior when the rising edge of the clock is true. Also, the condition of the threshold of the quantizer is computed to be equal to $c3x3(k) + u(k)$. The digital description of the quantizer is transformed into a recurrence equation using the approach defined in [1]. Thus, the equivalent recurrence equation that describes $v(k)$ is $v(k) = IF(c3x3(k) + u \geq 0, -a, a)$
The stability property P of the ΔΣ modulator is written as:

$$P(k) = ForAll(k \geq 0, Cond, -1 < x3(k) < 1)$$

The condition $Cond$ depends on the particular application.

7.2 Symbolic Simulation of the ΔΣ Modulator's SRE

Applying the symbolic simulation Algorithm 5.4, we obtain the following results for the signals of the system.

$$\begin{aligned} x_1(k+1) &= if(c_3x_3(k) + u >= 0, x_1(k) + b_1 u - a_1 a, \\ &\quad x_1(k) + b_1 u + a_1 a) \\ x_2(k+1) &= if(c_3x_3(k) + u >= 0, c_1x_1(k) + x_2(k) + b_2u(k) \\ &\quad -a_2 a, c_1x_1(k) + x_2(k) + b_2u(k) + a_2 a) \\ x_3(k+1) &= if(c_3x_3(k) + u >= 0, c_2x_2(k) + x_3(k) + b_3u(k) \\ &\quad -a_3 a, c_2x_2(k) + x_3(k) + b_3u(k) + a_3 a) \end{aligned}$$

The expression of the property after the simulation is:

$$\begin{aligned} P(k+1) = \quad & if(c_3x_3(k) + u >= 0, \\ & -1 < c_2x_2(k) + x_3(k) + b_3u(k) - a_3a < 1, \\ & -2 < c_2x_2(k) + x_3(k) + b_3u(k) + a_3a < 2) \end{aligned}$$

7.3 Proving the Stability of the ΔΣ Modulator

The correctness of the property $P(k+1)$ depends on the parameters A, B and C, the values of variables $x_1(k)$, $x_2(k)$ and $x_3(k)$, the time k, and the input signal $u(k)$.

We verify the ΔΣ modulator for two sets of parameters inspired from the analysis in [8]:

$$Parameters_1 \rightarrow \left\{ \begin{array}{ccc} a = 2 & a_1 = 0.044 & a_2 = 0.2881 \\ a_3 = 0.7997 & b_1 = 0.044 & b_2 = 0.2881 \\ b_3 = 0.7997 & c_1 = c_2 = c_3 = 1 & \end{array} \right.$$

$$Parameters_2 \rightarrow \left\{ \begin{array}{ccc} a = 2 & a_1 = 0.044 & a_2 = 0.2881 \\ a_3 = 0.7997 & b_1 = 0.07333 & b_2 = 0.2881 \\ b_3 = 0.7997 & c_1 = c_2 = c_3 = 1 & \end{array} \right.$$

We apply the Algorithm 6.7 in order to verify the ΔΣ modulator stability for the above sets of parameters and for two cases of conditions (state space constraints). Table 1 below summarizes the verification results. The property is *True* if it is proved under the set of conditions and the set of parameters for all $k > 0$. If there is no k for which the property is valid then it is *False*, and a counterexample is provided. When the property is valid for some values of k and not for other values, we say that the property is not proved and counterexamples are provided for both cases.

Table 1. Verification Results

	State Space Constraints	Property with Parameters$_1$	Property with Parameters$_2$
case1	Values at t=0		
	$0 \leq x_1(0) \leq 0.01$ $-0.01 \leq x_2(0) \leq 0$ $0.8 \leq x_3(0) \leq 0.82, u := 0.6$	True	True
	Values at t=n		
	$-0.1 \leq x_1(n) \leq 0.1$ $-0.5 \leq x_2(n) \leq 0.5$ $0.5 \leq x_3(n) \leq 1.5, u := 0.6$	Not Proved	True
case2	Values at t=0		
	$0 \leq x_1(0) \leq 0.02$ $-0.03 \leq x_2(0) \leq -0.01$ $1 \leq x_3(0) \leq 1.4, u := 0.8$	False $x_2[0] \mapsto -0.015569$ $x_3[0] \mapsto 1.4$	False $x_2[0] \mapsto 0.0227935$ $x_3[0] \mapsto 1$
	Values at t=n		
	$-0.1 \leq x_1(n) \leq 0.1$ $-1 \leq x_2(n) \leq 0.5$ $-1 \leq x_3(n) \leq 2.5, u := 0.8$	N/A	N/A

8 Conclusions

In this paper, we have presented a formal verification methodology for AMS designs. We used the notion of system of recurrence equations (SRE) as a mathematical model that can represent both the digital and analog parts of the design. The symbolic computation algorithm produces a set of recurrence relations; one for each property of the design. We have defined and implemented an induction based technique that traverses the structure of the normalized properties and provides a formal correctness proof or a counterexample, otherwise. The counterexample can then be used to refine the proof and be applied as a test case.

We have applied the approach on a third order $\Delta\Sigma$ modulator, which has illustrated the strength of the approach. In fact, we have proved the stability of the circuit under a set of given parameters and provided counterexamples for the failed properties. Our methodology overcomes the time bound limitations of exhaustive methods developed in related work.

Future work includes more applications in order to identify the limitations of the proposed approach. Additional work is needed in order to integrate the methodology in the design process. In this case, we are looking into the automatic generation of the SRE model from circuit descriptions given in HDL-AMS languages such as VHDL-AMS [3], as well as the definition of an expressive property language for specifying properties of analog signals.

References

[1] G. Al-Sammane. *Simulation Symbolique des Circuits Decrits au Niveau Algorithmique*. PhD thesis, Université Joseph Fourier, Grenoble, France, July 2005.

[2] B. Buchberger. A Theoretical Basis for the Reduction of Polynomials to Canonical Forms. In *SIGSAM Bulletin*, volume 10, pages 19–29, USA, 1976. ACM Press.

[3] E. Christen and K. Bakalar. VHDL-AMS - A Hardware Description Language for Analog and Mixed-signal Applications. In *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing.*, volume 46, pages 1263–1272, 1999.

[4] T. Dang, A. Donzé, and O. Maler. Verification of Analog and Mixed-Signal Circuits Using Hybrid System Techniques. In *Formal Methods in Computer-Aided Design*, volume 3312 of *LNCS*, pages 21–36. Springer, 2004.

[5] G. Frehse, B. H. Krogh, and R. A. Rutenbar. Verifying Analog Oscillator Circuits Using Forward/Backward Abstraction Refinement. In *Design, Automation and Test in Europe*, pages 257 – 262, Munich, Germany, March 2006. IEEE/ACM.

[6] G. G. E. Gielen and R. A. Rutenbar. Computer-aided Design of Analog and Mixed-signal Integrated Circuits. In *Proceedings of the IEEE*, volume 88, pages 1825–1852, 2000.

[7] M. R. Greenstreet and I. Mitchell. Reachability Analysis Using Polygonal Projections. In *Hybrid Systems: Computation and Control*, volume 1569 of *LNCS*, pages 103–116. Springer, 1999.

[8] S. Gupta, B. H. Krogh, and R. A. Rutenbar. Towards Formal Verification of Analog Designs. In *International Conference on Computer-Aided Design*, pages 210–217. IEEE/ACM, 2004.

[9] W. Hartong, R. Klausen, and L. Hedric. Formal Verification for Nonlinear Analog Systems: Approaches to Model and Equivalence Checking. In *Advanced Formal Verification*, pages 205–245. Kluwer, 2004.

[10] R. P. Kurshan and K. L. McMillan. Analysis of Digital Circuits Through Symbolic Reduction. In *IEEE Trans. on CAD of Integrated Circuits and Systems*, volume 10, pages 1356–1371, 1991.

[11] J. S. Moore. Introduction to the OBDD Algorithm for the ATP Community. In *Journal of Automated Reasoning*, volume 12, pages 33–46. Springer, 1994.

[12] A. Strzebonski. A Real Polynomial Decision Algorithm Using Arbitrary-Precision Floating Point Arithmetic. In *Reliable Computing*, volume 5, pages 337–346. Springer, 1999.

[13] A. Swift and G. R. Lindfield. Comparison of a Continuation Method with Brent's Method for the Numerical Solution of a Single Nonlinear Equation. In *Computer Journal*, volume 21, pages 359–362. OUP, 1978.

[14] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison Wesley Longman Publishing, USA, 1991.

[15] M. H. Zaki, S. Tahar, and G. Bois. Formal Verification of Analog and Mixed Signal Designs: Survey and Comparison. In *IEEE Northeast Workshop on Circuits and Systems*, pages 281–284, Montréal, Canada, 2006.