CAFFEINE: Template-Free Symbolic Model Generation of Analog Circuits via Canonical Form Functions and Genetic Programming

Trent McConaghy, Tom Eeckelaert, Georges Gielen

K.U. Leuven, ESAT-MICAS, Kasteelpark Arenberg 10, B-3001 Leuven, Belgium
{trent.mcconaghy},{tom.eeckelaert},{georges.gielen}@esat.kuleuven.ac.be

Abstract:

This paper presents a method to automatically generate compact symbolic performance models of analog circuits with no prior specification of an equation template. The approach takes SPICE simulation data as input, which enables modeling of any nonlinear circuits and circuit characteristics. Genetic programming is applied as a means of traversing the space of possible symbolic expressions. A grammar is specially designed to constrain the search to a canonical form for functions. The approach generates a set of symbolic models that collectively provide a tradeoff between error and model complexity. Experimental results show that the symbolic models generated are compact and easy to understand, making this an effective method for aiding understanding in analog design. The models also demonstrate better prediction quality than posynomials.

1. Introduction

Symbolic models of analog circuits have many applications. Fundamentally, they increase a designer's understanding of a circuit, which leads to better decision-making in circuit sizing, layout, verification, and topology design. Automated approaches to symbolic model generation are therefore of great interest.

In *symbolic analysis*, models are derived via topology analysis. [1] is a survey. Its main weakness is that it is limited to linear and weakly nonlinear circuits. Leveraging SPICE in modeling is promising because simulators readily handle nonlinear circuits, as well as environmental effects, manufacturing effects, and different technologies. Simulation data has been used to train neural networks as in [2,3,4]. However, such models provide no insight.

The aim of *symbolic modeling* is to *use simulation data* to generate interpretable mathematical expressions that relate the circuit performances to the design variables. In [5,6], symbolic models are built with posynomials. Unfortunately, the models are constrained to templates, which restricts the functional form and also imposes bias. Also, the models have dozens of terms, limiting their interpretability. Finally, the approach assumes posynomials can fit the data; in analog circuits there is no guarantee of this, and one might never know in advance.

The problem we address in this paper is how to generate symbolic models with more open-ended functional forms (i.e. without a pre-defined template), for arbitrary nonlinear circuits, and at the same time ensure that the models are interpretable. A target flow that reflects these goals is shown in Figure 1.

We approach the question by starting with genetic programming (GP) [7], but constraining it via a *grammar* designed to generate *interpretable* models. We name the approach CAFFEINE: Canonical Functional Form Expressions in Evolution.

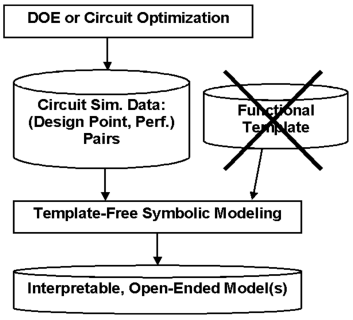

Figure 1: Template-free symbolic modeling flow

The contributions of this paper are as follows:

- To the best of our knowledge, a first-ever tool to do template-free symbolic modeling, with the flexibility of SPICE simulations therefore allowing modeling of any nonlinear circuits.
- A means to make the models compact and understandable, yet with arbitrary accuracy; in fact providing a tradeoff between accuracy and complexity. Final models are highly predictive.

This paper is organized as follows. Section 2 defines the problem. Sections 3 and 4 describe CAFFEINE and the grammar. Section 5 has results; section 6 concludes.

2. Problem Formulation

Given: A set of $\{\mathbf{x}(t),y(t)\}, t=1..N$ data samples where $\mathbf{x}(t)$ is a d-dimensional design point t and $y(t)$ is a corresponding circuit performance value measured from simulation, and *no* model template

Determine: A set of symbolic models $f^* \in F$ that provide a tradeoff between prediction error and complexity.

3. CAFFEINE

Genetic Programming (GP) [7] is an evolutionary algorithm, where GP individuals (points in the design space) are *trees*. It can evolve unrestricted functional forms, but those functions are virtually un-interpretable.

CAFFEINE extends GP, attacking interpretability in two main ways: a multi-objective approach [8] that provides a set of models that trade off normalized mean-squared error and complexity, and more notably, a specially designed grammar to constrain the search to specific functional forms without cutting out good solutions. "Complexity" is dependent on the number of basis functions, the number of nodes in each tree, and the exponents of "variable combos" (VCs, described later):

$$\text{complexity}(f) = \sum_{j=1}^{M_f} (w_b + \text{nnodes}(j) + \sum_{k=1}^{\text{nvc}(j)} \text{vccost}(vc_{k,j})) \qquad (1)$$

where w_b is a constant to give a minimum cost to each basis function, nnodes(j) is the number of tree nodes of basis function j, nvc(j) is number of VCs of basis function j, and $\text{vccost}(\text{vc}) = w_{vc} \sum_{\text{dim}=1}^{d} \text{abs}(\text{vc}(\text{dim}))$.

4. Grammar and Operators

A grammar can constrain GP [9]. Evolutionary operators must respect the derivation rules of the grammar, i.e. only subtrees with the same root can be crossed over, and random generation of trees must follow the derivation rules. Even though grammars can usefully constrain search, none have yet been carefully designed for functional forms. CAFFEINE is for functions:

```
REPVC =>  'VC' | REPVC '*' REPOP | REPOP
REPOP => REPOP '*' REPOP | 1OP '(' 'W' '+' REPADD
')' | 2OP '(' 2ARGS ')' | ... 3OP, 4OP etc
2ARGS => 'W' '+' REPADD ',' MAYBEW | MAYBEW ','
'W' '+' REPADD
MAYBEW => 'W' | 'W' '+' REPADD
REPADD => 'W' '*' REPVC | REPADD '+' REPADD
2OP    => 'DIVIDE' |'POW' | 'MAX' | ...
1OP    => 'INV' | 'LOG10' | ...
```

Terminal symbols are in quotes; the remaining symbols are nonterminal, which means that they expand. Each nonterminal symbol has a set of derivation rules separated by '|'. The start symbol is REVPC. An individual is a set of trees (set of basis functions); basis functions are linearly weighted using least-squares learning. Basis function operators include: creating a new individual by randomly choosing >0 basis function from each of 2 parents; deleting a random basis function; adding a randomly generated tree as a basis function; copying a subtree from one individual to make a new basis function for another.

The root is a product of variables and/or nonlinear functions (REPVC and REPOP). Within each nonlinear function is a weighted sum of basis functions (REPADD). Each basis function can be, once again, a product of variables and/or nonlinear functions. And so on.

A 'VC' is a rational combination of variables. With each VC, a vector holding an integer value per design variable as the variable's exponent. An example vector is [1,0,-2,1], which means $(x_1 * x_4) / (x_3)^2$. VC operators include: one point crossover, and randomly adding or subtracting to an exponent value.

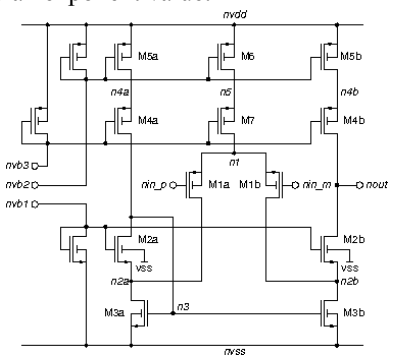

Figure 2: Schematic of high-speed CMOS OTA

5. Experiments

5.1 Experimental Setup

The circuit being modeled is a high-voltage CMOS OTA as shown in Figure 2. The goal is to discover expressions for low-frequency gain (A_{LF}), unity-gain frequency (f_u), phase margin (PM), input-referred offset voltage (v_{offset}), and the positive and negative slew rate (SR_p, SR_n). To allow a direct comparison to posynomials [6], an identical problem setup was used, with the exception that we did not pre-scale the data (in the aim of interpretable expressions).

For model input variables, we used an operating-point driven formulation [10] (device sizes could have been readily used instead). For training inputs, orthogonal-hypercube Design-Of-Experiments (DOE) sampling of design points was used, with scaled dx=0.1 to have 243 samples with three simulations each. Testing inputs were also sampled with DOE and 243 samples, but with dx=0.03. Run settings were: maximum number of basis functions = 15, population size 200, 5000 generations, maximum tree depth 8, w_b = 10, and w_{vc} = 0.25. Single-input operators allowed were: ln, $\log_{10}$, $1/x$, abs, sin, cos, tan, max(0, x), min(0,x), 2^x, 10^x, sqrt, and sqr. Double-input operators allowed are add, mult, divide, and power. Also, a lte() variant was used.

5.2 Results and Discussion

Figure 3 illustrates CAFFEINE-generated tradeoffs between training error (q_{wc}) and complexity. Each point is a different model. As expected, the number of basis functions usually rises with the complexity, but not always, as larger trees increase complexity too. Figure 3 also shows testing error (q_{tc}). Unlike training error, it is not monotonically decreasing as complexity rises, which means that some less complex models are more predictive

than more complex ones. However, because our goal is interpretable expressions, we can prune away models not on the *testing* error vs. complexity tradeoff.

Note the testing error is almost always lower than the training error. While odd at first glance, this is actually alright, because this testing data tests *interpolation* ability (training had dx=0.10, but testing had dx=0.03).

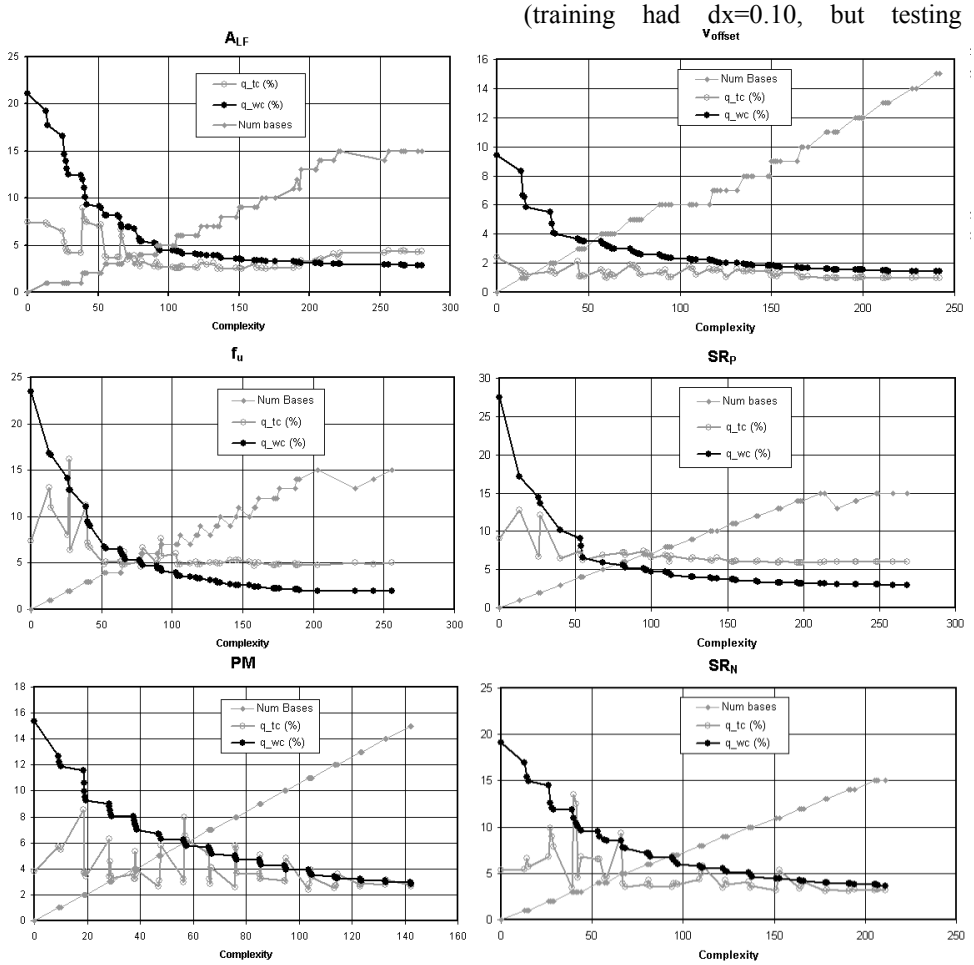

Figure 3: Models' training error (q_{wc}), testing error (q_{tc}), and number of bases vs. complexity

Perf.	Target (%)		Expression
	q_{wc}	q_{tc}	
A_{LF}	10	10	-10.3 + 7.08e-5 / id1 + 1.87 * ln(-1.95e+9 + 1.00e+10 / (vsg1*vsg3)+ 1.42e+9 *(vds2*vsd5) / (vsg1*vgs2*vsg5*id2))
f_u	10	10	10^(5.68 - 0.03 * vsg1 / vds2 - 55.43 * id1+ 5.63e-6 / id1)
PM	10	10	90.5 + 190.6 * id1 / vsg1 + 22.2 * id2 / vds2
v_{offset}	10	10	- 2.00e-3
SR_p	10	10	2.36e+7 + 1.95e+4 * id2 / id1 - 104.69 / id2 + 2.15e+9 * id2 + 4.63e+8 * id1
SR_n	10	10	- 5.72e+7 - 2.50e+11 * (id1*id2) / vgs2 + 5.53e+6 * vds2 / vgs2 + 109.72 / id1

Table I: CAFFEINE-generated symbolic models which have less than 10% training and testing error

Test error (%)	Train error (%)	PM Expression
3.98	15.4	90.2
3.71	10.6	90.5 + 186.6 * id1 + 22.1 * id2 / vds2
3.68	10.0	90.5 + 190.6 * id1 / vsg1 + 22.2 * id2 / vds2
3.39	8.8	90.1 + 156.85 * id1 / vsg1 - 2.06e-03 * id2 / id1 + 0.04 * vgs2 / vds2
3.31	8.0	91.1 - 2.05e-3 * id2 / id1 + 145.8 * id1 + 0.04 * vgs2 / vds2 - 1.14 / vsg1
3.20	7.7	90.7 - 2.13e-3 * id2 / id1 + 144.2 * id1 + 0.04 * vgs2 / vds2 - 1.00 / (vsg1*vsg3)
2.65	6.7	90.8 - 2.08e-3 * id2 / id1 + 136.2 * id1 + 0.04 * vgs2 / vds2 -1.14 / vsg1 + 0.04 * vsg3 / vsd5
2.41	3.9	91.1 - 5.91e-4 * (vsg1*id2) / id1 + 119.79 * id1 + 0.03 * vgs2 / vds2 - 0.78 / vsg1 + 0.03 * vsg1 / vsd5 -2.72e-7 / (vds2*vsd5*id1) + 7.11 * (vgs2*vsg4*id2) - 0.37 / vsg5 - 0.58 / vsg3 - 3.75e-6 / id2 - 5.52e-6 / id1

Table II: CAFFEINE-generated models of PM, in order of decreasing error and increasing complexity

Table I shows the symbolic models that provide <10% error. We can examine the equations in more detail to gain an understanding of how design variables in the topology affect performance. For example, A_{LF} is inversely proportional to i_{d1}, the current at the OTA's differential pair. Or, SR_p is solely dependent on i_{d1} and i_{d2} and the ratio i_{d1} / i_{d2}. Or, within the design region sampled, the nonlinear coupling among the design variables is quite weak, typically only as ratios for variables of the same transistor. Or that each expression only contains a (sometimes small) subset of design variables. Or, that transistor pairs M1 and M2 are the only devices affecting five of the six performances (within 10%).

One may improve understanding by examining expressions of varying complexity for a performance characteristic. Low-complexity models will show the macro-effects; alterations to get improved error show how the model is refined to handle details. Table II shows PM models in decreasing training and testing error. A constant of 90.2, while giving 15 % training error, had only 4% test error. For better prediction, CAFFEINE injected two more basis functions; one basis being the current into the differential pair i_{d1}, the other basis, i_{d2} / v_{ds2}, the ratio of current to drain-source voltage at M2. The next model turns the input current term into a ratio i_{d1} / v_{sg1}. Interestingly, and reassuringly, almost all ratios use the same transistor in the numerator and denominator.

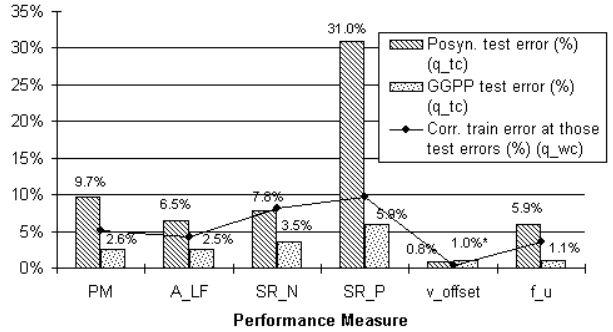

Figure 4: CAFFEINE vs. posynomials

Figure 4 compares CAFFEINE to posynomials [5]. To fairly pick CAFFEINE model, we fixed the training error to what the posynomial achieved, then compared testing errors. In one case they both had low testing error (<1%), but in the remaining 5 cases, CAFFEINE had *2.2x to 5.2x* less testing error. %. What we saw in previous data, and we see again here, is that CAFFEINE has lower testing error than training error, which provides great confidence to the models. In contrast, in all cases but v_{offset}, the posynomials had higher testing error than training error, even on this interpolative data set. With posynomials having weak prediction ability even in interpolation, in comparison to more compact models, one might question the trustworthiness of constraining models of analog circuits to posynomials.

6. Conclusion

This paper presented CAFFEINE, a tool which for the first time can generate interpretable, template-free symbolic models of nonlinear analog circuit performance characteristics. CAFFEINE is built upon genetic programming, but its key is a grammar that restricts symbolic models to a canonical functional form.

CAFFEINE generates a set of models that collectively trade off between error and complexity. Visual inspection of the models demonstrates that the models are interpretable. These models were also shown to be significantly better than posynomials in predicting unseen data.

7. References

[1] G. E. Gielen, "Techniques and Applications of Symbolic Analysis for Analog Integrated Circuits: A Tutorial Overview", in Computer Aided Design of Analog Integrated Circuits And Systems, R.A. Rutenbar et al., eds., IEEE, 2002, pp. 245-261

[2] P. Vancorenland, G. Van der Plas, M. Steyaert, G. Gielen, W. Sansen, "A Layout-aware Synthesis Methodology for RF Circuits," Proc. ICCAD 01, Nov. 2001, p.358

[3] H. Liu, A. Singhee, R.A. Rutenbar, L.R. Carley, "Remembrance of Circuits Past: Macromodeling by Data Mining in Large Analog Design Spaces," Proc. DAC 02, June 2002, pp. 437-442

[4] G. Wolfe, R.Vemuri, "Extraction and Use of Neural Network Models in Automated Synthesis of Operational Amplifiers." IEEE Trans. CAD, Feb. 2003

[5] W. Daems, G. Gielen, and W. Sansen, "An Efficient Optimization-based Technique to Generate Posynomial Performance Models for Analog Integrated Circuits", Proc. DAC 02, June 2002

[6] W. Daems, G. Gielen, W. Sansen, "Simulation-based generation of posynomial performance models for the sizing of analog integrated circuits," IEEE Trans. CAD 22(5), May 2003, pp. 517-534

[7] John R. Koza. Genetic Programming. MIT Press, 1992.

[8] K. Deb, S. Agrawal, A. Pratap, T.A. Meyarivan, "A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II," Proc. PPSN VI, Sept. 2000, pp. 849-858

[9] P. A. Whigham, "Grammatically-based Genetic Programming," Proc. Workshop on GP: From Theory to Real-World Applications, J.R. Rosca, ed., 1995.

[10] F. Leyn, G. Gielen, W. Sansen, "An Efficient Dc Root Solving Algorithm with Guaranteed Convergence for Analog Integrated CMOS Circuits", Proc. ICCAD 98, Nov. 1998