High-Dimensional Statistical Modeling and Analysis of Custom Integrated Circuits

(Invited Paper)

Trent McConaghy
Solido Design Automation Inc.

***Abstract*—Custom circuit designers have long favored manual equation-based approaches in early design stages, because it gives excellent insight and control over the design. However, this flow is threatened: as modern process nodes advance, process variation affects circuit performance more strongly, hurting the accuracy of existing equations. Because designers are typically not statistical modeling experts, it is difficult to adapt the equations to incorporate statistical variations. This paper presents a fast, deterministic technique to help designers revise equations to account for statistical variation. Specifically, the technique extracts compact equations of performance as a function of process variables, even for cases when there are thousands of possible variables and the equations are highly nonlinear. In fact, it provides a whole set of equations that trade off simplicity versus accuracy compared to SPICE. The technique is validated on a broad range of custom integrated circuit modeling problems.**

I. Introduction

With Moore's Law driving the continual shrinking of semiconductor devices [1], small random imperfections in manufacturing are having an increasing effect on circuit performance and therefore yield. As variation issues worsen, variation-aware design is becoming increasingly important in custom integrated circuit design (analog, mixed-signal, memory, and radio frequency circuits).

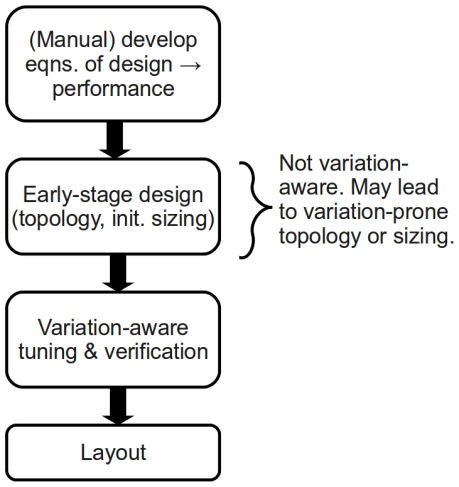

Fig. 1. Status quo design flow.

Figure 1 illustrates a typical custom design flow. It has an early-stage design step, where the designer is developing a or selecting a topology, and performing manual sizing from first-principles equations [2], [3]. Unfortunately, this early equation-based design phase typically ignores variation. Traditionally, this has not been an issue: if variation affected performance up to 10-15%, for most circuits early-stage design could defer variation worries to a follow-up step of variation-aware tuning and verification[1].

But in modern processes, variation can cause performance to vary not just by 10-15%, but by orders of magnitude [4]. Commercial tools cannot help directly: they are designed for follow-up tuning, not making a topology more fundamentally variation-aware. Thus, the initial-stage design may be extremely prone to variation.

To handle massive variation without compromising design performance or yield, early stage designs need to be created with variation-awareness. One may consider manually developing equations that account for process variation. This is a great challenge, because there are so many variables, the designer "rules of thumb" about what matters can change with each new process node, and finally, designers are experts at circuits, not statistical modeling.

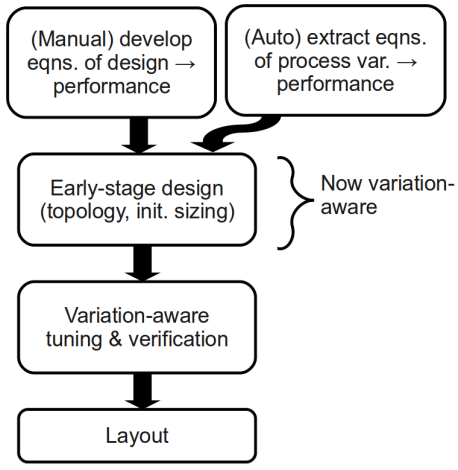

Fig. 2. Proposed design flow.

This paper explores an approach to help designers develop equations in early-stage design that account for process variation. Figure 2 illustrates the flow. Specifically, this paper proposes a fast, scalable, deterministic algorithm called FFX (Fast Function Extraction) to extract interpretable nonlinear

[1]These days, this follow-up step is well supported by commercial variation-aware tools that reconcile corner-based design with verification, with scaling to tens of thousands of devices and millions of simulations.

equations that map process variables to performance. In fact, it provides a whole set of equations: some simple but less accurate, then more equations with increasing accuracy (relative to SPICE) but also increasing complexity. The designer can then pick one of these equations, and merge it with his or her existing design equations. Merging can be as simple as summing the design equation's components and the variation equation's components. With variation-aware equations in hand, the designer can proceed with *variation-aware early-stage design*.

The CAFFEINE equation-extraction tool [5] had similar aims. But it was limited by scalability (≈100 variables), runtime (10 minutes or more), and consistency (stochastic algorithm). In contrast, we will show FFX solve problems with >1000 variables, deterministically in <20 s. A key enabler is a recently-developed technique from the machine learning literature: pathwise regularized learning [7].

This paper verifies the FFX approach on a variety of custom circuits: an opamp, a voltage reference, a bitcell, a sense amp, a GMC filter, and a comparator.

The rest of this paper is organized as follows. Section II mathematically describes the problem. Section III introduces pathwise regularized learning, a key component of FFX. Section IV describes the FFX algorithm in detail. Sections V and VI present the experimental setup and results, respectively. Section VII concludes.

II. Problem Definition

The problem of extracting equations is known as template-free symbolic modeling [5], with inputs and outputs as follows.
Given:

- $\boldsymbol{X}$ and $\boldsymbol{y}$: A set of $\{\boldsymbol{x_j}, y_j\}, j = 1..N$ data samples where $\boldsymbol{x_j}$ is a N_d-dimensional point j and y_j is a corresponding output value. In our application, $\boldsymbol{X}$ will be process variable values from Monte Carlo sampling, and $\boldsymbol{y}$ will be the output values from SPICE-simulating the samples.
- **No** model template

Determine:

- A set of symbolic models (equations) M that provide a tradeoff between minimizing model complexity f_1 and minimizing future model prediction error f_2.

Each model m maps an N_d-dimensional input $\boldsymbol{x}$ to a scalar output value $\hat{y}$, i.e. $\hat{y} = m(\boldsymbol{x})$. Future model prediction error $f_2 = E_{x,y}L(m)$ where $L(m)$ is the squared-error loss function $y - m(\boldsymbol{x}))^2$.

We restrict ourselves to the class of generalized linear models (GLMs) [6]. A GLM is a linear combination of N_B basis functions $B_i; i = 1, 2, ..., N_B$:

$$\hat{y} = m(\boldsymbol{x}) = a_0 + \sum_{i=1}^{N_B} a_i * B_i(\boldsymbol{x}) \quad (1)$$

We measure complexity simply as the number of basis functions (bases) in model m; that is, $complexity(m) = N_B(m)$.

III. Background: Pathwise Regularized Learning

Least-squares (LS) learning aims to find the values for each coefficient a_i in equation (1) that minimize $||\boldsymbol{y} - \boldsymbol{X} * a||^2$, where the $\boldsymbol{X}$ and $\boldsymbol{y}$ are training data. Therefore LS learning aims to minimize training error; it does not acknowledge testing error (future model prediction error). Because it is singularly focused on training error, LS learning may return model coefficients a where a few coefficients are extremely large, making the model overly sensitive to those coefficients. This is overfitting.

Regularized learning aims to minimize the model's sensitivity to overfit coefficient values, by adding minimization terms that are dependent solely on the coefficients: $||a||^2$ or $||a||_1$. This has the implicit effect of minimizing expected future model prediction error. The overall problem formulation is:

$$a^* = minimize\ ||\boldsymbol{y} - \boldsymbol{X} * a||^2 + \lambda_2||a||^2 + \lambda_1||a||_1 \quad (2)$$

Including both regularization terms is an *elastic net* formulation of regularized learning [8][1]. To make the balance between λ_1 and λ_2 explicit, we can set $\lambda_1 = \lambda$ and $\lambda_2 = (1 - \rho) * \lambda$, where λ is now the regularization weight, and ρ is a "mixing parameter."

A *path* of solutions sweeps across a set of possible λ values; returning an a for each λ. Interestingly, we can start at a *huge* value of λ, where all a_i are zero; then work towards smaller λ, uniformly on a log scale. Figure 3 illustrates: the path starts on the far left, and the with λ decreasing (going right), coefficients a_i take nonzero values one at a time.

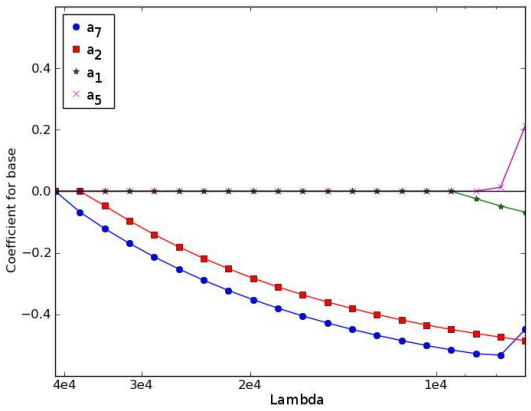

Fig. 3. A path of regularized regression solutions: each vertical slice of the plot gives a vector of coefficient values a for each of the respective basis functions. Going left to right (decreasing λ), each coefficient a_i follows its own path, starting at zero then increasing in magnitude (and sometimes decreasing).

An extremely fast variant of pathwise regularized learning was recently developed / rediscovered: coordinate descent [7].

[1]The middle term (quadratic term, like ridge regression), encourages correlated variables to group together rather than letting a single variable dominate, and makes convergence more stable. The last term (l_1 term, like lasso), drives towards a sparse model with few coefficients, but discourages any coefficient from being too large. $||a||_1 = \sum_i |a_i|$.

At each point on the path, coordinate descent solves for coefficient vector a by: looping through each a_i one at a time, updating the a_i through a trivial formula while holding the rest of the parameters fixed, and repeating until a stabilizes. For speed, it uses "hot starts": at each new point on the path, coordinate descent starts with the previous point's a.

Some highly useful properties of pathwise regularized learning are:

- Learning speed is comparable or better than LS.
- Unlike LS, can learn when there are fewer samples than coefficients $N < n$.
- Can learn thousands or more coefficients.
- It returns a whole *family* of coefficient vectors, with different tradeoffs between number of nonzero coefficients and training accuracy.

For further details, we refer the reader to [7][8].

IV. FFX Algorithm

A. FFX Introduction

The FFX algorithm has these steps:

- First, from a smaller set of input variables, it generates a massive set of basis functions, where each basis function combines one or more interacting nonlinear subfunctions.
- Then, it uses regularized learning to rapidly identify the important basis functions from the set of bases and their corresponding coefficients. In fact, it exploits the special "path-following" property of regularized learning, to identify the best coefficients and bases when there are 0 bases, 1 base, 2 bases, and so on.
- Finally, it filters the candidate functions to a nondominated set that trades off number of bases and error.

For maximum coverage of possible functions, FFX leverages a special technique to include rationals, with negligible extra computational cost.

The rest of this section elaborates upon each the three FFX steps, and the rational functions trick.

B. FFX Step One

Here, FFX generates a massive set of basis functions, where each basis function combines one or more interacting nonlinear subfunctions.

Table I gives the pseudocode. Steps 1-10 generate univariate bases, and steps 11-20 generate bivariate bases (and higher orders of univariate bases). The algorithm simply has nested loops to generate all the bases. The $eval$ function (line 5, 9, and 18) evaluates a base b given input data $\boldsymbol{X}$. The $ok()$ function returns $False$ if any evaluated value is inf, -inf, or NaN, e.g. as caused by divide-by-zero, log on negative values, or negative exponents on negative values. Therefore, ok filters away all poorly-behaving expressions. Line 16 means that expressions of the form $op() * op()$ are not allowed; these are deemed too complex.

TABLE I
STEP ONE: GENERATEBASES()

Inputs: $\boldsymbol{X}$ #input training data
Outputs: $\boldsymbol{B}$ #list of bases

```
# Generate univariate bases
1.  B1 = {}
2.  for each input variable v = {x1, x2, ...}
3.      for each exponent exp = {0.5, 1.0, 2.0 }
4.          let expression b_exp = v^exp
5.          if ok(eval(b_exp, X))
6.              add b_exp to B1
7.              for each operator op = {abs(), log10, ...}
8.                  let expression b_op = op(b_exp)
9.                  if ok(eval(b_op, X))
10.                     add b_op to B1

# Generate interacting-variable bases
11. B2 = {}
12. for i = 1 to length(B1)
13.     let expression b_i = B1[i]
14.     for j = 1 to i - 1
15.         let expression b_j = B1[j]
16.         if b_j is not an operator # disallow op() * op()
17.             let expression b_inter = b_i * b_j
18.             if ok(eval(b_inter, X))
19.                 add b_inter to B2
20. return B = B1 ∪ B2
```

The operators used are: $abs(x)$, $log_{10}(x)$, $max(0, x - thr)$, and $max(0, thr - x)$. The latter two operators are *hinge* operators [9], famously used in multivariate adaptive regression splines (MARS) [10]. Hinge operators add model flexibility, allowing it to "turn off" some regions of input space and focus on remaining regions. For each hinge operator at each variable x_j, we allowed 5 different threshold values thr, uniformly distributed from $minx_j + 0.2 * (maxx_j - minx_j)$ to $minx_j + 0.8 * (maxx_j - minx_j)$; where $minx_j$ and $maxx_j$ are the minimum and maximum values seen for x_j in all training samples.

To scale to hundreds or thousands of input variables, we made a small change to the procedure in Table I: after step 10, do a pilot run of linear learning on the univariate bases and remember the magnitude of the coefficients; then generate interacting-variable bases with priority to the univariate bases having highest-magnitude coefficients; and stop adding bases once a maximum number of bases (e.g. 10000) is exceeded.

C. FFX Step Two

Table II gives the pseudocode. Steps 1-2 create a large matrix $\boldsymbol{X_B}$ which has evaluated input matrix $\boldsymbol{X}$ on each of the basis functions in $\boldsymbol{B}$. Steps 3-4 determine a log-spaced set of N_{lambda} values; see [8] for motivations here. Steps 5-16 are the main work, doing path-following. At each iteration of the loop it performs an pathwise linear fit (line 11) from $\boldsymbol{X_B} \mapsto \boldsymbol{y}$ to find the linear coefficients $\boldsymbol{a}$. A key to the speed of this linear fit is reusing the previous iteration's values for $\boldsymbol{a}$. There are several options to implement the actual fitting; we use coordinate descent [7].

FFX step two is like standard regularized-linear path-following, *except* that whereas the standard approach covers a

whole range of λ such that all coefficients eventually get included, FFX stops as soon as there are more than $N_{max-bases}$ (e.g. 250) nonzero coefficients (line 9).

TABLE II
STEP TWO: PATHWISELEARN()

Inputs: $\boldsymbol{X}$, $\boldsymbol{y}$, $\boldsymbol{B}$ #input data, output data, bases
Outputs: $\boldsymbol{A}$ #list of coefficent-vectors

Compute $\boldsymbol{X_B}$
1. for i = 1 to length($\boldsymbol{B}$)
2. $\quad\boldsymbol{X_B}[\boldsymbol{i}]$ = eval($\boldsymbol{B}[\boldsymbol{i}]$, $\boldsymbol{X}$)

Generate $\boldsymbol{\lambda_{vec}}$ = range of λ values
3. $\lambda_{max} = max(|\boldsymbol{X}^T\boldsymbol{y}|)/(N * \rho)$
4. $\boldsymbol{\lambda_{vec}} = logspace(log_{10}(\lambda_{max} * eps), log_{10}(\lambda_{max}), N_\lambda)$

Main path-following
5. $\boldsymbol{A}$ = {}
6. $N_{bases} = 0$
7. i = 0
8. $\boldsymbol{a}$ = $\{0, 0, \ldots\}$
9. while $N_{bases} < N_{max-bases}$ and i ¡ length($\boldsymbol{\lambda_{vec}}$)
10. $\quad\lambda = \boldsymbol{\lambda_{vec}}[\boldsymbol{i}]$
11. $\quad\boldsymbol{a}$ = pathwiseLinearFit($\boldsymbol{X_B}$, $\boldsymbol{y}$, λ, ρ, $\boldsymbol{a}$)
12. $\quad N_{bases}$ = number of nonzero values in $\boldsymbol{a}$ (not counting offset)
13. $\quad$if $N_{bases} < N_{max-bases}$
14. $\quad\quad$add $\boldsymbol{a}$ to $\boldsymbol{A}$
15. $\quad i = i + 1$
16. return $\boldsymbol{A}$

D. FFX Step Three

Here, FFX filters the candidate functions to a nondominated set that trades off number of bases and error.

Table II gives the pseudocode. Steps 1-8 take the coefficients and bases determined in previous FFX steps, and simply combine them to create a set of candidate models M_{cand}. Steps 9-13 apply standard nondominated filtering to the models, with objectives to minimize complexity and test error.

TABLE III
STEP THREE: NONDOMINATEDFILTER()

Inputs: $\boldsymbol{A}$, $\boldsymbol{B}$ # coefficient vectors, bases
Outputs: M # Pareto-optimal tradeoff of equations

Construct candidate models
1. M_{cand} = {}
2. for i = 1 to length(||$\boldsymbol{A}$||)
3. $\quad\boldsymbol{a} = \boldsymbol{A}[\boldsymbol{i}]$
4. $\quad a_0 = \boldsymbol{a}[\boldsymbol{0}]$ # offset
5. $\quad\boldsymbol{a_{nz}}$ = nonzero values in $\boldsymbol{a}$ (ignoring offset)
6. $\quad\boldsymbol{B_{nz}}$ = expressions in $\boldsymbol{B}$ corr. to nonzero values in $\boldsymbol{a}$
7. $\quad m$ = model(a_0, $\boldsymbol{a_{nz}}$, $\boldsymbol{B_{nz}}$), following eqn. (1)
8. $\quad$add m to M_{cand}

Nondominated filtering
9. $\boldsymbol{f_1}$ = $complexity(m)$ for each m in M_{cand}
10. $\boldsymbol{f_2}$ = $testError(m)$ or $trainError(m)$ for each m in M_{cand}
11. J = nondominatedIndices($\boldsymbol{f_1}, \boldsymbol{f_2}$)
12. $M = M_{cand}[j]$ for each j in J
13. return M

E. Rational Functions Trick

For maximum coverage of possible functions, FFX leverages a special technique inspired by [11] to include rational functions, with negligible extra computational cost. The general idea is: learning the coefficients of a rational function can be cast into a linear regression problem, solved with linear regression, then back-transformed into rational function form. Let us elaborate:

A rational function has the form:

$$\hat{y} = m(\boldsymbol{x}) = \frac{a_0 + \sum_{i=1}^{N_{BN}} a_i * B_i(\boldsymbol{x})}{1.0 + \sum_{i=N_{BN}+1}^{N'_B} a_i * B_i(\boldsymbol{x})} \quad (3)$$

where N'_B is the number of numerator bases (N_{BN}) plus the number of denominator bases (N_{BD}).

Let us perform simple algebraic manipulations to transform this problem. First, we multiply both sides by the denominator:

$$y * \left(1.0 + \sum_{i=N_{BN}+1}^{N'_B} a_i * B_i(\boldsymbol{x})\right) = a_0 + \sum_{i=1}^{N_{BN}} a_i * B_i(\boldsymbol{x}) \quad (4)$$

Then we expand the left-hand side:

$$y + \sum_{i=N_{BN}+1}^{N'_B} a_i * B_i(\boldsymbol{x}) * y = a_0 + \sum_{i=1}^{N_{BN}} a_i * B_i(\boldsymbol{x}) \quad (5)$$

where $B_i(\boldsymbol{x}) * y$ is element-wise multiplication, i.e. $B_i(\boldsymbol{X_j}) * \boldsymbol{y_j}$ for each data point j. Now, subtract to isolate y on the left-hand side:

$$y = a_0 + \sum_{i=1}^{N_{BN}} a_i * B_i(\boldsymbol{x}) - \sum_{i=N_{BN}+1}^{N'_B} a_i * B_i(\boldsymbol{x}) * y \quad (6)$$

Finally, let us define a new set of basis functions.

$$B'_i = \begin{Bmatrix} B_i & i \leq N_{BN} \\ B_i * y & otherwise \end{Bmatrix} \quad (7)$$

At the end of FFX step one, we had N_B basis functions. Before we start step 2, we insert all N_B functions into both the numerator and denominator; therefore $N_{BN} = N_{BD} = N_B$, and $N'_B = 2 * N_B$. We redefine the basis functions according to eqn. (7). Then, all the subsequent FFX steps are performed with these new basis functions. Once the coefficients are found, the final model is extracted by applying the algebraic manipulations in reverse: eqn. (6), then eqn. (5), then eqn. (4).

V. EXPERIMENTAL SETUP

A. Problem Setup

We tested on the circuits and outputs shown in table IV.

For space reasons, we show just a few schematics. The opamp is shown in Figure 4, on TSMC 0.18μm CMOS. Figure 5 gives schematics for the bitcell and sense amp memory circuits. The bitcell's temp=25°C, power supply voltage V_{dd}=1.0 V, and V_{cn}=0.0 V. The sense amp's environmental conditions were: load capacitance C_l=1e-15 F, temp=25°C, and V_{dd}=1.0 V. The technology for the bitcell was TSMC 45nm CMOS, and for the sense amp 28nm CMOS.

TABLE IV
SUMMARY OF TEST PROBLEMS

Circuit	# Devices	# Process variables	Outputs Modeled
opamp	30	215	AV (gain), BW (bandwidth), PM (phase margin), SR (slew rate)
bitcell	6	30	$cell_i$ (read current)
sense amp	12	125	$delay$, pwr (power)
voltage reference	11	105	$DVREF$ (difference in voltage), PWR (power)
GMC filter	140	1468	$ATTEN$ (attenuation), IL
comparator	62	639	BW (bandwidth)

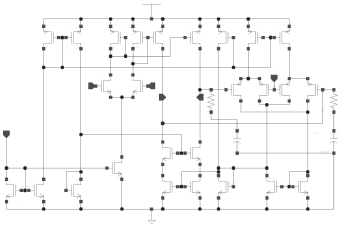

Fig. 4. Opamp schematic

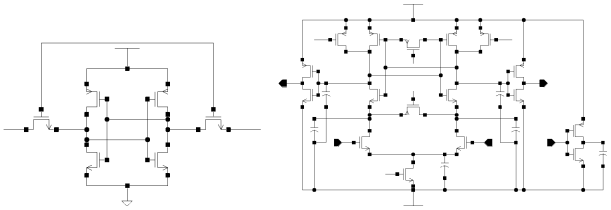

Fig. 5. Left: bitcell schematic. Right: sense amp schematic

Process variation is modeled as a joint probability density function. We use the back-propagation of variance (BPV) model of process variation [12], where random variables are "process variables" which model quantities like N_{sub} (substrate doping concentration). In this model, there are about 10 normal independent identically-distributed (NIID) random variables per transistor, for local variation; along with about 10 NIID global process variables.

To generate data for use by FFX, we apply Monte Carlo sampling and simulation. Specifically: for input points, we draw samples (process points) from this distribution. To generate corresponding outputs, we simulate the process points and apply measure statements. We use HSPICE^{TM}. Each circuit's device sizes were set to have "reasonable" first-cut values by a custom circuit designer, leading to "reasonable" performance values. The opamp and voltage reference had 800 Monte Carlo sample points, the comparator and GMC filter 2000, and bitcell and sense amp 5000.

We calculate normalized mean-squared error on the training data and on the separate testing data: $nmse = \sqrt{\sum_i((\widehat{y}_i - y_i)/(max(\boldsymbol{y}) - min(\boldsymbol{y}))^2)}$. The testing error is ultimately the more important measure, because it measures the model's ability to generalize to unseen data. The separate testing data is chosen as follows: sort the data according to the y-values, then take every 4th point for testing. This is faster than cross-validation, yet gives consistent, reliable answers.

B. FFX Setup

Up to $N_{max-bases}$=250 bases are allowed. Exponents on variables are $x^{1/2}$ (=$\sqrt{(}x)$), x^1 (=x), and x^2. The pathwise learning settings followed good defaults: ρ = 0.5, $\lambda_{max} = max|\boldsymbol{X}^T\boldsymbol{y}|/(N * \rho)$, $eps = 10^{-70}$, and N_λ=1000.

VI. EXPERIMENTAL RESULTS

This section validates the flow by investigating the model fit versus complexity, and actual equations output by FFX.

Each FFX run took 5-20 s on a single-core 1-GHz CPU. Notably, this is orders of magnitude faster than the previous symbolic modeling approach CAFFEINE [5].

A. Test Error

Table V shows the lowest test error found by FFX, compared to standard least-squares linear or quadratic approaches. FFX performed the best, though it did find some outputs challenging.

B. Error Vs. Complexity

Rather than pre-determining what the ideal balance is, FFX extracts a whole set of equations, ranging from the very simple (but higher error) to the very complex (with lower error). Then, the user can examine the tradeoff, and determine which model is most appropriate for his particular design challenge. Figures 6 to 8 show error vs. complexity tradeoffs for representative problems. Each square represents a different model with an associated test error and complexity. For a given subplot, the simplest model is a constant, at the far left. It also has the highest error. As new bases are added (higher complexity) moving to the right, error drops.

The curves have different signatures; we give a representative plot for each. Figure 6 left shows the curve for opamp BW. It has a marked "knee": above the knee at 2%, even small reductions in complexity lead to large increments in error; below the knee, reductions in error add substantial complexity. Opamp AV and opamp SR have similar "knee"-style curves.

Figure 6 right shows the curve for opamp PM. It has no discernable knee, but instead has a smooth tradeoff, a steady reduction in error as complexity increases (until about 20 bases).

TABLE V
TEST ERROR (%) ON THE TEST PROBLEMS. "QUAD (LS)" FAILED WHEN IT HAD TOO FEW SAMPLES FOR THE NUMBER OF COEFFICIENTS.

Approach	opamp $\boldsymbol{AV}$	opamp $\boldsymbol{BW}$	opamp $\boldsymbol{PM}$	opamp $\boldsymbol{SR}$	bitcell $\boldsymbol{cell_i}$	sense amp $\boldsymbol{delay}$
Lin (LS)	1.7	1.3	1.3	3.2	12.7	3.4
Quad (LS)	FAIL	FAIL	FAIL	FAIL	12.5	3.5
FFX	**1.0**	**0.9**	**1.0**	**2.0**	**12.4**	**3.0**
Approach	sense amp $\boldsymbol{pwr}$	voltage reference $\boldsymbol{DVREF}$	voltage reference $\boldsymbol{PWR}$	GMC filter $\boldsymbol{ATTEN}$	GMC filter $\boldsymbol{IL}$	comparator $\boldsymbol{BW}$
Lin (LS)	3.5	2.4	22.8	16.4	17.3	27.2
Quad (LS)	2.9	2.8	40.4	FAIL	FAIL	FAIL
FFX	**2.7**	**1.0**	**2.0**	**7.0**	**8.5**	**17.0**

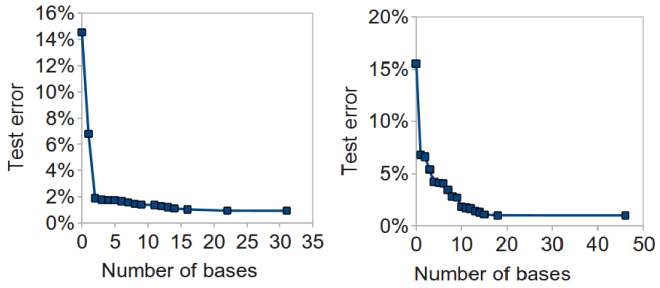

Fig. 6. Error vs. Complexity: opamp BW (left), opamp PM (right).

The curve in Figure 7 left is for bitcell $cell_i$ (read current). It has a soft knee between 2 and 5 bases. Then, error virtually flattens as complexity is increased further. The outputs for comparator BW and voltage reference PWR have similar profiles.

Figure 7 right is for sense amp $delay$. It has a soft knee (at about 20 bases), with decent tradeoffs smoothly extending in both directions. Sense amp pwr has a similar profile.

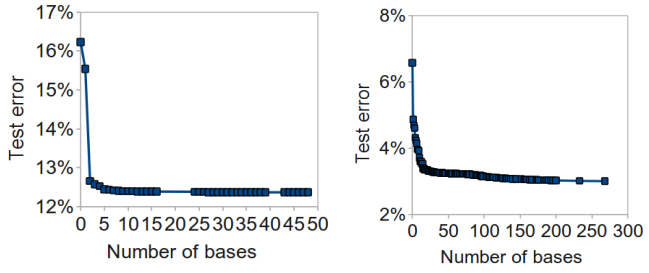

Fig. 7. Error vs. Complexity: bitcell $cell_i$ (left), sense amp $delay$ (right).

Figure 8 left is the profile for voltage reference $DFREF$. Its most remarkable feature is that more than 8 bases did not help; which also means it had a small number of models in the tradeoff.

In contrast, Figure 8 right, for GMC filter $ATTEN$, has up to 250 bases with an equally large number of models, and a very smooth tradeoff between error and complexity. GMC filter IL has a similar profile.

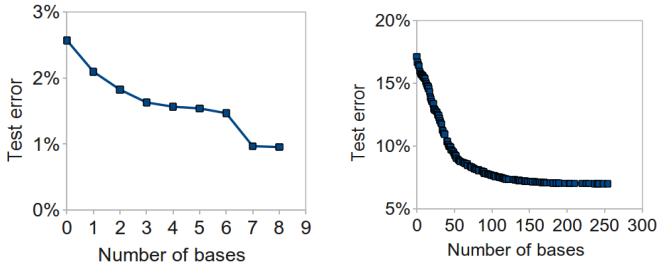

Fig. 8. Error vs. Complexity: voltage reference $DVREF$ (left), GMC filter $ATTEN$ (right).

C. Analysis of Extracted Equations

The core value of FFX comes from inspecting and using the equations extracted by FFX, to make the early-stage custom design flow more variation-aware.

Table VI shows some functions that FFX extracted for opamp PM. At 0 bases is a constant, of course. From 1 to 4 bases, FFX adds one more linear base at a time, gradually adding resolution to the model. At 5 bases, it adds a base that has both an $abs()$ operator, and an interaction term: $abs(dvthn) * dvthn$. It keeps adding bases up to a maximum of 46 bases. By the time it gets to 46 bases, it has actually started using a rational model, as indicated by the $/(1 + \ldots)$ term.

There are two ways for designers to identify the most useful variables and bases:

- First, simply seeing which terms get added first. For opamp PM, the variables came in order dxl, $cgop$, $dvthn$, and $dvthp$. Because they are not subscripted by a device, these indicate global process variables, which modify their respective values in the SPICE model.
- Second, in printing a given function, we order the bases from highest-magnitude coefficient to lowest. In the case of opamp PM, it tends to have the highest coefficients on the variables it adds first (dxl, $cgop$, etc.); though this is not always the case.

We saw the key variables in the equations for opamp PM are global process variables. This means that variation on

TABLE VI
EQUATIONS FOR OPAMP PM, EXTRACTED BY FFX.

# Bases	Test error	Extracted Function
0	15.5	59.6
1	6.8	$59.6 - 0.303 * dxl$
2	6.6	$59.6 - 0.308 * dxl - 0.00460 * cgop$
3	5.4	$59.6 - 0.332 * dxl - 0.0268 * cgop + 0.0215 * dvthn$
4	4.2	$59.6 - 0.353 * dxl - 0.0457 * cgop + 0.0403 * dvthn - 0.0211 * dvthp$
5	4.1	$59.6 - 0.354 * dxl - 0.0460 * cgop - 0.0217 * dvthp + 0.0198 * dvthn + 0.0134 * abs(dvthn) * dvthn$
6	4.07	$59.6 - 0.354 * dxl - 0.0466 * cgop - 0.0224 * dvthp + 0.0202 * dvthn + 0.0135 * abs(dvthn) * dvthn + 0.000550 * DXL$
⋮	⋮	⋮
46	1.0	$(58.9 - 0.136 * dxl + 0.0299 * dvthn - 0.0194 * max(0, 0.784 - dvthn) + \ldots)/(1.0 + \ldots)$

opamp PM is dominated by global variation, rather than local mismatch variation. In contrast, let us look at Table VII, which is the equations for comparator BW. In this case, we see that the variables contributing to the model are local variables (indicated by the subscript pointing to the transistor). Therefore the comparator BW's variation is dominated by local mismatch. Looking deeper, we see that the prominent variables all come from transistor $m1$ or $m2$ in current mirror 1 ($cm1$), and that the particular type of variation is $lint$. Variables dxl_{var0} and $vthp_{var0}$ get added next. However, after 5 bases, the model still does a poor job of explaining the mapping (with error at 17.9%); and even adding significantly more bases it still gets to just 17%.

In many modeling problems, FFX determined that just linear and quadratic terms were appropriate for the best equations. Besides the functions above, these include the GMC filter IL, GMC filter $ATTEN$, opamp SR (for errors $>$ 2.5%), and bitcell $cell_i$[1].

But in some problems, FFX used more strongly nonlinear functions, which of course would be much more challenging for the custom IC designer to develop without the aid of automation. These include the voltage reference $DVREF$, sense amp $delay$, and sense amp pwr. Let us explore some of these.

Table VIII shows some functions that FFX extracted for voltage reference $DVREF$. It always determines that a rational with a constant numerator is the best option. It uses the hinge-style basis functions, including interactions when 3 or more bases are used. It only needs 8 bases (in the denominator) to capture error of 0.9%. Of the 105 possible variables, FFX determined that variable $dvthn$ was highly useful, by reusing it in many ways. $dvthp$ and dxw also had prominence. Once again, the use of global variables indicates that global variation is causing the main variation issues for $DVREF$.

[1]A warning on the bitcell: just 5000 Monte Carlo samples were taken, therefore not stressing the bitcell into its nonlinear high-sigma failure regions.

D. Automated Sensitivity Analysis

As described above, the user can learn about variable sensitivities (impacts) via manual inspection of coefficients and tradeoffs. Another approach is suited to automation: set the contribution of each variable as the sum of the absolute coefficients of bases that use that variable; then normalize. Table IX illustrates impacts for the lowest-error model of opamp PM. Like we found from inspecting the equations, the top-impacting variables are global process variables dxl and $cgop$. Local process variables such as $CM1\_M1\_\_nsmm\_LINT$ are much farther down the list.

TABLE IX
HIGHEST-IMPACT VARIABLES FOR OPAMP PM.

% Impact	Variable Name
46.5%	dxl
10.2%	$cgop$
9.7%	$dvthn$
7.4%	$dvthp$
3.9%	$RCN\_\_nsmm\_DXL$
3.8%	$RCP\_\_nsmm\_DXL$
3.6%	dxw
3.1%	$cgon$
2.3%	$RCP\_\_nsmm\_DXW$
2.1%	$RCN\_\_nsmm\_DXW$
1.1%	$cjswn$
0.8%	cjn
0.7%	$dxlr$
0.5%	$dtoxn$
0.3%	$CM1\_M1\_\_nsmm\_LINT$
0.3%	$dtoxc$
0.3%	$CMB2\_M1\_\_nsmm\_NSUB$
0.3%	$cjswp$
0.2%	$drshrpo$
0.2%	$CMB2\_M1\_\_nsmm\_VFB$
⋮	⋮

TABLE VII
EQUATIONS FOR COMPARATOR BW, EXTRACTED BY FFX.

# Bases	Test error	Extracted Function
0	18.6	$1.72e7$
1	18.3	$1.72e7 - 3.71e5 * x_{cm1,m1,lint} * x_{cm1,m2,lint}$
2	18.3	$1.72e7 - 3.81e5 * x_{cm1,m1,lint} * x_{cm1,m2,lint} + 2327 * x^2_{cm1,m1,lint}$
3	18.1	$1.71e7 - 4.57e5 * x_{cm1,m1,lint} * x_{cm1,m2,lint} + 5.23e4 * x^2_{cm1,m1,lint} + 4.80e4 * x^2_{cm1,m2,lint}$
4	18.0	$1.71e7 - 4.86e5 * x_{cm1,m1,lint} * x_{cm1,m2,lint} + 7.24e4 * x^2_{cm1,m1,lint} + 6.82e4 * x^2_{cm1,m2,lint} - 2.24e4 * dxl_{var0}$
5	17.9	$1.70e7 - 5.22e5 * x_{cm1,m1,lint} * x_{cm1,m2,lint} + 9.80e4 * x^2_{cm1,m1,lint} + 9.26e4 * x^2_{cm1,m2,lint} - 5.40e4 * dxl_{var0} + 2.54e4 * dvthp_{var0}$

TABLE VIII
EQUATIONS FOR VOLTAGE REFERENCE $DVREF$, EXTRACTED BY FFX.

# Bases	Test error	Extracted Function
0	2.6	512.7
1	2.1	$504/(1.0 + 0.121 * max(0, dvthn + 0.875))$
2	1.8	$503 - 199 * max(0, dvthn + 1.61) - 52.1 * max(0, dvthn + 0.875)$
3	1.6	$496/(1.0 - 0.0447 * max(0, -1.64 - dvthp) * max(0, dvthn + 0.875) - 0.0282 * max(0, -1.90 - dxw) * max(0, dvthn + 0.875) - 0.0175 * max(0, -1.64 - dvthp) * max(0, dvthn + 0.142))$
⋮	⋮	⋮
8	0.9	$476/(1.0 + 0.105 * max(0, dvthn + 1.61) - 0.0397 * max(0, -1.64 - dvthp) * max(0, dvthn + 0.875) - 0.0371 * max(0, -1.90 - dxw) * max(0, dvthn + 0.875) - 0.0151 * max(0, -1.64 - dvthp) * max(0, dvthn + 0.142) \ldots)$

VII. CONCLUSION

This paper proposed a flow for custom IC designers to address process variation earlier in the design flow. The aim is to augment the early-stage design equations with variation-awareness. To meet that aim, this paper proposed an algorithm called FFX, that automatically extracts a set of nonlinear equations that trade off complexity versus error. The equations can include interactions, and strongly nonlinear functions such as rationals and $max(0, x - thr)$. FFX extracts these equations quickly and deterministically. With chosen equations to draw from, the designer can then augment his or her existing equations with variation awareness, enabling a more variation-aware early-stage design flow.

The FFX algorithm was enabled by a recent machine learning technique, coordinate-descent to solve pathwise regularized linear regression [7]. FFX exploits the structure of regularization path-following for early stopping.

FFX was verified on a variety of custom circuits: an opamp, a voltage reference, a bitcell, a sense amp, a GMC filter, and a comparator, having up to 1468 input variables.

Compared to the state-of-the-art symbolic modeling method CAFFEINE [5], FFX was shown to scale to 10x more input variables, run 30x faster, and has reliable deterministic (vs. stochastic) convergence.

ACKNOWLEDGMENT

Funding for the reported research results is acknowledged from Solido Design Automation Inc.

REFERENCES

[1] ITRS_authors, "International technology roadmap for semiconductors," ITRS, Tech. Rep., 2010. [Online]. Available: http://www.itrs.net/Links/2010ITRS/Home2010.htm
[2] W. Sansen, *Analog Design Essentials*. Springer, 2006.
[3] B. Razavi, *Design of Analog CMOS Integrated Circuits*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 2001.
[4] S. Nassif, "Overcoming cmos reliability challenges: From devices to circuits and systems," in *Workshop, Design Automation and Test Europe (DATE)*, March 2011.
[5] T. McConaghy and G. G. E. Gielen, "Template-free symbolic performance modeling of analog circuits via canonical-form functions and genetic programming," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 28, pp. 1162–1175, August 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1669822.1669827
[6] J. A. Nelder and R. W. M. Wedderburn, "Generalized linear models," *Journal of the Royal Statistical Society, Series A, General*, vol. 135, pp. 370–384, 1972.
[7] J. H. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *Journal of Statistical Software*, vol. 33, no. 1, pp. 1–22, 2 2010. [Online]. Available: http://www.jstatsoft.org/v33/i01
[8] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal Of The Royal Statistical Society Series B*, vol. 67, no. 2, pp. 301–320, 2005. [Online]. Available: http://ideas.repec.org/a/bla/jorssb/v67y2005i2p301-320.html
[9] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2001.
[10] J. H. Friedman, "Multivariate adaptive regression splines," *Annals of Statistics*, vol. 19, no. 1, pp. 1–141, 1991.
[11] H. Leung and S. Haykin, "Rational function neural network," *Neural Comput.*, vol. 5, pp. 928–938, November 1993. [Online]. Available: http://portal.acm.org/citation.cfm?id=188045.188060
[12] P. Drennan and C. McAndrew, "A comprehensive mosfet mismatch model," in *Proc. International Electron Devices Meeting*, 1999.